应用密码学高级课程

Dan Boneh
Victor Shoup

Jan. 2020

前言

密码学是用于保护计算机系统中信息的一个不可或缺的工具。它无处不在，每天被全世界数十亿人用于保护静态的或动态的数据。密码学系统是很多标准协议的重要组成部分，比如说非常知名的运输层安全 (TLS) 协议，它们使得把强大的密码学技术纳入到各式各样的应用当中变成一件相当容易的事情。

尽管密码学非常有用，它往往也是非常脆弱的。最安全的密码学系统也会因为一个规范上或编程上的错误而完全丧失安全性，而且无论我们进行多少次单元测试都无法发现密码学系统的安全漏洞。相反，为了论证一个密码学系统是安全的，我们得依靠数学建模和证明来说明某个系统满足指定的安全属性。我们经常需要引入某些合理的假设来推动我们的安全论证。

本书正是关于这一点：构建实用的密码学系统，并在合理的假设下论证其安全性。本书涵盖了许多针对密码学中不同任务的构造。对于每一项任务，我们都会定义一个精确的安全目标，然后提出实现所需目标的构造。为了分析这些结构，我们会发展一套统一的框架来进行密码学证明。掌握了这个框架的读者将有能力将其应用于书中可能没有提到的其他新构造。

在整本书中，我们通过展示案例来调查已经实际部署的系统是如何运作的。我们会描述读者需要避免的常见错误以及对现实世界系统的常见攻击方式，并借此说明密码学中严谨的重要性。在每一章的结尾，我们都会给出一个有趣的应用，它们会以某种意想不到的方式践行本章的观点。

目标受众以及本书的使用

本书旨在自成一体。我们在附录中提供了一些包含概率论和代数学的基本知识作为补充材料。本书主要分为三部分：

- 第一部分将介绍**对称加密**，它会试图解释，当 Alice 和 Bob 两方拥有攻击者未知的共享密钥时，如何安全地交换信息。我们会讨论数据机密性、数据完整性以及认证加密的重要概念。

- 第二部分将介绍**公钥加密**和**数字签名**的概念，这些技术使得 Alice 和 Bob 可以在没有预先共享密钥的情况下安全地进行通信。

- 第三部分关于**密码学协议**，比如关于用户身份识别、密钥交换、零知识和安全计算的协议。

初学者可以通过阅读本书来了解密码学系统是如何工作的以及为什么它们是安全的。书中的每一个安全定理后面都附有一个证明思路，它们会从高层次上解释了为什么该方案是安全的。初读时，为了不丢失连续性，读者可以跳过详细的证明。那些探讨某些定义的细微差别的数学细节章节也可以在第一次阅读时跳过。

进阶读者可能会喜欢阅读详细的证明，以学习如何在密码学中进行证明。我们在每一章的结尾都提供了很多练习，它们会探讨该章所涉及的内容的其他方面。有些练习是对所学知识的复习，但更多的练习是对内容的扩展，以及讨论本章所未涉及的主题。

本书的状态

目前的草稿包含第一部分和第二、三部分的大部分内容。最后两章即将问世。我们希望你喜欢这本书。如果你发现错别字或错误，请给我们提出意见。

引用. 目前的草稿虽然已基本完成，但本书参考的许多已有工作和参考文献尚未被全部纳入。这些内容很快就会被添加进来，我们会在每章末尾的 “笔记” 一节列举该章所涉及的参考文献。

Dan Boneh 和 Victor Shoup
2020 年 1 月

目录

第一章　简介

1.1　密码的历史

1.2　本书的术语简介

第一部分

私钥密码学

第二章　加密

假设 Alice 和 Bob 共享一个密钥 k，Alice 想通过网络向 Bob 传输一条消息 m，同时在有窃听对手的情况下保持 m 的机密性。本章初步介绍解决这一问题的基本技术。除了在网络上传输消息外，这些技术还允许 Alice 在磁盘上存储一个文件，使其他能访问磁盘的人无法读取该文件，但 Alice 自己可以在之后读取该文件。

我们强调，尽管我们在本章中介绍的用以解决这一基本问题的技术是重要且有趣的，但它们本身并不能解决与“安全通信”有关的所有问题：

- 这些技术只在 Alice 使用每个密钥传输单一消息的情况下提供机密性。如果 Alice 想用同一个密钥传输多条消息，那么她必须使用第五章中介绍的方法。

- 这些技术没有提供任何对*消息完整性*的保证：如果攻击者有能力在密文从 Alice 到 Bob 的传输过程中修改它的比特，那么 Bob 可能无法意识到这一点，并接受一个与 Alice 发送的原文不同的消息。我们将在第六章中讨论提供消息完整性的技术。

- 这些技术并没有提供一种让 Alice 和 Bob 共享密钥的机制。也许他们能够在某个时间点使用一些安全的网络（或物理的、面对面的会面）来共享密钥，然后在之后的某个时间点发送消息。这时 Alice 和 Bob 就可以通过一个不安全的网络进行通信。然而，只要有适当的基础设施，也有一些协议允许 Alice 和 Bob 通过不安全的信道交换密钥，我们将在第二十一章讨论这类协议。

2.1　香农密码与完美安全性

2.1.1　香农密码的定义

使用共享密钥对信息进行加密的基本机制被称为*密码 (cipher)* 或*加密方案 (encryption scheme)*。在本节中，我们将介绍一个略微简化的密码概念，称为**香农密码 (Shannon cipher)**。

一个**香农密码**是一个函数对 $\mathcal{E} = (E, D)$，其中：

- 函数 E（**加密函数**）接受一个**密钥 (key)** k 和一条**消息 (message)** m（也称作**明文 (plaintext)**）作为输入，输出一条**密文 (ciphertext)** c，即：

$$c = E(k, m)$$

我们称 c **是** m **在** k **下的加密**。

- 函数 D（**解密函数**）接受一个密钥 k 和一条密文 c 作为输入，输出一条消息 m，即：

$$m = D(k, c)$$

我们称 m **是** c **在** k **下的解密**。

- 我们要求解密能够 “抵消” 加密；也就是说，密码必须满足这样的**正确性属性**：对于所有的密钥 k 和所有的消息 m，都有：

$$D(k, E(k, m)) = m$$

更正式地说，我们假设 $\mathcal{K}$ 是所有密钥的取值集合（**密钥空间**）；$\mathcal{M}$ 是所有消息的取值集合（**消息空间**）；$\mathcal{C}$ 是所有密文的取值集合（**密文空间**）。基于以上表记，我们可以记：

$$E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$$
$$D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$$

此外，我们可以称密码 $\mathcal{E}$ **定义在** $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ **上**。

假设 Alice 和 Bob 想使用这样一个密码来保护他们发送的消息。我们的想法是，Alice 和 Bob 必须以某种方式事先就密钥 $k \in \mathcal{K}$ 达成一致。假设做到了这一点，那么当 Alice 想向 Bob 发送一条消息 $m \in \mathcal{M}$ 时，她使用密钥 k 对 m 进行加密，得到密文 $c = E(k, m) \in \mathcal{C}$，然后通过某种通信网络将 c 发送给 Bob。Bob 收到 c 后同样使用 k 对 c 进行解密，正确性属性会确保 $D(k, c)$ 与 Alice 的原始消息 m 相同。要做到这一点，我们必须假设 c 在从 Alice 到 Bob 的传输过程中没有被篡改。当然，从直觉上讲，我们的目标是，一个窃听者在传输过程中可能获得 c，但不会了解到太多关于 Alice 的消息 m 的信息，这个直观的概念就是我们在下面探讨的安全的正式定义所要体现的。

在实践中，密钥、消息和密文通常都是字节序列。密钥通常有一定的固定长度，例如 16 字节（即 128 位）的密钥非常普遍。消息和密文可以是某种固定长度的字节序列，也可以是可变长的。例如，消息可以是一个 1 GB 的视频文件，一个 10 MB 的音乐文件，一条 1 KB 的电子邮件，甚至是在电子选举中编码为 “是” 或 “否” 的单一比特。

密钥、消息和密文也可以是其他类型的数学对象，比如整数或整数序列（也许位于某个特定的区间），或其他更复杂的数学对象类型（多项式、矩阵或群元素）。不管这些数学对象有多花哨，在实践中，它们必须能在某些时候被表示为字节序列，以便在计算机中存储和传输。

为了简单起见，在我们对密码的数学处理中，我们将假设 $\mathcal{K}$、$\mathcal{M}$ 和 $\mathcal{C}$ 是有限大小的集合。虽然这简化了理论，但这意味着如果一个现实世界的系统允许长度不受限制的消息，我们将（有点人为地）对合法的消息长度施加一个（大的）上限。

为了强化对上述术语的理解，我们再看一下第一章中讨论的一些密码实例。

例 2.1. 一个**一次性密码本 (one-time pad)** 是一个香农密码 $\mathcal{E} = (E, D)$，其密钥、消息和密文是相同长度的比特序列；也就是说，$\mathcal{E}$ 定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上：

$$\mathcal{K} := \mathcal{M} := \mathcal{C} := \{0,1\}^L$$

其中 L 是一个固定参数。对于一个密钥 $k \in \{0,1\}^L$ 和一条消息 $m \in \{0,1\}^L$，加密函数的定义为：

$$E(k, m) := k \oplus m$$

对于一个密钥 $k \in \{0,1\}^L$ 和一条密文 $c \in \{0,1\}^L$，解密函数的定义为：

$$D(k,c) := k \oplus c$$

其中，“$\oplus$” 表示按位异或，也即按位模 2 加法。对于任意比特序列 $x, y, z \in \{0,1\}^L$，都有：

$$x \oplus y = y \oplus x, \quad x \oplus (y \oplus z) = (x \oplus y) \oplus z, \quad x \oplus 0^L = x, \quad x \oplus x = 0^L$$

我们很容易从模 2 加法的相应属性中推导出上述属性。利用这些属性，我们不难验证正确性属性对 $\mathcal{E}$ 是成立的，因为对于所有 $k, m \in \{0,1\}^L$，都有：

$$D(k, E(k,m)) = D(k, k \oplus m) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0^L \oplus m = m$$

在这种情况下，加密和解密函数恰好是相同的，但是当然，并非所有的密码都有这种特性。

例 2.2. 一个**变长一次性密码本 (variable length one-time pad)** 是一个香农密码 $\mathcal{E} = (E, D)$，其中，密钥是某些固定长度 L 的比特序列，而消息和密文是变长的比特序列，它们的长度最大为 L。因此，$\mathcal{E}$ 定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上，且：

$$\mathcal{K} := \{0,1\}^L, \quad \mathcal{M} := \mathcal{C} := \{0,1\}^{\leq L}$$

L 是一个固定参数。这里，$\{0,1\}^{\leq L}$ 表示所有不长于 L 的比特序列的集合（包括空序列）。对于一个密钥 $k \in \{0,1\}^L$ 和一个长度为 ℓ 的消息 $m \in \{0,1\}^{\leq L}$，加密函数定义如下：

$$E(k,m) := k[0 \ldots \ell - 1] \oplus m$$

而对于密钥 $k \in \{0,1\}^L$ 和一个长度为 ℓ 的密文 $c \in \{0,1\}^{\leq L}$，解密函数定义如下：

$$D(k,m) := k[0 \ldots \ell - 1] \oplus c$$

这里，$k[0 \ldots \ell - 1]$ 表示将 k 截断到其前 ℓ 位。读者可以自行验证 $\mathcal{E}$ 的正确性属性是成立的。

例 2.3. 一个**置换密码 (substitution cipher)** 是一个具有如下形式的香农密码 $\mathcal{E} = (E, D)$。令 Σ 是一个有限符号表（例如字母 `A-Z`，加上一个空格符 ␣）。消息空间 $\mathcal{M}$ 和密文空间 $\mathcal{C}$ 都是来自 Σ 的某个固定长度 L 的符号序列，即：

$$\mathcal{M} := \mathcal{C} := \Sigma^L$$

密钥空间 $\mathcal{K}$ 包含 Σ 上的所有全排列；也就是说，每个 $k \in \mathcal{K}$ 都是 Σ 上的一个双射。注意，$\mathcal{K}$ 是一个非常大的集合；事实上 $|\mathcal{K}| = |\Sigma|!$（对于 $|\Sigma| = 27$，$|\mathcal{K}| \approx 1.09 \times 10^{28}$）。

用密钥 $k \in \mathcal{K}$（Σ 的一个全排列）加密一条消息 $m \in \Sigma^L$ 的加密函数定义如下：

$$E(k,m) := \big(k(m[0]), k(m[1]), \ldots, k(m[L-1])\big)$$

其中 $m[i]$ 表示 m 的第 i 项（下标从零开始），$k(m[i])$ 表示对符号 $m[i]$ 的做 k 置换后的结果。因此，要用 k 加密 m，我们只需将置换 k 分别按顺序应用于序列 m 中的每一项。使用密钥 $k \in \mathcal{K}$ 解密一条密文 $c \in \Sigma^L$ 的解密函数定义如下：

$$D(k,c) := \big(k^{-1}(c[0]), k^{-1}(c[1]), \ldots, k^{-1}(c[L-1])\big)$$

这里，k^{-1} 是 k 的逆置换。为了用 k 解密 c，我们只需将 k^{-1} 分别按顺序应用于序列 c 中的每一项。置换密码的正确性属性很容易验证：对于一条消息 $m \in \Sigma^L$ 和密钥 $k \in \mathcal{K}$，我们有：

$$\begin{aligned} D(k, E(k,m)) &= D(k, (k(m[0]), k(m[1]), \ldots, k(m[L-1])) \\ &= (k^{-1}(k(m[0])), k^{-1}(k(m[1])), \ldots, k^{-1}(k(m[L-1]))) \\ &= (m[0], m[1], ..., m[L-1]) = m \end{aligned}$$

例 2.4 (加性一次性密码本). 我们还可以定义一个 “加法模 n” 的一次性密码本的变体。它是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的密码 $\mathcal{E} = (E, D)$，其中 $\mathcal{K} := \mathcal{M} := \mathcal{C} := \{0, \ldots, n-1\}$，$n$ 是一个正整数。加密和解密的定义如下：

$$E(k,m) = m + k \mod n, \quad D(k,c) = c - k \mod n$$

读者很容易验证正确性属性对 $\mathcal{E}$ 成立。

2.1.2 完美安全性

到目前为止，我们只是定义了香农密码的基本语法和正确性要求。接下来我们将讨论一个问题：什么才是 “安全” 的密码？直观地说，答案是，一个安全的密码是即使能够看到加密过程，也能在加密后保持对消息的 “良好隐藏” 的密码。然而，把这个直观的答案变成一个既具有数学意义又具有实际意义的答案是一个真正的挑战。事实上，尽管密码已经被使用了几个世纪，但数学上可接受的安全性定义在最近的几十年才刚刚被归纳出来。

在本节，我们将阐述**完美安全性 (perfect security)** 的数学概念，这是安全学的一个黄金标准（至少在我们只关心加密单一消息并且不关心其完整性时）。我们还将看到，实现这种安全水平是可能的；事实上，我们将表明，一次性密码本就满足这个定义。然而一次性密码本不是很实用，因为它的密钥必须和消息一样长：如果 Alice 想发送一个 1 GB 的文件给 Bob，他们必须已经共享了一个长达 1 GB 的密钥！这是不现实的。不幸的是，这一点无法避免：我们还将证明，任何具备完美安全性的密码都必须有一个至少与它的消息空间一样大的密钥空间。这个事实为我们构造一个更弱的安全性定义提供了动力，它应当允许我们使用更短的密钥来加密长的消息。

如果 Alice 使用一个密钥 k 加密一条消息 m，而一个窃听的对手获得了密文 c，Alice 只有在密钥 k 难以猜测的情况下才有希望保持 m 的秘密，而这至少意味着密钥 k 应该从一个大的密钥空间中随机选出。说 m “隐藏得很好”，至少得意味着在不知道 k 的情况下，很难从 c 中完全确定 m；然而，这其实是不够的。即使对手可能不知道 k，我们假设他知道加密算法和 k 的分布。事实上，我们假设当一个信息被加密时，密钥 k 总是从密钥空间的所有密钥中随机均匀选出。对手也可能对被加密的消息有一些了解，考虑到具体情况，他可能可以将消息的取值空间缩小到一个较小的范围，而且他可能对每个可能的消息被选出的可能性有一定的了解。比方说，假设他知道消息 m 可能是 $m_0 =$ `"ATTACK␣AT␣DAWN"` 或者 $m_1 =$ `"ATTACK␣AT␣DUSK"`。根据对手现有的情报，Alice 同样可能选择这两个消息中的任何一个。在没有看到密文 c 的情况下，对手只有 50% 的机会猜到 Alice 发送的是哪条信息。但我们假设对手确实知道 c。即使有这样的知识，两个信息仍然都有可能；也就是说，可能存在密钥 k_0 和 k_1 使得 $E(k_0, m_0) = c$ 和 $E(k_1, m_1) = c$ 都成立，所以他还是不能确定 $m = m_0$ 还是 $m = m_1$。但是，他还是可以猜测的。假如说密码有这样一个属性，有 800 个密钥 k_0 使得 $E(k_0, m_0) = c$ 成立，有 600 个密钥 k_1 使得 $E(k_1, m_1) = c$ 成立。那么，这样的猜测正确的概率就等于 $800/(800 + 600) \approx 57\%$，这比他在不知道密文时只有 50%

的可能性要高。我们对完美安全性的正式定义明确地排除了这样的一种可能性，即对密文的了解能够增加猜测原始消息的概率，或者确定消息明文的任何属性。

闲话少说，我们下面正式定义完美安全性。在这个定义中，我们将考虑一个概率实验，其中密钥是从密钥空间中随机均匀选取的。我们记 k 为代表这个随机密钥的随机变量。对于一个消息 m，$E(\mathsf{k}, m)$ 是另一个随机变量，它代表将加密函数应用于随机密钥 k 和消息 m。因此，每个消息 m 都会产生一个不同的随机变量 $E(\mathsf{k}, m)$。

定义 2.1 (完美安全性). 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的香农密码。考虑一个概率实验，其中随机变量 k 均匀分布在 $\mathcal{K}$ 上。如果对于所有的 $m_0, m_1 \in \mathcal{M}$ 和所有的 $c \in \mathcal{C}$，都有：

$$\Pr[E(\mathsf{k}, m_0) = c] = \Pr[E(\mathsf{k}, m_1) = c]$$

我们就说 $\mathcal{E}$ 是一个**完美安全的**香农密码。

下面，我们将探讨一些完美安全性的等价表述。

定理 2.1. 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的香农密码。以下表述是等价的：

(i) $\mathcal{E}$ 是完美安全的。

(ii) 对于任意 $c \in \mathcal{C}$，存在 N_c（可能取决于 c）使得对于所有 $m \in \mathcal{M}$，都有：

$$|\{k \in \mathcal{K} : E(k, m) = c\}| = N_c$$

(iii) 如果随机变量 k 均匀分布在 $\mathcal{K}$ 上，那么对于 $m \in \mathcal{M}$，每个随机变量 $E(\mathsf{k}, m)$ 的分布都相同。

证明. 首先，让我们把 (ii) 重述如下：对于每个 $c \in \mathcal{C}$，都存在一个 P_c（取决于 c）使得对于所有 $m \in \mathcal{M}$，都有 $\Pr[E(\mathsf{k}, m) = c] = P_c$。这里，$\mathsf{k}$ 是一个均匀分布在 $\mathcal{K}$ 上的随机变量。注意到 $P_c = N_c/|\mathcal{K}|$，其中 N_c 与 (ii) 的原始表述一致。

这个版本的 (ii) 显然与 (iii) 相同。

(i) $\Longrightarrow$ *(ii)*。假设 (i) 成立，我们证明 (ii)。为了证明 (ii)，令 $c \in \mathcal{C}$ 是某个固定密文。挑选某个任意消息 $m_0 \in \mathcal{M}$，并令 $P_c := \Pr[E(\mathsf{k}, m_0) = c]$。根据 (i)，我们知道对于所有的 $m \in \mathcal{M}$，我们都有 $\Pr[E(\mathsf{k}, m) = c] = Pr[E(\mathsf{k}, m_0) = c] = P_c$，这就证明了 (ii)。

(ii) $\Longrightarrow$ *(i)*。假设 (ii) 成立，我们证明 (i)。考虑任意固定的 $m_0, m_1 \in \mathcal{M}$ 和 $c \in \mathcal{C}$，(ii) 表明 $\Pr[E(\mathsf{k}, m_0) = c] = P_c = \Pr[E(\mathsf{k}\ m_1) = c]$，这就证明了 (i)。 □

正如我们所承诺的，我们下面给出一个证明，证明一次性密码本（见例 2.1）是完美安全的。

定理 2.2. 一次性密码本是一种完美安全的香农密码。

证明. 假设香农密码 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的一次性密码本，其中 $\mathcal{K} := \mathcal{M} := \mathcal{C} := \{0,1\}^L$。对于任意固定消息 $m \in \{0,1\}^L$ 和密文 $c \in \{0,1\}^L$，都存在一个唯一密钥 $k \in \{0,1\}^L$ 满足：

$$k \oplus m = c$$

即 $k := m \oplus c$。因此 $\mathcal{E}$ 满足定理 2.1 中的 (ii)（对每个 c 都有 $N_c = 1$）。 □

例 2.5. 再考虑一下例 2.2 中定义的变长一次性密码本。这种密码并不符合我们对完美安全的定义，因为一个密文的长度与相应的明文相同。事实上，如果我们选择一个长度为 1 的任意字符串 m_0 以及一个长度为 2 的任意字符串 m_1。此外，假设 c 是一个长度为 1 的任意字符串，而 k 是一个在密钥空间均匀分布的随机变量。那么我们有：

$$\Pr[E(\mathsf{k}, m_0) = c] = 1/2, \qquad \Pr[E(\mathsf{k}, m_1) = c] = 0$$

这恰好是定理 2.1 的一个直接的反例。

直观地说，变长一次性密码本不能满足我们对完美安全的定义，因为任何密文都会泄露相应的明文的*长度*。然而，在某种意义上（我们现在尚不明确），这确实是*唯一*泄露的信息，因此，不清楚这应该被看作是密码的问题，还是我们对完美安全的定义的问题。一方面，我们可以想象消息长度可能会有很大变化的场景，虽然我们总是可以通过 “填充” 的手段来把较短的消息补充到同样的长度，但从实际的角度来看，这可能是不可接受的，因为这会造成带宽上的浪费。另一方面，我们必须意识到，在某些应用中，仅仅泄露信息的长度也可能是危险的：如果你正在加密一个问题的 “yes” 或 “no” 的答案，那么仅仅是这些字符串的 ASCII 编码长度就会泄露*一切*，所以你最好把 “no” 填充到三个字符长。

例 2.6. 再考虑一下例 2.3 中定义的置换密码的情况。有几种不同的方法可以看出这个密码也不是完美安全的。

比方说，选择一对消息 $m_0, m_1 \in \Sigma^L$，其中 m_0 的前两项是相等的，而 m_1 的前两项不想等，即：

$$m_0[0] = m0[1], \quad m_1[0] \neq m_1[1]$$

那么对于每个密钥 k，也就是 Σ 上的一种置换，如果 $c = E(k, m_0)$，那么必然有 $c[0] = c[1]$；而如果 $c = E(k, m_1)$，那么也必然有 $c[0] \neq c[1]$。由此可见，如果 k 在密钥空间上是均匀分布的，那么 $E(\mathsf{k}, m_0)$ 和 $E(\mathsf{k}, m_1)$ 的分布就不会相同。

有的人可能会认为上面描述的弱点显得有些刻意，但还有一种更加现实的对替换密码的攻击方式。假设置换密码被用来加密电子邮件信息。众所周知，电子邮件以一个 “标准首部” 开始，如 `"FROM"`。假设密文 $c \in \Sigma^L$ 被对手截获。因为密钥 k 其实就是 Σ 的一个排列组合。对手知道：

$$c[0 \ldots 3] = (k(\mathtt{F}), k(\mathtt{R}), k(\mathtt{O}), k(\mathtt{M}))$$

因此，如果原始消息 $m \in \Sigma^L$，对手现在可以找到 m 中所有出现 `F`、`R`、`O` 和 `M` 的位置。仅仅根据这些信息，再加上关于消息的具体上下文信息和关于字母频率的一般信息，对手就可能能够推断出关于原始消息的相当多的信息。

例 2.7. 考虑例 2.4 中定义的加性一次性密码本的情况。很容易验证加性一次性密码本是完美安全的。事实上，它满足定理 2.1 中的条件 (ii)（对每个密文 c 都有 $N_c = 1$）。

下面要介绍的两个定理进一步阐述了完美安全性的另外两个特征。对于前一个定理，假设一个偷听的对手将某个谓词 ϕ 应用于他所获得的密文。谓词 ϕ（密文空间上的一个布尔函数）的逻辑可能非常简单，比如奇偶判断函数（判断密文中比特 1 的数量是偶数还是奇数），也可能是类型更加复杂的统计测试。但无论谓词 ϕ 是简单还是复杂，完美安全性都会保证谓词在密文上的变换也不会透露出任何关于明文消息的信息。

定理 2.3. 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的香农密码。考虑一个概率实验，其中 k 是一个在 $\mathcal{K}$ 上均匀分布的随机变量。那么，当且仅当对于 $\mathcal{C}$ 上的每个谓词 ϕ 和所有的 $m_0, m_1 \in \mathcal{M}$，我们都有：

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \Pr[\phi(E(\mathsf{k}, m_1))]$$

时，$\mathcal{E}$ 是完美安全的。

证明. 这实际上只是一个简单的计算。一方面，假设 $\mathcal{E}$ 是完美安全的，并给定 ϕ，m_0 和 m_1，令 $S := \{c \in \mathcal{C} : \phi(c)\}$，那么我们有：

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \sum_{c \in S} \Pr[E(\mathsf{k}, m_0) = c] = \sum_{c \in S} \Pr[E(\mathsf{k}, m_1) = c] = \Pr[\phi(E(\mathsf{k}, m_1))]$$

这里，我们在建立第二个等号时使用了 $\mathcal{E}$ 是完美安全的假设。另一方面，假设 $\mathcal{E}$ 不是完全安全的，那么必然存在 m_0，m_1 和 c 使得：

$$\Pr[E(\mathsf{k}, m_0) = c] \neq \Pr[E(\mathsf{k}, m_1) = c]$$

不妨定义谓词 ϕ 对该特定密文 c 输出真，对所有其他密文都输出假，我们可以发现：

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \Pr[E(\mathsf{k}, m_0) = c] \neq \Pr[E(\mathsf{k}, m_1) = c] = \Pr[\phi(E(\mathsf{k}, m_1))] \qquad \square$$

下一个定理以另一种方式说明，完美安全性保证了密文不会泄露任何关于明文消息的信息。假设 m 是分布在消息空间 $\mathcal{M}$ 上的一个随机变量，我们并没有假设 m 是均匀分布在 $\mathcal{M}$ 上的。现在假设 k 是均匀分布在密钥空间 $\mathcal{K}$ 上的一个随机变量，与 m 无关，并定义 $\mathsf{c} := E(\mathsf{k}, \mathsf{m})$ 是分布在密文空间 $\mathcal{C}$ 上的一个随机变量。那么下面的定理将表明，完美安全性会保证随机变量 c 和 m 相互独立。

描述这种独立性的一种方法是，对于每个以非零概率出现的密文 $\mathsf{c} \in \mathcal{C}$ 和每个消息 $\mathsf{m} \in \mathcal{M}$，我们都有：

$$\Pr[\mathsf{m} = m \mid \mathsf{c} = c] = \Pr[\mathsf{m} = m]$$

直观上说，这意味着在看到一个密文后，我们对消息的了解不会比在看到密文之前更多。

另一种描述这种独立性的方法是，对于每个以非零概率出现的消息 $\mathsf{m} \in \mathcal{M}$ 和每个密文 $\mathsf{c} \in \mathcal{C}$，我们都有：

$$\Pr[\mathsf{c} = c \mid \mathsf{m} = m] = \Pr[\mathsf{c} = c]$$

直观上说，这意味着明文消息的选择对密文的分布没有任何影响。

限制 m 和 k 是相互独立的随机变量是明智的：在使用任何密码时，根据消息选择密钥是一个非常糟糕的主意，反之亦然（参见练习 2.16）。

定理 2.4. 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的香农密码。考虑一个随机实验，其中 k 和 m 是随机变量，满足：

- k 在 $\mathcal{K}$ 上均匀分布，
- m 在 $\mathcal{M}$ 上分布，且
- k 和 m 相互独立。

定义随机变量 $\mathsf{c} := E(\mathsf{k}, \mathsf{m})$。那么我们有：

- 如果 $\mathcal{E}$ 是完美安全的，那么 c 和 m 相互独立。
- 反过来说，如果 c 和 m 相互独立，并且 $\mathcal{M}$ 中的每个消息都以非零概率出现，那么 $\mathcal{E}$ 是完美安全的。

证明. 对于第一个结论，假设 $\mathcal{E}$ 是完美安全的。考虑任意给定的 $\mathsf{m} \in \mathcal{M}$ 和 $\mathsf{c} \in \mathcal{C}$，我们想要证明：

$$\Pr[\mathsf{c} = c \wedge \mathsf{m} = m] = \Pr[\mathsf{c} = c] \Pr[\mathsf{m} = m]$$

我们有：

$$\begin{aligned}
\Pr[\mathsf{c} = c \wedge \mathsf{m} = m] &= \Pr[E(\mathsf{k}, \mathsf{m}) = c \wedge \mathsf{m} = m] \\
&= \Pr[E(\mathsf{k}, m) = c \wedge \mathsf{m} = m] \\
&= \Pr[E(\mathsf{k}, m) = c] \Pr[\mathsf{m} = m] \quad (\mathsf{k} \text{和} \mathsf{m} \text{相互独立})
\end{aligned}$$

因此下面只需要证明 $\Pr[E(\mathsf{k}, m) = c] = \Pr[\mathsf{c} = c]$ 就够了。我们有：

$$\begin{aligned}
\Pr[\mathsf{c} = c] &= \Pr[E(\mathsf{k}, \mathsf{m}) = c] \\
&= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, \mathsf{m}) = c \wedge \mathsf{m} = m'] \quad (\text{全概率}) \\
&= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, m') = c \wedge \mathsf{m} = m'] \\
&= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, m') = c] \cdot \Pr[\mathsf{m} = m'] \quad (\mathsf{k} \text{和} \mathsf{m} \text{相互独立}) \\
&= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, m) = c] \cdot \Pr[\mathsf{m} = m'] \quad (\text{完美安全性的定义}) \\
&= \Pr[E(\mathsf{k}, m) = c] \sum_{m' \in \mathcal{M}} \Pr[\mathsf{m} = m'] \\
&= \Pr[E(\mathsf{k}, m) = c] \quad (\text{全概率})
\end{aligned}$$

对于第二个结论，假设 c 和 m 相互独立，并且 $\mathcal{M}$ 中的每个消息都以非零概率出现。令 $m \in \mathcal{M}$，$c \in \mathcal{C}$。我们将要证明 $\Pr[E(\mathsf{k}, m) = c] = \Pr[\mathsf{c} = c]$，这样自然就能证明 $\mathcal{E}$ 是完美安全的。由于 $\Pr[\mathsf{m} = m] \neq 0$，我们可以看出：

$$\begin{aligned}
Pr[E(\mathsf{k}, m) = c] \Pr[\mathsf{m} = m] &= \Pr[E(\mathsf{k}, m) = c \wedge \mathsf{m} = m] \quad (\mathsf{k} \text{和} \mathsf{m} \text{相互独立}) \\
&= \Pr[E(\mathsf{k}, \mathsf{m}) = c \wedge \mathsf{m} = m] \\
&= \Pr[\mathsf{c} = c \wedge \mathsf{m} = m] \\
&= \Pr[\mathsf{c} = c] \Pr[\mathsf{m} = m] \quad (\mathsf{c} \text{和} \mathsf{m} \text{相互独立}) \quad \square
\end{aligned}$$

2.1.3 坏消息

我们把坏消息留到最后。下一个定理表明，完美安全是一个如此强大的概念，以至于没有任何其他方法能够超越一次性密码本：想要实现完美安全性，密钥长度至少要与消息相等。因此，在实践中几乎不可能使用完美安全的密码：如果 Alice 想给 Bob 发送一个 1 GB 的视频文件，那么 Alice 和 Bob 就必须事先商定一个 1 GB 的密钥。

定理 2.5 (香农定理). 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的香农密码。如果 $\mathcal{E}$ 是完美安全的，则 $|\mathcal{K}| \geq |\mathcal{M}|$。

证明. 假设 $|\mathcal{K}| < |\mathcal{M}|$，我们现在想要证明在这种情况下 $\mathcal{E}$ 不是完美安全的。为此，我们说明存在消息 m_0 和 m_1，以及一条密文 c 使得：

$$\Pr[E(\mathsf{k}, m_0) = c] > 0 \tag{2.1}$$

$$\Pr[E(\mathsf{k}, m_1) = c] = 0 \tag{2.2}$$

成立。这里，k 是一个均匀分布在 $\mathcal{K}$ 上的随机变量。

为了做到这一点，我们选择某个消息 $m_0 \in \mathcal{M}$ 和某个密钥 $k_0 \in \mathcal{K}$。令 $c := E(k_0, m_0)$。很明显，此时式 2.1 成立。

接下来，令：

$$S := \{D(k_1, c) : k_1 \in \mathcal{K}\}$$

明显有：

$$|S| \leq |\mathcal{K}| < |\mathcal{M}|$$

这样，我们就可以选取一个消息 $m_1 \in \mathcal{M} \setminus S$。

为了证明式 2.2 也成立，我们需要说明不存在密钥 k_1 能使 $E(k_1, m_1) = c$。相反地，我们假设存在一个密钥 k_1 使得 $E(k_1, m_1) = c$ 成立，那么对于这个密钥 k_1，根据密码的正确性属性，我们有：

$$D(k_1, c) = D(k_1, E(k_1, m_1)) = m_1$$

这就意味着 m_1 属于 S，而事实并非如此。因此式 2.2 成立，定理得证。 □

2.2 计算性密码与语义安全性

正如我们在香农定理（定理 2.5）中所看到的，实现完美安全的唯一方法是拥有和消息一样长的密钥。然而这是很不现实的，我们往往希望能够用一个短的密钥（比如几百比特）来加密一个长的消息（比如几兆字节）。绕过香农定理的唯一方法是放松我们对安全性的要求。我们要做的是，不考虑所有可能的对手，而只考虑计算上可行的对手，也就是说，“真实世界” 对手必须在真实的计算机上使用合理的时间和内存进行计算。这就将我们导向了一个稍弱的安全定义，称为**语义安全 (semantic security)**。这个安全定义更加灵活，只要一个允许变长消息空间的密码不向对手泄露除消息长度之外的任何有用信息，我们就称该密码是安全的。另外，由于我们现在关注的是 “实用性” 而不是 “数学上的可能性”，我们还会坚持认为，加密和解密函数本身都是有效算法，而不是任意的函数。

2.2.1 计算性密码的定义

一个**计算性密码 (computational cipher)** $\mathcal{E} = (E, D)$ 是一对有效密码算法 E 和 D。加密算法 E 接受一个密钥 k 和一条消息 m 作为输入，生成并输出一条密文 c。揭秘算法 D 接受一个密钥 k 和一条密文 c 作为输入，并输出一条消息 m。密钥 k 位于某个有限的密钥空间 $\mathcal{K}$ 中，消息 m 位于某个有限的消息空间 $\mathcal{M}$ 中，密文 c 位于某个有限的密文空间 $\mathcal{C}$ 中。正如香农密码一样，我们称 $\mathcal{E}$ 定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上。

尽管对于我们在本章中的目的来说，这并不是必要的，但我们仍将允许加密函数 E 是一种概率性算法（参见第二十六章）。这意味着对于固定的输入 k 和 m，$E(k,m)$ 的输出值可能是多个值中的一个。为了强调这种计算的概率性质，我们用：

$$c \xleftarrow{\mathrm{R}} E(k,m)$$

来表示执行 $E(k,m)$ 并将其输出分配给程序变量 c 的过程。在本书中，只要我们使用概率性算法，我们就会使用这一符号。同样地，我们用：

$$k \xleftarrow{\mathrm{R}} \mathcal{K}$$

来表示从密钥空间 $\mathcal{K}$ 中随机均匀地选取一个值赋给程序变量 k 的过程。我们会使用类似的符号表示从任何有限集中均匀随机抽样的过程。

在本章中，我们不会介绍任何具体地概率加密算法（我们将在第五章中看到这方面的第一个例子）。虽然我们也可以令解密算法也是概率性的，但是并没有这个必要，因此我们只会讨论使用确定性解密算法的密码。然而，我们允许解密算法返回一个（与所有有效消息都不同的）特殊的 reject，这在某些场合是很有用的，可以表明解密过程中发生了某种错误。

由于加密算法是概率性的，对于一个给定的密钥 k 和消息 m，加密算法可能会输出许多可能的密文之一；然而，这些可能的密文中的每一个都应该解密为 m。我们可以更正式地表述这个**正确性要求**：对于所有的密钥 $k \in \mathcal{K}$ 和消息 $m \in \mathcal{M}$，如果我们执行：

$$c \xleftarrow{\mathrm{R}} E(k,m), \quad m' \xleftarrow{\mathrm{R}} D(k,c)$$

那么 $m = m'$ 成立的概率为 1。

> 从现在开始，每当我们提到一个**密码**时，我们指的都是如上所定义的**计算性密码**。此外，如果加密算法恰好也是确定性的，那么我们就可以称该密码为一个**确定性密码**。

请注意，任何确定性密码都是香农密码，但是计算性密码不一定是香农密码（如果其有一个概率性加密算法），香农密码也不一定是计算性密码（如果其加密或解密操作没有有效实现）。

例 2.8. 一次性密码本（见例 2.1）和变长一次性密码本（见例 2.2）都是确定性密码，因为它们的加密和解密操作都可以通过有效的确定性算法来实现。替换密码（见例 2.3）只要字母表 Σ 不是太大，也属于确定性密码。事实上，在显而易见的实现方法中，一个密钥也就是对 Σ 的一种置换方案，会用一个以 Σ 为索引的数组来表示，所以我们需要 $O(|\Sigma|)$ 的空间来存储一个密钥。因此，这只适合大小合理的 Σ 的情形。例 2.4 中讨论的加性一次性密码本也是一个确定性密码，因为加密和解密操作都可以有效地实现（如果 n 很大，可能需要特殊的软件来进行大整数的算术运算）。

2.2.2 语义安全性的定义

为了引入语义安全性的定义，考虑一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的确定性密码 $\mathcal{E} = (E,D)$。我们重新考察定理 2.3 给出的关于完美安全性的表述。它表明，对于密文空间上所有的谓词逻辑 ϕ 和所有消息 m_0，m_1，我们都有：

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \Pr[\phi(E(\mathsf{k}, m_1))] \tag{2.3}$$

其中 k 是一个在 $\mathcal{K}$ 上均匀分布的随机变量。现在，我们不再要求概率相等，而只是要求它们非常接近，即：

$$|\Pr[\phi(E(\mathsf{k}, m_0))] - \Pr[\phi(E(\mathsf{k}, m_1))]| \leq \epsilon \tag{2.4}$$

对一个非常小的，或者*可以忽略不计的* ϵ 成立。就其本身而言，这种放松并没有多大帮助（见练习 2.5）。然而，我们现在不再要求式 2.4 对所有可能的谓词 ϕ 和消息 m_0，m_1 都成立，而是只要求式 2.4 对所有可以由某种有效算法生成的消息 m_0，m_1，以及所有可以由某种有效算法计算的谓词 ϕ 成立（这些有效算法可以是概率性的）。例如，假设使用最好的算法来生成 m_0 和 m_1，以及测试一些谓词 ϕ，并使用（例如）一万台计算机在并行计算十年的情况下式 2.4 在 $\epsilon = 2^{-100}$ 的情况下仍然成立。虽然这种密码仍然不是完美安全的，但是我们可以认为这种密码对于*所有的实际目的都是足够安全的*。

此外，在定义语义安全时，我们还解决了例 2.5 中的一个问题。在那个例子中，我们看到变长一次性密码本并不满足完美安全性的定义。然而，我们希望我们的定义有足够的灵活性，以便像变长一次性密码本这样的密码，它除了长度之外不会泄露任何关于消息的信息，也可以被认为是安全的。

现在说说细节。为了精确定义语义安全性，我们将描述一个在两方之间进行的**攻击游戏 (attack game)**，这两方分别为**挑战者 (challenger)** 和**对手 (adversary)**。正如我们将要看到的，挑战者会遵循一个非常简单且固定的协议。然而对手 $\mathcal{A}$ 可以遵循一个任意的（但仍然有效的）协议。挑战者和对手 $\mathcal{A}$ 按照他们各自的协议来回发送消息，在游戏结束时，$\mathcal{A}$ 将输出一些数值。事实上，我们用于定义语义安全的攻击游戏包括两个可供选择的 “子游戏” 或 “子实验”，在这两个实验中，对手会遵循相同的协议，但挑战者的行为在这两个实验中略有不同。攻击游戏还定义了一个概率空间，而这反过来又定义了对手的*优势*，它衡量的是在这个概率空间中两个事件的概率之差。

攻击游戏 2.1 (语义安全性). 给定一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的确定性密码 $\mathcal{E} = (E, D)$，对于一个给定对手 $\mathcal{A}$，我们定义两个实验：实验 0 和实验 1。对于 $b = 0, 1$，我们定义：
实验 b:

- 对手计算长度相同的消息两个 $m_0, m_1 \in \mathcal{M}$，并将其发送给挑战者。
- 挑战者计算 $k \xleftarrow{\mathrm{R}} \mathcal{K}$，$c \xleftarrow{\mathrm{R}} E(k, m_b)$，并将 c 发送给对手。
- 对手输出一个比特 $\hat{b} \in \{0, 1\}$。

对于 $b = 0, 1$，令 W_b 为 $\mathcal{A}$ 在实验 b 中输出 1 的事件。那么我们将对手 $\mathcal{A}$ 相对于 $\mathcal{E}$ 的**语义安全优势**定义为：

$$\mathrm{SSadv}[\mathcal{A}, \mathcal{E}] = |\Pr[W_0] - \Pr[W_1]|$$

请注意，在上述游戏中，事件 W_0 和 W_1 的定义依赖于 k 的随机选择、加密算法的随机选择（如果有的话）和对手的随机选择（如果有的话）所共同决定的概率空间。$\mathrm{SSadv}[\mathcal{A}, \mathcal{E}]$ 的值是一个介于 0 和 1 之间的数字。

攻击游戏 2.1 的示意图见图 2.1。如图所示，$\mathcal{A}$ 的 “输出” 实际上只是给挑战者的一个最终消息。

定义 2.2 (语义安全性). *如果对于所有有效对手 $\mathcal{A}$ 来说，$\mathrm{SSadv}[\mathcal{A}, \mathcal{E}]$ 的值都可以忽略不计，则称密码 $\mathcal{E}$ 是语义安全的。*

作为一个正式的定义，这还不是很完整，因为我们还没有定义 “相同长度的消息”、“有效对手” 和 “可忽略不计” 是什么意思。我们很快就会回到这个问题上。

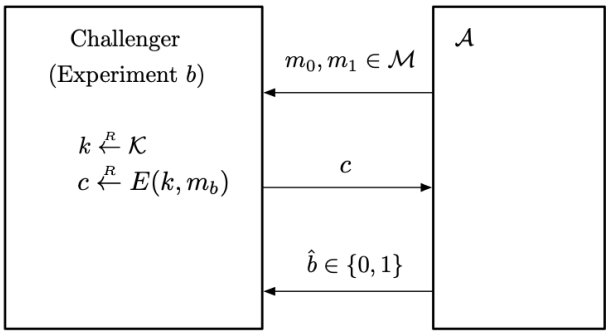


图 2.1: 攻击游戏 2.1 中的实验 b

让我们把这个正式定义与前面的讨论联系起来。假设攻击游戏 2.1 中的对手 $\mathcal{A}$ 是确定性的。首先，对手以一种确定性的方式计算出消息 m_0 和 m_1，然后考察密文 c 上的谓词 ϕ，如果结果为真就输出 1，为假则输出 0。语义安全的意思是，式 2.4 中的 ϵ 是可以忽略不计的。如果 $\mathcal{A}$ 是概率性的，我们可以将 $\mathcal{A}$ 看作是这样的结构：他从某个适当的集合中生成一个随机数 r，并确定性地计算取决于 r 的消息 $m_0^{(r)}$ 和 $m_1^{(r)}$，接着评估谓词 $\phi^{(r)}$，它也取决于 r。这里，语义安全说明，当用 $m_0^{(r)}$、$m_1^{(r)}$ 和 $\phi^{(r)}$ 替换式 2.4 中的 m_0、m_1 和 ϕ 时，ϵ 的大小仍然是可以忽略不计的，但现在的概率是针对一个随机选择的密钥和随机选择的 r 值而言的。

备注 2.1. 现在让我们说一下，为什么我们要求在攻击游戏 2.1 中，对手 $\mathcal{A}$ 计算出来的消息 m_0 和 m_1 必须是相同长度的。

- 首先，消息的 “长度” 概念是针对特定的消息空间 $\mathcal{M}$ 而言的。换句话说，在指定一个消息空间时，我们必须指定一个规则，它能将长度（一个非负整数）与任何给定的消息关联起来。对于大多数具体的消息空间来说，这种联系是很显然的。例如，对于消息空间 $\{0,1\}^{\leq L}$（如例 2.2）来说，消息 $m \in \{0,1\}^{\leq L}$ 的长度就是它的比特位数 $|m|$。然而，为了使我们的定义具有一定的普遍性，我们把长度的概念作为一个抽象概念。事实上，有些消息空间可能没有特别的长度概念，在这种情况下，所有的消息都可以被看作是长度为 0 的。

- 其次，要求 m_0 和 m_1 具有相同的长度，意味着对手不会因为能够有效地根据长度区分两条密文就被认为是破坏了系统。这就是我们的正式定义所捕捉到的概念，即消息的加密被允许泄漏消息的长度（但不会泄漏其他信息）。

  我们在例 2.5 中已经讨论过，在某些应用中，泄露消息的长度可能会带来灾难性的后果。然而，由于这个问题没有通用的解决方案，大多数现实世界的加密方案（例如 TLS）根本没有尝试去隐藏消息的长度。这可能会导致真正的攻击。例如，Chen 等人表明，加密消息的长度可以揭示用户提供给云应用程序的私人数据的大量信息。他们以一个在线报税系统为例，但也有其他工作表明这种类型的攻击同样适用于很多其他的系统。

例 2.9. 令 $\mathcal{E}$ 是一个确定性密码，且是完美安全的。那么很容易看出，对于任意对手 $\mathcal{A}$（无论是否有效），我们都有 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 0$。这几乎可以立即从定理 2.3 中得出（唯一稍微复杂的是，我们在攻击游戏 2.1 中的对手 $\mathcal{A}$ 可能是概率性的，但这很容易处理）。特别是，$\mathcal{E}$ 是语义安全的。因此，如果 $\mathcal{E}$ 是一次性密码本（见例 2.1），那么对于所有对手 $\mathcal{A}$，我们有 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 0$，因此一次性密码本是语义安全

的。由于语义安全的定义对于变长的消息空间来说是比较宽容的，所以也很容易看出，如果 $\mathcal{E}$ 是变长一次性密码本（见例 2.2），那么对于所有对手 $\mathcal{A}$，$\mathsf{SSadv}[\mathcal{A},\mathcal{E}] = 0$ 都成立，因此变长一次性密码本也是语义安全的。

关于 “有效” 和 “可忽略不计” 这两个词，我们还要多说几句。在下面的 2.3 节中，我们将填补其余的细节（这些细节可能会稍显乏味，而且实际上并不是很有启发性）。直观地说，*可忽略不计*的意思是小到 “对所有实际用途来说都是零”：想像一下 2^{-100} 这样的数字，如果你在下一年自燃的概率是 2^{-100}，那么你就不会担心这种事件的发生。我们还使用下列术语：

- 一个*有效对手*是一个能在 “合理” 时间内运行的对手。
- 如果 $1/N$ 可以忽略不计，就称 N 是*超多项式 (super-poly)* 的。
- 一个*多项式边界 (poly-bounded)* 的值是一个 “合理” 大小的数字。特别地，我们可以说，一个有效对手的运行时间是多项式边界的。

事实 2.6. *如果 ϵ 和 ϵ' 是两个可以忽略不计的值，而 Q 和 Q' 是两个多项式边界的值，那么：*

(i) *$\epsilon + \epsilon'$ 是一个可以忽略不计的值，*

(ii) *$Q + Q'$ 和 $Q \cdot Q'$ 都是多项式边界的，且*

(iii) *$Q \cdot \epsilon$ 是一个可以忽略不计的值。*

现在，读者可以把这些事实作为公理。我们不纠缠于这些技术问题，而是讨论一个例子，说明在分析一个使用语义安全密码的更大系统的安全性时，我们通常如何使用这个定义。

2.2.3 与较弱的安全概念的联系

2.2.3.1 消息恢复攻击

直观地说，在消息恢复攻击 (message recovery attack) 中，对手被给定一个随机消息的加密，并且能够以明显优于随机猜测的概率（即概率为 $1/|\mathcal{M}|$）从密文中恢复消息。当然，任何合理的安全概念都应该排除这种攻击，语义安全当然也是如此。

虽然这在直觉上似乎是显而易见的，但我们给出了这一点的正式证明。我们这样做的动机之一是为了详细说明*安全归约 (security reduction)* 的概念，这是用于推理系统安全性的主要技术。基本上，该证明将论证，任何能够有效地对 $\mathcal{E}$ 发起消息恢复攻击的有效对手 $\mathcal{A}$，都可以被用于建立一个能够破坏 $\mathcal{E}$ 的语义安全性的有效对手 $\mathcal{B}$；由于语义安全性意味着不存在这样的 $\mathcal{B}$，我们就可以得出结论，这样的 $\mathcal{A}$ 也是不存在的。

为了更详细地给出这个证明，我们需要一个消息恢复攻击的正式定义。和以前一样，这也是通过一个攻击游戏来完成的，该攻击游戏是一个挑战者与一个对手之间的游戏。

攻击游戏 2.2 (消息恢复). 给定一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的密码 $\mathcal{E} = (E, D)$，对于一个给定对手 $\mathcal{A}$，攻击游戏的过程如下：

- 挑战者计算 $m \xleftarrow{R} \mathcal{M}$，$k \xleftarrow{R} \mathcal{K}$，$c \xleftarrow{R} E(k, m)$，并将 c 发送给对手。
- 对手输出一条消息 $\hat{m} \in \mathcal{M}$。

令 W 为 $\hat{m} = m$ 成立的事件。我们称，在这种情况下，$\mathcal{A}$ 赢得该游戏。我们将对手 $\mathcal{A}$ 相对于 $\mathcal{E}$ 的**消息恢复优势**定义为：

$$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] := |\Pr[W] - 1/|\mathcal{M}||$$

定义 2.3 (针对消息恢复的安全性). 如果对于所有有效对手 $\mathcal{A}$ 来说，$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}]$ 的值都可以忽略不计，我们就说密码 $\mathcal{E}$ 对消息恢复是安全的。

定理 2.7. 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的密码。如果 $\mathcal{E}$ 是语义安全的，那么 $\mathcal{E}$ 对消息恢复也是安全的。

证明. 假设 $\mathcal{E}$ 是语义安全的。我们的目标是证明 $\mathcal{E}$ 对于消息恢复也是安全的。

为了证明 $\mathcal{E}$ 对于消息恢复是安全的，我们必须证明每一个有效对手 $\mathcal{A}$ 在攻击游戏 2.2 中的优势都可忽略不计。为了证明这一点，我们给定一个任意但高效的对手 $\mathcal{A}$，现在我们的目标是证明 $\mathcal{A}$ 的消息恢复优势 $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}]$ 是可以忽略不计的。令 p 为 $\mathcal{A}$ 赢得消息恢复游戏的概率，那么有：

$$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] = |p - 1/|\mathcal{M}||$$

我们下面展示如何构建一个有效对手 $\mathcal{B}$，其在攻击游戏 2.1 中的语义安全优势与 $\mathcal{A}$ 在攻击游戏 2.2 中的消息恢复优势有如下关系：

$$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] \leq \mathrm{SSadv}[\mathcal{B}, \mathcal{E}] \tag{2.5}$$

由于 $\mathcal{B}$ 是有效的，且我们假设 $\mathcal{E}$ 是语义安全的，因此式 2.5 中不等号右侧的值可以忽略不计，由此我们可以得出结论：$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}]$ 也可以忽略不计。

因此，完成证明所剩下的工作就是说明如何构造一个满足式 2.5 要求的有效的 $\mathcal{B}$。我们的想法是把 $\mathcal{A}$ 作为一个 “黑箱”，因为我们根本不需要了解 $\mathcal{A}$ 的内部运作情况。

以下是 $\mathcal{B}$ 的工作方式。对手 $\mathcal{B}$ 产生两条随机消息 $m_0, m_1 \in \mathcal{M}$，并将其发送给自己的语义安全挑战者。这个挑战者向 $\mathcal{B}$ 发送一条密文 c，$\mathcal{B}$ 将其转发给 $\mathcal{A}$，就像它来自 $\mathcal{A}$ 的消息恢复挑战者一样。当 $\mathcal{A}$ 输出一个信息 $\hat{m}$ 时，如果 $\hat{m} = m_1$，我们的对手 $\mathcal{B}$ 就输出 $\hat{b} = 1$，否则输出 $\hat{b} = 0$。

这就完成了对 $\mathcal{B}$ 的描述。注意，$\mathcal{B}$ 的运行时间与 $\mathcal{A}$ 的运行时间基本相同。我们现在分析 $\mathcal{B}$ 的语义安全优势，并将其与 $\mathcal{A}$ 的消息恢复优势关联起来。

对于 $b = 0, 1$，令 p_b 表示当 $\mathcal{B}$ 的语义安全挑战者加密了 m_b 时，$\mathcal{B}$ 的输出为 1 的概率。根据定义，我们有：

$$\mathrm{SSadv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0|$$

一方面，当 c 是对 m_1 的加密时，概率 p_1 正好等于 $\mathcal{A}$ 在消息恢复游戏中获胜的概率，所以 $p_1 = p$。另一方面，当 c 是对 m_0 的加密时，对手 $\mathcal{A}$ 的输出与 m_1 无关，所以有 $p_0 = 1/|\mathcal{M}|$。由此可见：

$$\mathrm{SSadv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0| = |p - 1/|\mathcal{M}|| = \mathrm{MRadv}[\mathcal{A}, \mathcal{E}]$$

这就证明了式 2.5。事实上，式 2.5 中的等号是成立的，但这对本证明并不重要。 □

读者应该确保自己理解这个证明的逻辑，因为这种类型的证明将在本书中反复使用。我们再回顾一下该证明的重要部分，并给出另一种思考方式。

证明的核心建立在以下事实上：对于每一个在攻击游戏 2.2 中攻击 $\mathcal{E}$ 的有效消息恢复对手 $\mathcal{A}$，都存在一个在攻击游戏 2.1 中攻击 $\mathcal{E}$ 的有效语义安全对手 $\mathcal{B}$，使得：

$$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] \leq \mathrm{SSadv}[\mathcal{B}, \mathcal{E}] \tag{2.6}$$

我们试图证明，如果 $\mathcal{E}$ 是语义安全的，那么 $\mathcal{E}$ 对消息恢复也是安全的。在上面的证明中，我们认为如果 $\mathcal{E}$ 是语义安全的，那么式 2.6 的右手边一定是可以忽略的，因此式 2.6 的右手边也一定是可以忽略的；由于这对所有有效对手 $\mathcal{A}$ 都是成立的，我们就能得出结论：$\mathcal{E}$ 对消息恢复是安全的。

证明该定理的另一种方法是反证法：如果 $\mathcal{E}$ 对消息恢复不是安全的，那么 $\mathcal{E}$ 也不是语义安全的。因此，我们假设 $\mathcal{E}$ 对消息恢复不安全。这意味着存在一个有效对手 $\mathcal{A}$，其消息恢复优势是不可忽略不计的。利用 $\mathcal{A}$，我们建立一个满足式 2.6 的有效对手 $\mathcal{B}$。根据假设，$\mathsf{MRadv}[\mathcal{A},\mathcal{E}]$ 是不可忽略不计的，而式 2.6 意味着 $\mathsf{SSadv}[\mathcal{B},\mathcal{E}]$ 也只能是不可忽略的。由此，我们得出结论，$\mathcal{E}$ 不是语义安全的。

说得更简单些：为了证明语义安全意味着针对消息恢复的安全，我们展示了如何将一个可以打破消息恢复的有效对手转化成一个可以打破语义安全性的有效对手。

我们还强调，证明中构建的对手 $\mathcal{B}$ 只是将 $\mathcal{A}$ 作为一个“黑箱”。事实上，我们之后将要看到的几乎所有构造都是这种类型的。$\mathcal{B}$ 本质上只是 $\mathcal{A}$ 的一个包装，它由 $\mathcal{B}$ 的挑战者和 $\mathcal{A}$ 的单个运行实例之间的一些简单有效的“接口层”组成。理想情况下，我们希望接口层的计算复杂性不依赖于 $\mathcal{A}$ 的计算复杂性；然而，一些依赖性是不可避免的：如果一个攻击游戏允许 $\mathcal{A}$ 向其挑战者进行多次查询，$\mathcal{A}$ 的查询次数越多，接口层必须执行的工作就越多，但这个工作应该只依赖于查询的数量，而不是 $\mathcal{A}$ 的运行时间。

因此，当对手 $\mathcal{B}$ 可以像上述那样被结构化为与 $\mathcal{A}$ 互动的有效接口时，我们就称它是围绕对手 $\mathcal{A}$ 的一个**基本包装器 (elementary wrapper)**。其突出的特性是：

- 如果 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，并且 $\mathcal{A}$ 是有效的，那么 $\mathcal{B}$ 也是有效的。
- 如果 $\mathcal{C}$ 是围绕 $\mathcal{B}$ 的一个基本包装器，而 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，那么 $\mathcal{C}$ 也是一个 $\mathcal{A}$ 的基本包装器。

2.2.3.2 计算消息的单个比特

如果一个加密方案是安全的，我们不仅应该很难恢复整个消息，也应该很难计算出关于消息的任何部分的信息。

我们不会在这里证明一个完全通用的定理，而是考虑一个具体的例子。

假设 $\mathcal{E}=(E,D)$ 是一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的密码，其中 $\mathcal{M}=\{0,1\}^L$。对于 $m\in\mathcal{M}$，如果 m 中比特 1 的数量是奇数，我们就定义 $\mathrm{parity}(m)$ 为的值 1，否则为 0。与这个结论等价的另一个结论是，$\mathrm{parity}(m)$ 的值就等于对构成 m 的所有比特做异或操作的结果。

我们将证明，如果 $\mathcal{E}$ 是语义安全的，那么给定一个随机消息 m 的加密 c，很难预测 $\mathrm{parity}(m)$。由于 $\mathrm{parity}(m)$ 是一个 1 比特的信息，任何对手都可以通过随机猜测以 1/2 的概率猜中这个值。尽管如此，我们想要说明的是，没有任何有效对手能够稍微提升哪怕一点猜中的概率。

作为热身，假设有一个有效对手 $\mathcal{A}$ 能够以 1 的概率猜中 $\mathrm{parity}(m)$。这意味着，对于每个消息 m，每个密钥 k，以及 m 的每个加密 c，当我们向 $\mathcal{A}$ 提供密文 c 时，它就能够输出 m 的奇偶性。我们的对手任意选择两条消息 m_0 和 m_1，但 $\mathrm{parity}(m_0)=0$，$\mathrm{parity}(m_1)=1$。然后，它把这两条消息交给自己的语义安全挑战者，得到一个密文 c，然后将其转发给 $\mathcal{A}$。在收到 c 后，对手 $\mathcal{A}$ 输出一个比特 $\hat{b}$，而 $\mathcal{B}$ 输出这个相同的比特 $\hat{b}$ 作为它自己的输出。很容易看出，$\mathcal{B}$ 的语义安全优势恰好是 1：当它的语义安全挑战者加密的是 m_0 时，它总是输出 0，而当它的语义安全挑战者加密的是 m_1 时，它总是输出 1。

这表明，如果 $\mathcal{E}$ 是语义安全的，就不存在能以 1 的概率预测奇偶性的有效对手。然而，我们可以再进一步：如果 $\mathcal{E}$ 是语义安全的，就不存在能以明显优于 1/2 的概率预测奇偶性的有效对手。为了准确说明这一点，我们给出一个攻击游戏。

攻击游戏 2.3 (奇偶性预测). 给定一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的密码 $\mathcal{E}=(E,D)$，对于一个给定对手 $\mathcal{A}$，攻击游戏的过程如下：

- 挑战者计算 $m \xleftarrow{R} \mathcal{M}$，$k \xleftarrow{R} \mathcal{K}$，$c \xleftarrow{R} E(k,m)$，并将 c 发送给对手。
- 对手输出 $\hat{b} \in \{0,1\}$。

令 W 为 $\hat{b} = \text{parity}(m)$ 的事件。我们定义 $\mathcal{A}$ 对于 $\mathcal{E}$ 的**奇偶性预测优势**为：

$$\text{Parityadv}[\mathcal{A},\mathcal{E}] := |\Pr[W] - 1/2|$$

定义 2.4 (奇偶性预测). 如果对于所有有效对手 $\mathcal{A}$，$\text{Parityadv}[\mathcal{A},\mathcal{E}]$ 的值都可以忽略不计，我们就称密码 $\mathcal{E}$ 对奇偶性预测是安全的。

定理 2.8. 令 $\mathcal{E}=(E,D)$ 是一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的密码，其中 $\mathcal{M}=\{0,1\}^L$。如果 $\mathcal{E}$ 是语义安全的，那么 $\mathcal{E}$ 对奇偶性预测是安全的。

证明. 与定理 2.7 的证明一样，我们也给出一个基于规约的证明。特别地，我们将证明，对于每个按照攻击游戏 2.3 攻击 $\mathcal{E}$ 的奇偶性预测对手 $\mathcal{A}$，都存在一个按照攻击游戏 2.1 攻击 $\mathcal{E}$ 的语义安全对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足：

$$\text{Parityadv}[\mathcal{A},\mathcal{E}] = \frac{1}{2}\cdot \text{SSadv}[\mathcal{B},\mathcal{E}]$$

令 $\mathcal{A}$ 是一个奇偶性预测对手，他能以 $1/2+\epsilon$ 的概率预测奇偶性，那么有 $\text{Parityadv}[\mathcal{A},\mathcal{E}]=\epsilon$。

下面我们展示，如何构建我们的语义安全对手 $\mathcal{B}$。

我们的对手 $\mathcal{B}$ 生成一个随机消息 m_0，并设置 $m_1 \leftarrow m_0 \oplus (0^{L-1}||1)$。也就是说，$m_1$ 除了最后一位被翻转之外，其他各位都与 m_0 相同。因此 m_0 与 m_1 的奇偶性正好相反。

我们的对手 $\mathcal{B}$ 将这对 m_0，m_1 发送给它自己的语义安全挑战者，并从挑战者那里收到一条密文 c，然后将 c 转发给 $\mathcal{A}$。当 $\mathcal{A}$ 输出一个比特 $\hat{b}$ 时，如果 $\hat{b}=\text{parity}(m_0)$，我们的对手 $\mathcal{B}$ 就输出 1，否则就输出 0。

对于 $b=0,1$，令 p_b 是当对手 $\mathcal{B}$ 的语义安全挑战者加密了 m_b 的情况下对手 $\mathcal{B}$ 输出为 1 的概率。所以根据定义，我们有：

$$\text{SSadv}[\mathcal{B},\mathcal{E}] = |p_1 - p_0|$$

我们声称 $p_0 = 1/2+\epsilon$，$p_1 = 1/2-\epsilon$。这是因为，无论是加密的是 m_0 还是 m_1，m_b 在 $\mathcal{M}$ 上的分布都是均匀的，因此在 $b=0$ 的情况下，我们的奇偶性预测者 $\mathcal{A}$ 将以 $1/2+\epsilon$ 的概率输出 $\text{parity}(m_0)$；而当 $b=1$ 时，我们的奇偶性预测者 $\mathcal{A}$ 将以 $1/2+\epsilon$ 的概率输出 $\text{parity}(m_1)$，也就是以 $1-(1/2+\epsilon)=1/2-\epsilon$ 的概率输出 $\text{parity}(m_0)$。

因此：

$$\text{SSadv}[\mathcal{B},\mathcal{E}] = |p_1 - p_0| = 2|\epsilon| = 2\cdot \text{Parityadv}[\mathcal{A},\mathcal{E}]$$

这就证明了该定理。 □

我们已经表明，如果一个对手能够有效地预测一条消息的奇偶性，那么它就可以被用来破坏语义安全。反过来说，事实证明，如果一个对手能够破坏语义安全，他就能有效地预测消息的某些谓词（见练习 3.15）。

2.2.4 语义安全的结果

在本节中，我们将在一个具体的例子，即电子赌博的背景下研究语义安全的后果。这个例子的具体细节并不那么重要，但是这个例子说明了人们通常是如何在应用中使用语义安全假设的。

考虑下面这个极其简化的轮盘赌版本，它是一种在*庄荷 (house)* 与一个*玩家 (player)* 之间进行的游戏。玩家给庄荷 1 美元，然后他可以下两种赌注中的一种：

- “高或低”，或者
- “偶或奇”。

下注后，庄荷随机选择一个数字 $r \in \{0, 1, ..., 36\}$。当 $r \neq 0$，并且：

- 玩家下注 “高”，且 $r > 18$，
- 玩家下注 “低”，且 $r \leq 18$，
- 玩家下注 “偶”，且 r 为偶数，
- 玩家下注 “奇”，且 r 为奇数。

时，玩家获胜。如果玩家赢了，庄荷将付给他 2 美元（即玩家净赚 1 美元）；如果玩家输了，庄荷将不会付给他钱（即玩家净输 1 美元）。显然，在这个游戏中，庄荷有一个虽然小但仍然显著的优势：玩家获胜的概率是 $18/37 \approx 48.65\%$。

现在，假设这个游戏是在互联网上进行的。此外，假设由于各种技术原因，在玩家下注*之前*，赌场需要公布 r 的加密（也许是由一些与赌场共享密钥的监管机构来解密）。玩家可以在下注前自由地分析这个加密消息，以此试图来增加他赢钱的机会。然而，如果这个密码是好的，玩家的机会应该不能增加很多。让我们来证明这一点，假设 r 是用定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的一个密码 $\mathcal{E} = (E, D)$ 加密的，其中 $\mathcal{M} = \{0, 1, ..., 36\}$（在这个例子中，我们将 $\mathcal{M}$ 中的所有消息视为具有相同的长度）。另外，从现在开始，让我们称玩家为 $\mathcal{A}$，以强调玩家的对抗性，并假设 $\mathcal{A}$ 的策略可以被建模为一种有效算法。图 2.2 展示了这个游戏。这里，*赌注 (bet)* 表示 “高”、“低”、“偶”、“奇” 中的一个。玩家 $\mathcal{A}$ 将赌注发送给庄荷，庄荷会评估函数 $W(r, bet)$。如果赌注相对于 r 而言是获胜的，函数的值就为 1，否则就为 0。我们定义：

$$\mathrm{IRadv}[A] := |\Pr[W(r, bet) = 1] - 18/37|$$

我们的目标是要证明以下定理。

定理 2.9. *如果 $\mathcal{E}$ 是语义安全的，那么对于每个有效玩家 $\mathcal{A}$，$\mathrm{IRadv}[A]$ 的值都可忽略不计。*

证明. 如我们在 2.2.3 节所做的那样，我们仍然通过安全规约来证明这一点。更具体地说，我们将证明，对于每个玩家 $\mathcal{A}$，都存在一个语义安全对手 $\mathcal{B}$，使得 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，并且：

$$\mathrm{IRadv}[A] = \mathrm{SSadv}[\mathcal{B}, \mathcal{E}] \tag{2.7}$$

因此，如果存在一个优势不可忽略不计的有效玩家 $\mathcal{A}$，我们就能够得到一个有效的语义安全对手 $\mathcal{B}$，它可以打破 $\mathcal{E}$ 的语义安全性，而我们已经假设这是不可能的。因此，不存在这样的 $\mathcal{A}$。

为了分析我们的新对手 $\mathcal{B}$，我们先考虑一个 “理想化” 的互联网轮盘赌版本。在这个版本中，庄荷不公布实际的 r 的加密，而是公布一个 “假” 值的加密，例如 0。图 2.3 展示了这个理想化的网络轮盘

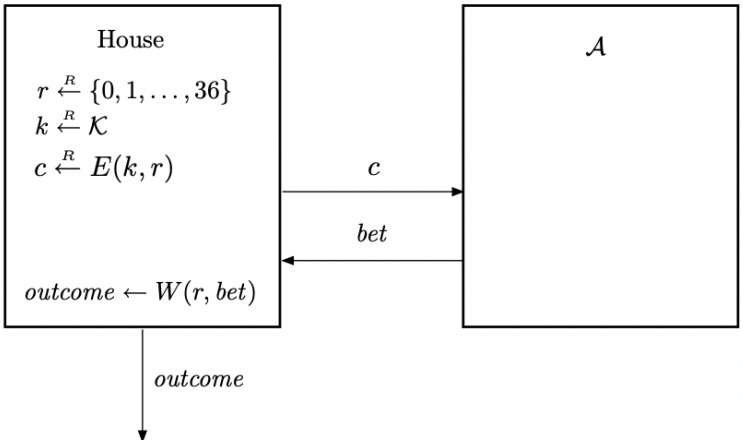


图 2.2: 网络轮盘赌

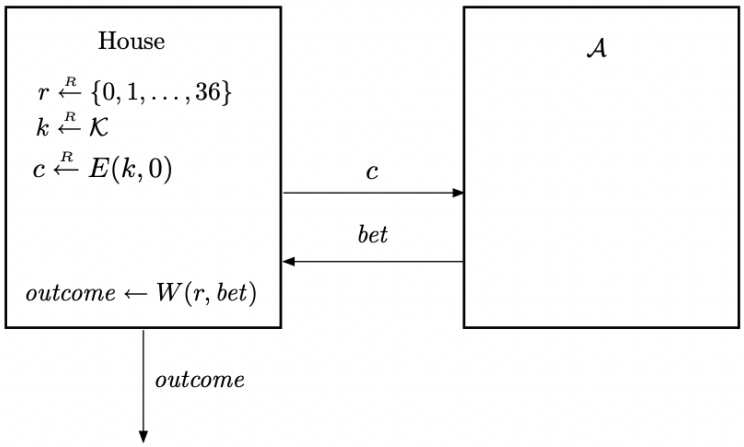


图 2.3: 网络轮盘赌的理想化版本

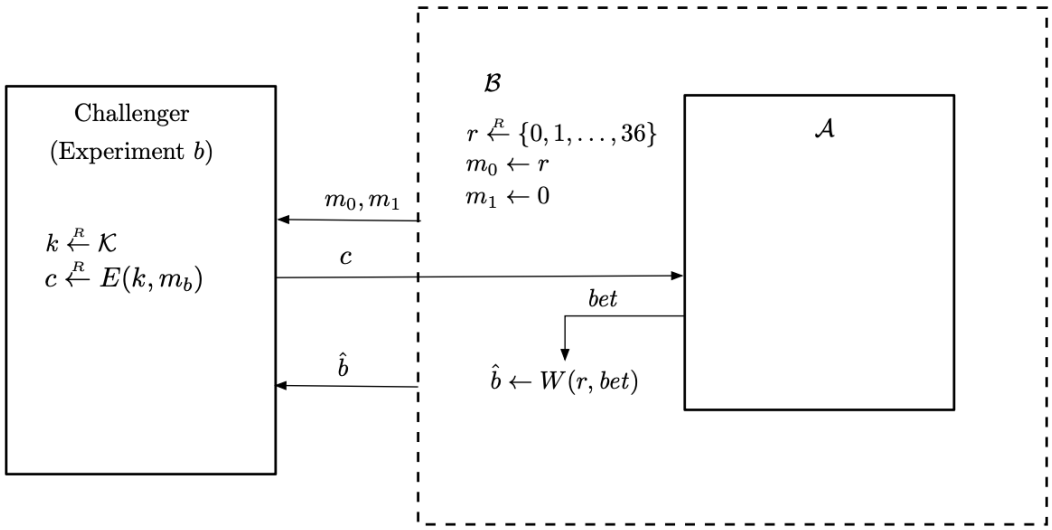


图 2.4: 攻击游戏 2.1 中的语义安全对手 $\mathcal{B}$

赌的逻辑。然而，请注意，在理想化版本的互联网轮盘赌中，庄荷仍然使用 r 的实际值来决定游戏的结果。令 p_0 为 $\mathcal{A}$ 在互联网轮盘赌中获胜的概率，p_1 为 $\mathcal{A}$ 在理想化互联网轮盘赌中获胜的概率。

我们的对手 $\mathcal{B}$ 被设计为在攻击游戏 2.1 中进行游戏。因此，如果令 $\hat{b}$ 表示 $\mathcal{B}$ 在该游戏中的输出，我们有：

- 如果 $\mathcal{B}$ 被置于实验 0 中，那么有 $\Pr[\hat{b}=1]=p_0$；
- 如果 $\mathcal{B}$ 被置于实验 1 中，那么有 $\Pr[\hat{b}=1]=p_1$。

图 2.4 展示了对手 $\mathcal{B}$ 的运行逻辑。通过构造可以看出，$\mathcal{B}$ 满足上面所声称的属性，特别是：

$$\mathrm{SSadv}[\mathcal{B},\mathcal{E}]=|p_1-p_0| \tag{2.8}$$

现在，考虑 $\mathcal{A}$ 在理想化互联网轮盘赌中获胜的概率 p_1。不管 $\mathcal{A}$ 的策略有多高明，他获胜的概率始终是 18/37，因为在这个理想化互联网轮盘赌中，赌注的值是由 c 计算出来的，而 c 在统计上与 r 的值毫无关系。因此：

$$\mathrm{IRadv}[A]=|p_1-p_0| \tag{2.9}$$

结合式 2.8 和式 2.9，我们就能得到式 2.7。 □

我们之后会反复使用这里用来分析互联网轮盘赌的方法。其基本思想是用一个理想化的系统组件替换原来的组件，然后分析这个新的、理想化版本的系统行为。

从上述例子中得到的另一个教训是，在推理一个系统的安全性时，我们将什么视为“对手”，取决于我们想要做什么。在上面的分析中，我们用几个组件拼凑了一个新的对手 $\mathcal{B}$：其中一个组件是原来的对手 $\mathcal{A}$，而其他组件则是从系统的其他部分（在这个例子中是“庄荷”的算法）搜刮来的。这在我们全书的安全分析中会非常典型。直观地说，如果我们想象一个系统图，在安全分析的不同点上，我们将在系统的不同组件周围画一个圆圈，以确定我们在分析那个点时的“对手”。

2.2.5 比特猜测：语义安全性的另一种表征

2.2.4 小节中的例子很典型，它说明人们可以使用语义安全的定义来分析使用语义安全密码的较大系统的安全属性。然而，语义安全还有另一种更便于使用的表征，当我们试图证明一个给定的密码满足该定义时，它往往更方便使用。在这个替代性的表述中，我们定义一个新的攻击游戏。对手所扮演的角色与之前完全相同。然而，我们现在不再进行两个不同的实验，而是只有一个单一的实验。在这个**比特猜测版本 (bit-guessing version)** 的攻击游戏中，挑战者随机选择一个比特 $b\in\{0,1\}$ 并运行攻击游戏 2.1 中的实验 b；对手的目标是以明显优于 1/2 的概率猜中比特 b。下面是该攻击游戏的详细情况。

攻击游戏 2.4 (语义安全性：比特猜测版本). 给定一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的密码 $\mathcal{E}=(E,D)$，对于一个给定对手 $\mathcal{A}$，攻击游戏按如下方式运行：

- 对手 $\mathcal{A}$ 计算相同长度的 $m_0,m_1\in\mathcal{M}$，并将它们发送给挑战者。
- 挑战者计算 $b\xleftarrow{R}\{0,1\}$，$k\xleftarrow{R}\mathcal{K}$，$c\xleftarrow{R}E(k,m_b)$，并将 c 发送给对手。
- 对手输出一个比特 $\hat{b}\in\{0,1\}$。

如果 $\hat{b}=b$，我们就称 $\mathcal{A}$ **赢得**这个游戏。

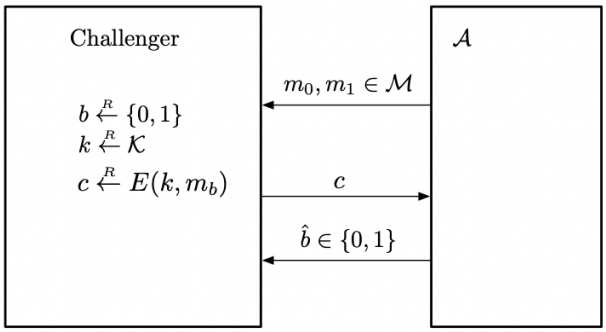


图 2.5: 攻击游戏 2.4

图 2.5 展示了攻击游戏 2.4。请注意，在这个游戏中，$\mathcal{A}$ 赢得游戏的事件是由 b 与 k 的随机选择、加密算法的随机选择（如果有的话）和对手的随机选择（如果有的话）共同构成的概率空间所定义的。

当然，任何对手都能以 1/2 的概率赢得游戏，只需要完全忽略 c 并随机选择 $\hat{b}$（或者说总是选择 $\hat{b} = 0$ 或总是选择 $\hat{b} = 1$）。我们感兴趣的是，对手能比随机猜测好多少。如果用 W 表示对手赢得上述攻击游戏的比特猜测版本的事件，那么我们感兴趣的是 $|\Pr[W] - 1/2|$ 的值，我们用 $\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]$ 表示该值。于是，我们有：

定理 2.10. *对于每个密码 $\mathcal{E}$ 和每个对手 $\mathcal{A}$，我们都有：*

$$\mathrm{SSadv}[\mathcal{B}, \mathcal{E}] = 2 \cdot \mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}] \tag{2.10}$$

证明. 这只是一个简单的计算。令 p_0 是对手在攻击游戏 2.1 的实验 0 中输出 1 的概率，p_1 是对手在攻击游戏 2.1 的实验 1 中输出 1 的概率。

现在考虑攻击游戏 2.4。从现在开始，所有的事件和概率都是关于这个游戏的。如果我们以 $b = 0$ 这一事件为条件，那么在这个条件概率空间中，挑战者和对手的所有其他随机选择的分布方式与攻击游戏 2.1 的实验 0 中的相应数值应该完全相同。因此，如果 $\hat{b}$ 是攻击游戏 2.4 中对手的输出，则有：

$$\Pr[\hat{b} = 1 \,|\, b = 0] = p_0$$

通过一个类似的论证，我们还可以得到：

$$\Pr[\hat{b} = 1 \,|\, b = 1] = p_1$$

因此，我们有：

$$\begin{aligned}
\Pr[\hat{b} = b] &= \Pr[\hat{b} = b \,|\, b = 0] \cdot \Pr[b = 0] + \Pr[\hat{b} = b \,|\, b = 1] \cdot \Pr[b = 1] \\
&= \Pr[\hat{b} = 0 \,|\, b = 0] \cdot \frac{1}{2} + \Pr[\hat{b} = b \,|\, b = 1] \cdot \frac{1}{2} \\
&= \frac{1}{2}(1 - \Pr[\hat{b} = 1 \,|\, b = 0] + \Pr[\hat{b} = 1 \,|\, b = 1]) \\
&= \frac{1}{2}(1 - p_0 + p_1)
\end{aligned}$$

因此：

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}] = |\Pr[\hat{b} = b] - \frac{1}{2}| = \frac{1}{2}|p_1 - p_0| = \frac{1}{2} \cdot \mathrm{SSadv}[\mathcal{B}, \mathcal{E}]$$

这就证明了该定理。 □

就像把 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]$ 称为 $\mathcal{A}$ 的“语义安全优势”一样，我们将把 $\mathsf{SSadv}^*[\mathcal{A},\mathcal{E}]$ 称为 $\mathcal{A}$ 的“比特猜测语义安全优势”。

2.2.5.1 一种推广

事实证明，上面的情况是相当普遍的。虽然我们在本章中不需要它，但为了将来参考，我们在此指出如何推广上述情况到更一般的场景。我们可能会遇到一些情况，对于一些密码学系统（称为“$\mathcal{S}$”），其中的一些特定的安全属性（称为“X”）可以用涉及两个实验（实验 0 和实验 1）的攻击游戏来定义，其中对手 $\mathcal{A}$ 的协议在两个实验中是相同的，而挑战者的协议则是不同的。对于 $b=0,1$，我们定义 W_b 为 $\mathcal{A}$ 在实验 b 中输出为 1 的事件，我们还定义：

$$\mathrm{Xadv}[\mathcal{A},\mathcal{S}] := |\Pr[W_0] - \Pr[W_1]|$$

是 $\mathcal{A}$ 的“X 优势”。就像上面一样，我们总是可以定义该攻击游戏的比特猜测版本，此时，挑战者随机选择一个 $b\in\{0,1\}$，然后运行实验 b 作为它的协议。如果令 W 是对手 $\mathcal{A}$ 的输出为 b 的事件，那么我们定义：

$$\mathrm{Xadv}^*[\mathcal{A},\mathcal{S}] := |\Pr[W] - 1/2|$$

是 $\mathcal{A}$ 的“比特猜测 X 优势”。

使用与定理 2.10 的证明完全相同的计算方法，我们有：

$$\mathrm{Xadv}[\mathcal{A},\mathcal{S}] = 2\cdot\mathrm{Xadv}^*[\mathcal{A},\mathcal{S}] \tag{2.11}$$

2.3 数学细节

迄今为止，我们一直比较随意地使用*有效 (efficient)* 和*可忽略不计 (negligible)* 这两个术语，而没有给出它们的正式定义：

- 我们要求，一个计算性密码要有*有效*的加密和解密算法；
- 对于一个语义安全的密码，我们要求任何*有效*对手在攻击游戏 2.1 中的优势*可忽略不计*。

本节的目标是为这些术语提供精确的数学定义。虽然这些定义为密码学作为一门数学学科的研究提供了一个令人满意的理论框架，但我们应该警告读者：

- 这些定义相当复杂，需要大量的符号；并且
- 这些定义只是非常粗略地模拟了我们对这些术语的直观理解。

我们强调，读者可以安全地跳过这一节，而不会在理解上遭受重大损失。在进入正式定义之前，让我们提醒读者，我们在这些定义中试图抓住的东西：

- 首先，当我们谈到有效的加密或解密算法时，我们通常指的是一种运行速度非常快的算法，比如以每字节 10 到 100 个时钟周期的速度加密数据。

- 其次，当我们说到一个有效对手时，我们通常指的是一个能在某个大的，但仍然可行的时间（和其他计算资源）中运行的算法。通常情况下，我们假设一个试图破解密码学系统的对手愿意花费比密码学系统的用户多得多的资源。因此，10,000 台计算机并行运行 10 年，可以被视为一个坚定的、有耐心的、财力雄厚的对手的可行计算的上限。然而在某些情况下，比如在 2.2.4 小节中的网络轮盘赌的例子中，对手的计算时间可能受到很大的限制。

- 第三，当我们说对手的优势可以忽略不计时，我们的意思是它的优势是如此之小，以至于就所有实际目的而言，它都可以被视为等于零。正如我们在网络轮盘赌的例子中所看到的，如果在攻击游戏 2.1 中，没有一个有效对手拥有超过 2^{-100} 的优势，那么在实践中，就没有一个玩家可以将他在网络轮盘赌中获胜的几率提升到 2^{-100} 以上。

尽管我们对*有效*一词的直观理解取决于上下文，但我们的形式化定义不会做这样的区分。事实上，我们将采用计算复杂性理论的习惯，把*有效*算法的概念等同于（*概率性*）*多项式时间*算法的概念。无论好坏，这都给我们提供了一个独立于任何特定计算模型的具体细节的形式化框架。

2.3.1 可忽略不计、超多项式与多项式边界函数

我们首先定义可忽略不计、超多项式和多项式边界函数的概念。

直观地讲，一个可忽略不计函数 $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ 是一个不仅在 $n \to \infty$ 时趋近于 0，而且趋近于 0 的速度比任何多项式的倒数都要更快的函数。

定义 2.5. *如果对于所有 $c \in \mathbb{R}_{>0}$，都存在 $n_0 \in \mathbb{Z}_{\geq 1}$ 使得对于所有的整数 $n \geq n_0$，都有 $|f(n)| < 1/n^c$，我们就称函数 $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ 是**可忽略不计的** **(negligible)**。*

可忽略不计函数的另一种表征，也许更容易理解和使用，如下所示：

定理 2.11. *当且仅当对于所有 $c > 0$，都有：*

$$\lim_{n\to\infty} f(n)n^c = 0$$

时，函数 $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ 是可忽略不计的。

证明. 留作练习。 □

例 2.10. 一些不可忽略函数的例子：

$$2^{-n}, \quad 2^{-\sqrt{n}}, \quad n^{-\log n}$$

一些可忽略函数的例子：

$$\frac{1}{1000n^4 + n^2 \log n}, \quad \frac{1}{n^{100}}$$

当我们正式定义了 “可忽略不计” 这个术语后，定义 “超多项式” 就很容易了。

定义 2.6. *如果 $1/f$ 可忽略不计，我们就称函数 $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ 是**超多项式的** **(super-poly)**。*

本质上，一个多项式边界函数 $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ 是一个被某个多项式（绝对值）约束的函数。正式地：

定义 2.7. *如果存在 $c, d \in \mathbb{R}_{>0}$ 使得对于所有整数 $n \geq 0$，都有 $|f(n)| \leq n^c + d$，我们就称函数 $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ 是**多项式边界的** **(poly-bounded)**。*

请注意，如果 f 是一个多项式边界函数，那么 $1/f$ 必然不是一个可忽略不计函数。然而，正如下面的例子所说明的那样，请务必注意，不要得出错误的推论。

例 2.11. 定义 $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$，使得对于所有偶数 n，都有 $f(n) = 1/n$；对于所有奇数 n，都有 $f(n) = 2^{-n}$。那么 f 不是可忽略不计的，但 $1/f$ 既不是多项式边界的，也不是超多项式的。

2.3.2 计算性密码：形式上的问题

下面，我们讨论形式上的问题。我们首先澄清一个事实：当我们说一个计算性密码 $\mathcal{E} = (E, D)$ 定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上，其中 $\mathcal{K}$ 是密钥空间，$\mathcal{M}$ 是消息空间，$\mathcal{C}$ 是密文空间，而且每个上述空间都是有限集时，我们并没有说出全部的事实。在数学模型中（尽管在现实世界的系统中并不总是如此），我们将一个密码族 $\mathcal{E}$ 的密钥、消息和密文空间相关联，其索引为：

- 一个**安全参数 (security parameter)** 是一个正整数，用 λ 表示，和
- 一个**系统参数 (system parameter)** 是一个比特串，用 Λ 表示。

因此，现在 $\mathcal{K}, \mathcal{M}, \mathcal{C}$ 不再单是三个有限集，而是三个有限集族：

$$\{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \{\mathcal{C}_{\lambda,\Lambda}\}_{\lambda,\Lambda}$$

在本定义中，我们将其视为比特串的集合（可以通过一些规范的编码函数来表示数学对象）。

我们的想法是，当密码 $\mathcal{E}$ 被部署时，安全参数 λ 会被固定为某个值。一般来说，λ 的值越大，意味着安全水平就越高（即对拥有更多计算资源的对手的具备抵抗力），但也意味着更大的密钥长度，以及更慢的加解密速度。因此，安全参数就像一个我们可以转动的 “拨盘”，用于在安全和效率之间进行权衡。

一旦选定了 λ，系统参数 Λ 就会用一种特定于密码的算法生成。我们的想法是，系统参数 Λ（连同 λ）给出了一个固定密码实例的详细描述，即：

$$(\mathcal{K}, \mathcal{M}, \mathcal{C}) = (\mathcal{K}_{\lambda,\Lambda},\ \mathcal{M}_{\lambda,\Lambda},\ \mathcal{C}_{\lambda,\Lambda})$$

这一固定的实例可能会被部署在一个更大的系统中，并被多方使用，其中的 λ 和 Λ 的值都是公开的，（包括对手在内的）每个人都能知道。

例 2.12. 考虑例 2.4 中讨论的加性一次性密码本。这个密码使用了一个模数 n 作为描述。要部署这样一个密码，需要生成一个合适的模数 n，并将其公开（有时甚至会将其直接 “硬编码” 到实现密码的程序里）。模数 n 就是这个密码的系统参数。安全参数的每个特定值决定了 n 的长度，以比特为单位。n 本身可能由某些算法产生，这些算法可能是概率性的，其输出分布可能取决于预期的应用场景。例如，在某些场景下我们希望 n 是一个素数。

在进一步讨论之前，我们需要先定义有效算法的概念。为了这个目的，我们将只考虑以安全参数 λ 和总长度由 λ 的多项式约束的其他参数作为输入的算法 A。基本上，我们想说 A 的运行时间是以 λ 的多项式为界的，但如果 A 是概率性的，情况将更加复杂：

定义 2.8 (有效算法). 令 A 是一个算法（可能是概率性的），它以一个安全参数 $\lambda \in \mathbb{Z}_{\geq 1}$，以及其他被编码成的一个比特串 $x \in \{0,1\}^{\leq p(\lambda)}$ 的若干参数作为输入，其中 $p(\lambda)$ 是 λ 的一个固定的多项式。如果存在

一个多项式边界函数 t 和一个可忽略不计函数 ϵ，使得对于所有的 $\lambda \in \mathbb{Z}_{\geq 1}$ 和所有的 $x \in \{0,1\}^{\leq p(\lambda)}$，算法 A 在输入 (λ, x) 上的运行时间超过 $t(\lambda)$ 的概率最多为 $\epsilon(\lambda)$，我们就称 A 为一个**有效算法 *(efficient algorithm)***。

我们强调，上述定义中的概率是针对 A 的抛硬币而言的：对概率的这个约束必须对所有可能的输入 x 都成立。[1]

下面是一个正式定义，它抓住了由安全参数和系统参数进行参数化的系统的基本要求，并引入了更多的术语。在下面的定义中，我们用 $\mathrm{Supp}(P(\lambda))$ 来指代分布 $P(\lambda)$ 的**支持 (support)**，即算法 P 在输入 λ 上所有可能输出的集合。

定义 2.9. 一个**系统参数化 *(system parameterization)*** 是一个有效概率性算法 P，它以一个给定的安全参数 $\lambda \in \mathbb{Z}_{\geq 1}$ 为输入，输出一个比特串 Λ，称为**系统参数**，其长度总是以 λ 的一个多项式为界。我们还定义以下术语：

- 一个由比特串的有限集 $\mathbf{S} = \{\mathcal{S}_{\lambda,\Lambda}\}_{\lambda,\Lambda}$ 组成的集合，其中 λ 在 $\mathbb{Z}_{\geq 1}$ 上运行，Λ 在 $\mathrm{Supp}(P(\lambda))$ 上运行，当每个集合 $\mathcal{S}_{\lambda,\Lambda}$ 中的所有比特串的长度都被 λ 的某个多项式 p 约束，我们就称 $\mathbf{S}$ 为**具有系统参数化 P 的空间族 *(family of spaces with system parameterization P)***。
- 如果存在一个有效确定性算法，在输入 $\lambda \in \mathbb{Z}_{\geq 1}$，$\Lambda \in \mathrm{Supp}(P(\lambda))$，$s \in \{0,1\}^{\leq p(\lambda)}$ 上能够确定 $s \in \mathcal{S}_{\lambda,\Lambda}$ 是否成立，就称 $\mathbf{S}$ 是**可有效识别的 *(efficiently recognizable)***。
- 如果存在一个有效概率性算法，在输入 $\lambda \in \mathbb{Z}_{\geq 1}$，$\Lambda \in \mathrm{Supp}(P(\lambda))$ 上能够输出一个均匀分布在 $\mathcal{S}_{\lambda,\Lambda}$ 上的元素，就称 $\mathbf{S}$ 是**可有效采样的 *(efficiently sampleable)***。
- 如果存在一个有效确定性算法，在输入 $\lambda \in \mathbb{Z}_{\geq 1}$，$\Lambda \in \mathrm{Supp}(P(\lambda))$，$s \in \mathcal{S}_{\lambda,\Lambda}$ 上能够输出一个非负整数，称为 s 的**长度**，则称 $\mathbf{S}$ **有一个有效长度函数 *(has an effective length function)***。

现在，我们就可以给出计算性密码完整且正式的定义：

定义 2.10 (计算性密码). 一个**计算性密码**由一对算法 E 和 D，以及三个具有系统参数化 P 的空间族：

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{C} = \{\mathcal{C}_{\lambda,\Lambda}\}_{\lambda,\Lambda}$$

组成，满足：

1. $\mathbf{K}$，$\mathbf{M}$ 和 $\mathbf{C}$ 都是可有效识别的。
2. $\mathbf{K}$ 是可有效采样的。
3. $\mathbf{M}$ 有一个有效长度函数。
4. 算法 E 是一个有效概率性算法，它接受 λ, Λ, k, m 作为输入，其中 $\lambda \in \mathbb{Z}_{\geq 1}$，$\Lambda \in \mathrm{Supp}(P(\lambda))$，$k \in \mathcal{K}_{\lambda,\Lambda}$，$m \in \mathcal{M}_{\lambda,\Lambda}$，总是输出一个 $\mathcal{C}_{\lambda,\Lambda}$ 中的元素。

[1] 我们不坚持概率性算法在指定的时间范围内以 1 的概率停止，这样能给我们一点“回旋的余地”，使得我们能够轻松地执行某些类型的随机抽样程序，这些程序没有先验的运行时间限制，但不太可能运行非常长的时间（例如扔硬币直到出现正面）。另一种方法是约束预期的运行时间，但由于技术原因，这被证明是有问题的。

请注意，这个有效算法的定义并不要求该算法在所有输入上都以 1 的概率停止。一个以 $2^{-\lambda}$ 的概率进入无限循环的算法也可以满足这个定义，即使它不是以 1 的概率停止。这些问题是相当传统的。一般来说，我们只考虑在所有输入上都能以 1 的概率停止的算法：这可以更自然地被看作是对算法的输出分布的要求，而不是对其运行时间的要求。

5. 算法 D 是一个有效确定性算法，它接受 λ, Λ, k, c 作为输入，其中 $\lambda \in \mathbb{Z}_{\geq 1}$，$\Lambda \in \mathrm{Supp}(P(\lambda))$，$k \in \mathcal{K}_{\lambda,\Lambda}$，$c \in \mathcal{C}_{\lambda,\Lambda}$，输出一个 $\mathcal{M}_{\lambda,\Lambda}$ 中的元素，或一个特殊符号 reject $\notin \mathcal{M}_{\lambda,\Lambda}$。

6. 对于所有的 $\lambda, \Lambda, k, m, c$，其中 $\lambda \in \mathbb{Z}_{\geq 1}$，$\Lambda \in \mathrm{Supp}(P(\lambda))$，$k \in \mathcal{K}_{\lambda,\Lambda}$，$m \in \mathcal{M}_{\lambda,\Lambda}$，如果 $c \in \mathrm{Supp}(E(\lambda, \Lambda; k, m))$，必有 $D(\lambda, \Lambda; k, c) = m$。

注意，在上述定义中，加密和解密算法接受 λ 和 Λ 作为辅助输入。为了与 2.2.1 小节中所介绍的符号保持一定的一致性，我们把它们写成 $E(\lambda, \Lambda; \cdots)$ 和 $D(\lambda, \Lambda; \cdots)$。

例 2.13. 考虑加性一次性密码本（见例 2.12）。在我们的正式框架中，安全参数 λ 决定了模数 n 的比特长度 $L(\lambda)$，也就是系统参数。系统参数生成算法将 λ 作为输入，生成长度为 $L(\lambda)$ 的模数 n。函数 $L(\cdot)$ 应该是一个多项式边界函数。有了这个假设，很明显，系统参数生成算法满足对它的要求。对密钥、消息和密文空间的要求也得到了满足：

1. 这些空间中的元素具有多项式边界长度：这是因为我们假设 $L(\cdot)$ 是一个多项式边界函数。

2. 密钥空间是可有效采样的：只需选择 $k \xleftarrow{R} \{0, \ldots, n-1\}$。

3. 密钥、信息和密文空间都是可有效识别的：只需测试一个比特串 s 是否是 0 和 $n-1$ 之间的某个整数的二进制编码即可。

4. 消息空间有一个有效长度函数：只需输出（比如）0 即可。

我们注意到，有些密码（例如一次性密码本）可能不需要系统参数。在这种情况下，我们可以假装系统参数是空字符串。我们还注意到，有些密码也并没有真正的安全参数；事实上，很多行业标准的密码根本就是现成的，有一个固定的密钥大小，没有可以调整的安全参数。这根本就是理论与实践的不匹配，但事实就是如此。

这就完成了我们对计算性密码的正式数学描述，并包含所有的细节。[2] 希望读者能够理解，虽然这些形式化的东西可以让我们给出数学上的精确和有意义的陈述，但它们并不是很有启发性，而且大多是为了掩盖真正发生的事情。因此，在正文中，我们将继续使用 2.2.1 小节的简化术语和符号来讨论密码，但需要理解，在本节讨论的形式化框架中，所有的陈述都有适当而自然的解释。这会是一个在接下来的章节中反复出现的模式：我们将主要使用简化的术语讨论各种类型的密码方案，而不讨论安全参数和系统参数。这些数学细节会在单独的一小节中讨论，一般会遵循这里建立的相同的一般模式。

2.3.3 有效对手和攻击游戏

在定义语义安全的概念时，我们必须定义所谓的*有效对手 (efficient adversary)* 是什么意思。由于这个概念将在本书中被广泛使用，我们要在这里提出一个更一般的框架。

对于任何类型的加密方案，我们都需要使用一个攻击游戏来定义其安全性，这个攻击游戏通常在一个对手 $\mathcal{A}$ 和他的挑战者之间进行。$\mathcal{A}$ 遵循一个任意的协议，而挑战者遵循一个简单且固定的协议，该协议由所讨论的密码方案及其安全概念所决定。此外，对手和挑战者都将一个共同的安全参数 λ 作为输入，挑战者在游戏开始时会计算出一个相应的系统参数 Λ，并将其发送给对手 $\mathcal{A}$。

[2] 请注意，2.2.1 小节中对香农密码的定义保持不变。在 2.2.1 小节末尾提出的 “任何确定性计算性密码也是一个香农密码” 的说法需要一个更恰当的解释：对于每个 λ 和 Λ，我们得到一个定义在 $(\mathcal{K}_{\lambda,\Lambda}, \mathcal{M}_{\lambda,\Lambda}, \mathcal{C}_{\lambda,\Lambda})$ 上的香农密码。

为了对这些类型的交互进行建模，我们引入**交互式机器 (interactive machine)** 的概念。在这样的机器 M 启动之前，它总是会得到写在一个特殊缓冲区中的安全参数 λ，并且其内部状态的其他部分会被初始化为某个默认值。机器 M 还有两个特殊的缓冲区：一个*传入消息缓冲区 (incoming message buffer)* 和一个*传出消息缓冲区 (outcoming message buffer)*。机器 M 可以被多次调用：当 M 的外部环境向 M 的传入消息缓冲区写入一个字符串时，新的调用就开始了；M 读取该消息，进行一些计算，更新其内部状态，并在其传出消息缓冲区上写入一个字符串，最后结束调用，将传出消息交给外部环境。因此，M 通过一个简单的消息传递系统与它的环境进行交互。我们假设 M 可以在它的最后一个传出消息中嵌入一些信号，以此来表示它已经停止，随后 M 会忽略任何调用它的尝试。

我们假设传入和传出机器 M 的消息都被限制为*恒定的长度*。这其实并不是一个真正的限制，因为我们总是可以通过发送许多较短的消息来模拟一个长消息的传输。然而，做出这样的限制可以简化一些技术问题。从现在开始，我们对对手和其他任何类型的交互式机器都施加这种长度限制。

对于任何给定的环境，我们可以通过计算 M 在所有调用中执行的步骤的数量来衡量 M 的**总运行时间 (total running time)**。这个运行时间不仅取决于 M 和它的随机选择，还取决于 M 运行的环境。[3]

定义 2.11 (有效交互式机器). *如果存在一个多项式边界函数 t 和一个可忽略不计函数 ϵ，使得对于所有环境（甚至不仅是计算上有边界的环境），M 的总运行时间超过 $t(\lambda)$ 的概率最多为 $\epsilon(\lambda)$，我们就称 M 是一个有效交互式机器 (efficient interactive machine)*。

自然而然地，我们可以将对手建模为一个交互式机器。一个**有效对手 (efficient adversary)** 就是一个有效交互式机器。

我们可以将两个交互式机器连接在一起，比如使用 M' 和 M 构建一个新的交互式机器 $M'' = \langle M', M\rangle$。从环境到 M'' 的消息总是会被转发到 M'。机器 M' 可以向环境发送消息，也可以向 M 发送消息；在后一种情况下，M 发送的输出消息会被发送到 M'。我们假设，如果 M 停机，那么 M' 就不会再向它发送任何消息。

因此，当 M'' 被调用时，其传入的消息会被转发到 M'，然后 M' 和 M 可能会交互若干次，然后，当 M' 向环境发送消息时，M'' 的调用就结束了。我们称 M' 为 “开放的” 机器（即它与外界交互），M 为 “封闭的” 机器（即它只与 M' 交互）。

自然，我们可以通过将两台这样的机器连接在一起来模拟挑战者和对手的交互，如上所述：挑战者为开放的机器，而对手为封闭的机器。

在我们的安全规约中，我们通常会展示如何使用能够破坏某个系统的对手 $\mathcal{A}$ 来构建能够破坏其他系统的对手 $\mathcal{B}$。我们想要的基本属性是，如果 $\mathcal{A}$ 是有效的，那么 $\mathcal{B}$ 也是有效的。然而，我们的规约几乎总是一种非常特殊的形式，即 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个包装器，它由位于 $\mathcal{B}$ 的挑战者与 $\mathcal{A}$ 的单个运行实例之间的某个简单而有效的 “接口层” 组成。

理想情况下，我们希望接口层的计算复杂度不依赖于 $\mathcal{A}$ 的计算复杂度；然而，某种程度的依赖性是不可避免的：$\mathcal{A}$ 向其挑战者进行的查询越多，接口层就必须执行越多的工作，但这种工作应该只是依赖于这种查询的数量，而不依赖于 $\mathcal{A}$ 的运行时间。

为了形式定义这一点，我们将 $\mathcal{B}$ 构建为一个组合式机器 $\langle M', M\rangle$，其中 M' 代表接口层（即 “开放的” 机器），M 代表 $\mathcal{A}$ 的实例（即 “封闭的“机器）。这就导出了下面的定义。

[3] 与脚注 1 的讨论类似，我们对有效交互式机器的定义并不要求它在所有环境下都以 1 的概率停机。这也是一个传统问题，但它将是我们考虑的任何机器的一个隐含要求。

定义 2.12 (基本包装器). 如果存在一个多项式边界函数 t 和一个可忽略不计函数 ϵ，使得对于所有的机器 M（不必是计算上有界的），当我们在一个任意的环境（同样，不必是计算上有界的）中执行组合式机器 $\langle M', M\rangle$ 时，下述属性成立：

> 在组合式机器 $\langle M', M\rangle$ 执行的每一点上，如果 I 是 M' 和 M 之间截至该点的交互次数，T 是 M' 截至该点的总运行时间，则 $T > t(\lambda + I)$ 的概率最大为 $\epsilon()$。

我们就称交互式机器 M' 为**一个有效接口 *(efficient interface)***。如果 M' 是一个有效接口，而 M 是任何机器，我们都称 $\langle M', M\rangle$ **是围绕 M 的一个基本包装器 *(elementary wrapper around M)***。

因此，当对手 $\mathcal{B}$ 能够像上面那样被结构化为一个与 $\mathcal{A}$ 交互的有效接口时，我们就称 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器。我们的定义被设计成很适合协同工作的形式，其突出的属性是：

- 如果 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，而 $\mathcal{A}$ 是有效的，那么 $\mathcal{B}$ 也是有效的。
- 如果 $\mathcal{C}$ 是围绕 $\mathcal{B}$ 的一个基本包装器，$\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，那么 $\mathcal{C}$ 是围绕 $\mathcal{A}$ 的一个基本包装器。

还要注意的是，在我们的攻击游戏中，挑战者通常满足我们对有效接口的定义。对于这样的挑战者和任何一个有效对手 $\mathcal{A}$，我们可以将它们的整个交互看作是一个单一的有效机器的交互。

查询有界对手。 在我们到目前为止所介绍的所有攻击游戏中，对手都只会发起固定数量的查询。在后面的章节中，我们将看到对手 $\mathcal{A}$ 被允许进行多次查询，而且它的查询次数没有先验约束。如果 $\mathcal{A}$ 是有效的，那么查询的次数会被某个多项式边界值 Q 所约束（至少以几乎可忽略不计的概率）。在证明这种攻击游戏的安全性时，为了设计一个围绕 $\mathcal{A}$ 的基本包装器 $\mathcal{B}$，我们通常会预先向 $\mathcal{B}$ 告知 $\mathcal{A}$ 最终会进行的查询次数的上界 Q。为了应用形式化框架，我们可以让 $\mathcal{A}$ 在一开始就发送一连串的 Q 个特殊消息，来向 $\mathcal{B}$ “通知” 该查询次数上界。如果这样做，那么 $\mathcal{B}$ 不仅可以在其逻辑中使用 Q 值，而且还可以在不违反定义 2.12 中的时间约束的情况下，以取决于 Q 的时间运行。这就使得 $\mathcal{B}$ 可以初始化一些大小可能依赖于 Q 的数据结构。当然，所有这些都只是一个理论上的 “hack”，用来绕过那些本来就过于严苛的技术约束，因此不应该太认真。我们在接下来的章节中都不会再明确说明这种 “通知机制”。

2.3.4 语义安全性：形式上的问题

在定义任何类型的安全性时，我们把对手在攻击游戏中的优势定义为一个函数 $\mathrm{Adv}(\lambda)$。它的值由攻击游戏中某些事件发生的概率来决定：对于 λ 的每个值，我们都能得到一个不同的概率空间，它由挑战者的随机选择和对手的随机选择共同决定。安全性意味着，对于每个有效对手，函数 $\mathrm{Adv}(\cdot)$ 的值都可忽略不计。

下面我们谈谈密码的语义安全性。在攻击游戏 2.1 中，我们定义了 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ 这个值。这个值实际上是安全参数 λ 的一个函数。对定义 2.2 的正确解释是，如果对于所有有效对手 $\mathcal{A}$（如上所述，它被建模为一个交互式机器）来说，安全参数 λ 的函数 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ 都可忽略不计（如定义 2.5 所定义的那样），则 $\mathcal{E}$ 是安全的。回顾一下，挑战者和对手都接受 λ 作为共同的输入。交互从挑战者开始，他将系统参数发送给对手。随后，对手将他的查询发送给挑战者，其中包含两条明文，而挑战者会用一条密文作为应答。最后，对手会输出一个比特（严格来说，在我们的形式化机器的模型中，这个 “输出” 是发送给挑战者的一条消息，然后挑战者停机）。$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ 的值由挑战者的随机选择（包括系统参数的选择）和对手的随机选择共同决定。图 2.6 展示了攻击游戏 2.1 的全貌。

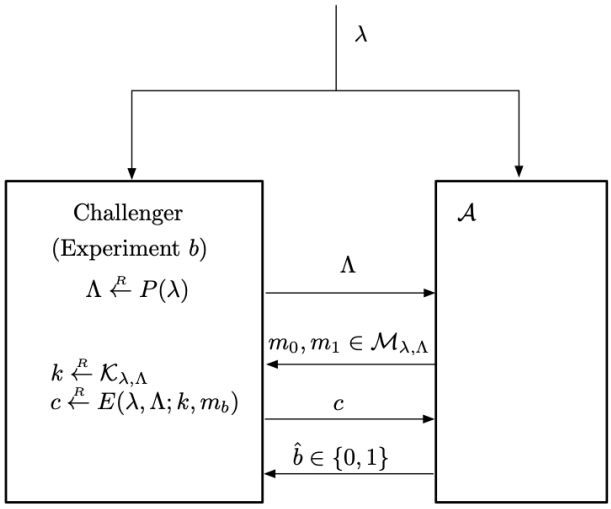


图 2.6: 攻击游戏 2.1 的完整详细版本

此外，在攻击游戏 2.1 中，我们要求对手给出的两条消息具有相同的长度，这意味着定义 2.10 第 3 部分给出的长度函数在两条消息上评估为相同的值。

根据这个形式上的定义，我们也可以讨论一下密码 $\mathcal{E}$ 的不安全到底意味着什么。这意味着存在一个有效对手 $\mathcal{A}$，使得 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]$ 是一个 λ 的不可忽略不计函数。即对于某个 $c>0$ 和无限多个安全参数 λ 的取值，有 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}] \geq 1/\lambda^c$ 成立。所以，不安全性并不意味着对于所有的安全参数 λ，对手 $\mathcal{A}$ 都能“攻破”密码 $\mathcal{E}$，而只是对于无限多个 λ，对手 $\mathcal{A}$ 能攻破密码 $\mathcal{E}$。

在正文中，我们一般会忽略安全参数、系统参数等。但读者需要明白，所有这些“简写”的背后都存在一个相应的精确的、形式化的数学表达。特别地，我们经常会称某个值 v 是可忽略不计的（多项式边界的），这实际上意味着 v 是安全参数的一个可忽略不计（多项式边界）函数。

2.4 一个有趣的应用：匿名路由

我们的老朋友 Alice 想向 Bob 发送一条消息 m，但她不想让 Bob 或其他人知道这条消息 m 是来自 Alice 的。比如说，Bob 可能在经营一个公共论坛，而 Alice 想在论坛上匿名发布一条评论。匿名发帖可以让 Alice 在不表明自己的真实身份的前提下自由地讨论健康问题等敏感议题。在本节中，我们假设 Alice 只想在论坛上发布一条消息。

一种方法是，Alice 选择一个代理，即 Carol，她将 m 发送给 Carol，并让 Carol 将消息转发给 Bob。这显然不能为 Alice 提供匿名性，因为任何监测网络的人都能看到 m 是由 Alice 发给 Carol，再由 Carol 发给 Bob 的。通过追踪 m 在网络中的路径，任何人都可以看到这个帖子来自于 Alice。

一种更好的方法是，Alice 与 Carol 建立一个共享密钥 k，并向 Carol 发送 $c := E(k,m)$，其中 $\mathcal{E}=(E,D)$ 是一个语义安全的密码。Carol 对 c 进行解密，并将 m 转发给 Bob。现在，监测网络的人将看到一条由 Alice 发给 Carol 的消息，以及另一条由 Carol 发给 Bob 的消息。然而，这种方法仍然不能确保 Alice 的匿名性：如果在某一天，Carol 收到的唯一消息就来自 Alice，而她发送的唯一消息是给 Bob 的，那么监测者就可以将这两者联系起来，仍然可以推断出发布的信息来自 Alice。

我们可以通过让 Carol 提供一个**混合服务** *(mixing service)* 来解决这个问题。所谓混合服务，就是

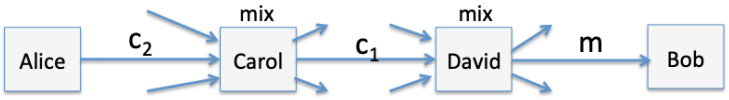


图 2.7: 一个使用两层混合的洋葱路由的例子

将来自许多不同参与方 $A_1, \ldots, A_n$ 的消息进行混合的服务。对于 $i = 1, \ldots, n$，Carol 与 A_i 建立一个密钥 k_i，每个参与方 A_i 都向 Carol 发送了一条加密消息 $c_i := E(k_i, \langle destination_i, m_i \rangle)$。Carol 收集所有 N 条传入的密文，用正确的密钥解密每条密文，并将得到的明文以某种随机的顺序转发到它们的目的地。现在，一个监测者在检查 Carol 的通信时，只能看到有 n 条消息传入和 n 条消息传出，但无法知道哪条消息被转发到了哪里。Alice 的消息是 Carol 发出的 n 条消息中的某一条，但监测者无法知道到底是哪一条。我们说 Alice 的**匿名集** *(anonymity set)* 大小为 n。

剩下的问题是，Carol 仍然知道 Alice 是在论坛发布某条特定消息的人。为了消除这最后的风险，Alice 可以使用多个混合服务，比如说 Carol 和 David。她与 Carol 建立一个密钥 k_c，与 David 建立一个密钥 k_d。为了向 Bob 发送她的消息，她按如下方式构建她的密文 c_2：

$$c_2 := E(k_c, E(k_d, m)) \tag{2.12}$$

完整起见，Alice 可能想在密文中嵌入路由信息，因此 c2 的实际构造为：

$$c_2 := E(k_c, \langle \mathsf{David}, c_1 \rangle) \quad \text{where} \quad c_1 := E(k_d, \langle \mathsf{Bob}, m \rangle)$$

接下来，Alice 将 c_2 发送给 Carol。Carol 对 c_2 进行解密并得到明文 $\langle \mathsf{David}, c_1 \rangle$，这示意她要把 c_1 发送给 David。David 解密 c_1 得到明文 $\langle \mathsf{Bob}, m \rangle$，示意他把 m 发送给 Bob。图 2.7 展示了这个解密嵌套密文的过程，它类似于一层一层地剥开洋葱。由于这个原因，这种路由过程通常被称为**洋葱路由** *(onion routing)*。

现在，即使 Carol 监测了所有的网络流量，也不能确定谁在 Bob 的论坛上发布了哪条消息。这对 David 来说也是如此。然而，如果 Carol 和 David 是串通的，他们就能搞清楚这些信息。由于这个原因，Alice 可能会通过两个以上的混合网络来传递她的信息。只要其中的一个混合服务不与其他混合服务共谋，Alice 就能够保护她的匿名性。

一个稍微有点复杂的情况是，当 Alice 与 David 建立共享秘钥 k_d 时，她不能向 David 透露她的身份。否则，David 就会知道 c_1 来自 Alice，这是我们不希望看到的。这个目标并不难实现，我们将在本书后面介绍如何做到这一点（见 21.1 节）。

嵌套加密的安全性。 为了保持 Alice 的匿名性，知道 k_c 的 Carol 不能从式 2.12 的嵌套密文 c_2 中获取任何关于 m 的信息。否则，Carol 就有可能利用她从 c_2 中了解到的关于 m 的信息，将 Alice 与她在 Bob 的论坛上发的帖子联系起来。比如说，假设 Carol 可以从 c_2 中了解到 m 的前几个字符，然后她发现 Bob 的论坛上只有一个帖子以这些字符开头。那么 Carol 就可以将整个帖子与 Alice 关联起来，因为她知道 c_2 是来自 Alice 的。

对 David 来说也是如此：知道 k_d 的 David 最好无法从式 2.12 的嵌套密文 c_2 中得知任何关于 m 的信息。

我们下面论证，如果 $\mathcal{E}$ 是语义安全的，那么在给定 c_2 和 k_c 或 k_d 两个密钥中的一个的情况下，任

何有效对手都无法了解任何关于 m 的信息。

一般地，对于定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的密码 $\mathcal{E} = (E, D)$，我们定义 n 层嵌套密码 $\mathcal{E}_n = (E_n, D_n)$ 如下：

$$E_n\big((k0, \ldots, k_{n-1}),\, m\big) = E\big(k_{n-1}, E(k_{n-2}, \cdots E(k_0, m)\cdots\big)$$

解密算法以相反的顺序使用密钥：

$$D_n\big((k0, \ldots, k_{n-1}),\, c\big) = D\big(k_0, E(k_1, \cdots E(k_{n-1}, m)\cdots\big)$$

我们的目标是表明，如果 $\mathcal{E}$ 是语义安全的，那么就算对手得到了除了 $k_0, \ldots, k_{n-1}$ 中某一个以外的所有密钥，$\mathcal{E}_n$ 也是语义安全的。为了更加精确，我们定义两个实验：实验 0 和实验 1，对于 $b = 0, 1$，实验 b 的工作方式如下：

- 对手将 (m_0, m_1, d) 交给挑战者，其中 $m_0, m_1 \in \mathcal{M}$ 是长度相等的两条消息，且 $0 \le d < n$。
- 挑战者选择 n 个密钥 $k_0, \ldots, k_{n-1} \xleftarrow{\mathrm{R}} \mathcal{K}$ 并计算 $c \xleftarrow{\mathrm{R}} E_n\big((k_0, \ldots, k_{n-1}),\, m_b\big)$。它将 c 和所有的密钥 $k_0, \ldots, k_{n-1}$ 一起发送给对手，但不包括密钥 k_{d}。
- 对手输出一个比特 $\hat{b} \in \{0, 1\}$。

这个游戏抓住了这样一个事实：对手得到除 k_{d} 以外的所有密钥 $k_0, \ldots, k_{n-1}$，并试图破坏语义安全性。

如同语义安全性的定义，我们定义对手的优势 $\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ 如下：

$$\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = \big|\Pr[W_0] - \Pr[W_1]\big|$$

其中 W_b 是 $\mathcal{A}$ 在实验 b 中输出 1 的事件，对于 $b = 0, 1$，如果 $\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ 可忽略不计，我们就说 $\mathcal{E}$ 对于 n 层嵌套是语义安全的。

定理 2.12. *对于每个常数 $n > 0$，如果 $\mathcal{E} = (E, D)$ 是语义安全的，那么 $\mathcal{E}$ 对于 n 层嵌套来说也是语义安全的。*

> *特别地，对于每个攻击 $\mathcal{E}_n$ 的 n 层嵌套对手 $\mathcal{A}$，都存在一个攻击 $\mathcal{E}$ 的语义安全对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足：*
>
> $$\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = \mathrm{SSadv}[\mathcal{B}, \mathcal{E}]$$

这个定理的证明很适合作为安全归约的练习。我们把它留给练习 2.15。

2.5 笔记

一次性密码本由吉尔伯特 · 弗纳姆于 1917 年提出，尽管也有证据表明它其实在那之前就已经被发现了。

对文献的引用有待补充。

2.6 练习

2.1 (乘性一次性密码本). 我们还可以定义一个 "乘法模 p" 的一次性密码本变体。这是一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的密码 $\mathcal{E}=(E,D)$，其中 $\mathcal{K}:=\mathcal{M}:=\mathcal{C}:=\{1,\ldots,p-1\}$，其中 p 是一个素数。加密和解密算法的定义如下：

$$E(k,m):=k\cdot m \mod p \qquad D(k,c):=k^{-1}\cdot c \mod p$$

这里，k^{-1} 表示 k 对 p 的模逆元。验证这个密码的正确性属性，并证明它是完美安全的。

2.2 (一个好的替换密码). 考虑例 2.3 中定义的替换密码 $\mathcal{E}=(E,D)$ 的一个变体，其中消息的每个符号都是用*独立的*置换来加密的。也就是说，令 $\mathcal{M}=\mathcal{C}=\Sigma^L$，其中 Σ 是某个有限符号表，L 是某个长度。令密钥空间为 $\mathcal{K}=S^L$，其中 S 是 Σ 上所有置换的集合。加密算法 $E(k,m)$ 的定义为：

$$E(k,m):=\Big(k[0](m[0]),\ k[1](m[1]),\ \ldots,\ k[L-1](m[L-1])\Big)$$

证明 $\mathcal{E}$ 是完美安全的。

2.3 (链式加密). 令 $\mathcal{E}=(E,D)$ 是一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的完美安全密码，其中 $\mathcal{K}=\mathcal{M}$。令 $\mathcal{E}'=(E',D')$ 是一个密码，其加密算法的定义为 $E'((k_1,k_2),m):=\big(E(k_1,k_2),E(k_2,m)\big)$。证明 $\mathcal{E}'$ 是完美安全的。

2.4 (被破坏的一次性密码本). 考虑一个消息空间为 $\{0,1\}^L$ 的一次性密码本变体，其密钥空间 $\mathcal{K}$ 被限制为所有包含偶数个 1 的 L 比特字符串所构成的集合。给出一个语义安全优势为 1 的有效对手。

2.5 (更强的不可能结果). 这个练习推广了香农定理（定理 2.5）。令 $\mathcal{E}$ 是一个定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的密码。假设 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]\le\epsilon$ 对*所有的*对手 $\mathcal{A}$ 都成立，甚至包括在*计算上无界的*对手。证明 $|\mathcal{K}|\ge(1-\epsilon)|\mathcal{M}|$。

2.6 (匹配的边界). 这个练习是对上一个练习的一种逆推。对于 $j=0,\ldots,L-1$，令 $\epsilon=1/2^j$。考虑定义在 $(\mathcal{K},\mathcal{M},\mathcal{C})$ 上的 L 比特一次性密码本变体 $\mathcal{E}$，其中 $\mathcal{M}=\mathcal{C}=\{0,1\}^L$。密钥空间 $\mathcal{K}$ 被限制为所有前 j 位不全为零的 L 比特字符串所构成的集合，因此 $|\mathcal{K}|=(1-\epsilon)|\mathcal{M}|$。证明：

(a) 存在一个有效对手 $\mathcal{A}$，使得 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]=\epsilon/(1-\epsilon)$；

(b) 对于*所有的*对手 $\mathcal{A}$，甚至包括*计算上无界的*对手，都有 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]\le\epsilon/(1-\epsilon)$。

注意： 由于 (a) 中 $\mathcal{A}$ 的优势是非零的，所以密码 $\mathcal{E}$ 不可能是完美安全的。

2.7 (确定性密码). 在这个练习中，我们要求你详细证明例 2.9 中的声称。也就是证明，如果 $\mathcal{E}$ 是一个完美安全的确定性密码，那么对于每个对手 $\mathcal{A}$ 都有 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]=0$（记住，$\mathcal{A}$ 可能是概率性的）；同时证明，如果 $\mathcal{E}$ 是变长一次性密码本，那么对于所有对手 $\mathcal{A}$ 都有 $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]=0$。

2.8 (轮盘赌). 在 2.2.4 小节中，我们论证了如果使用语义安全的密码对一个值 r 进行加密，那么玩家在网络轮盘赌中获胜的几率与真实轮盘赌非常接近。然而，我们的 "轮盘赌" 游戏相当简单。假设我们有一个更复杂的游戏，不同的结果可能会导致不同的赢利。规则并不那么重要，但假设规则很容易评估（给定一个赌注和数字 r），每个赌注的结果都是 $0,1,\ldots,n$ 美元，其中 n 是多项式边界的。令 μ 是这个游戏的真实版本（没有加密）的最佳策略中的预期赢利。令 μ' 是这个游戏的网络版本（有加密）中某个（有效）玩家的预期赢利。假设密码是语义安全的，证明 $\mu'\le\mu+\epsilon$，其中 ϵ 可忽略不计。

提示： 你可能需要利用这样一个事实：如果 X 是一个在 $\{0,1,\ldots,n\}$ 中取值的随机变量，那么 X 的期望等于 $\sum_{i=1}^{n}\Pr[\mathsf{X}\ge i]$。

2.9. 使用 2.3 节中的正式定义证明事实 2.6。

2.10 (练习语义安全性的定义). 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的语义安全密码，其中 $\mathcal{M} = \mathcal{C} = \{0,1\}^L$。以下哪种加密算法会产生语义安全的方案？要么给出一种攻击，要么通过明确的归约提供一个安全证明。

(a) $E_1(k, m) := 0 \,||\, E(k, m)$

(b) $E_2(k, m) := E(k, m) \,||\, \text{parity}(m)$

(c) $E_3(k, m) := \text{reverse}(E(k, m))$

(d) $E_4(k, m) := E(k, \text{reverse}(m))$

这里，对于一个比特串 s，如果 s 中有奇数个比特 1，则 parity(s) 为 1，否则为 0；另外，reverse(s) 是通过颠倒 s 中比特的顺序得到的字符串，例如，reverse(1011) = 1101。

2.11 (密钥恢复攻击). 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的密码。密钥恢复攻击用下面的挑战者与对手 $\mathcal{A}$ 之间的游戏建模：挑战者在 $\mathcal{K}$ 中选择一个随机密钥 k，在 $\mathcal{M}$ 中选择一个随机消息 m，计算 $c \xleftarrow{R} E(k, m)$，并将 (m, c) 发送给 $\mathcal{A}$。$\mathcal{A}$ 输出一个 $\mathcal{K}$ 中的猜测值 $\hat{k}$ 作为应答。如果 $D(\hat{k}, c) = m$，我们就称 $\mathcal{A}$ 赢得了游戏，并将 $\mathcal{A}$ 赢得游戏的概率定义为 $\text{KRadv}[\mathcal{A}, \mathcal{E}]$。和往常一样，如果对于所有有效对手 $\mathcal{A}$，优势 $KRadv[A, E]$ 都可忽略不计，我们就说 $\mathcal{E}$ 对于密钥恢复攻击是安全的。

(a) 证明一次性密码本对密钥恢复攻击是不安全的。

(b) 证明如果 $\mathcal{E}$ 是语义安全的，并且 $\epsilon = |\mathcal{K}|/|\mathcal{M}|$ 可忽略不计，那么 $\mathcal{E}$ 对密钥恢复攻击是安全的。特别地，证明对于每个有效的密钥恢复对手 $\mathcal{A}$，都存在一个有效的语义安全对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，使得：

$$\text{KRadv}[\mathcal{A}, \mathcal{E}] \leq \text{SSadv}[\mathcal{B}, \mathcal{E}] + \epsilon$$

提示：你的语义安全对手 $\mathcal{B}$ 在语义安全实验 0 中会以 $\text{KRadv}[\mathcal{A}, \mathcal{E}]$ 的概率输出 1，在实验 1 中以最多 ϵ 的概率输出 1。由此可以推导出用 ϵ 表示的 $\text{SSadv}[\mathcal{B}, \mathcal{E}]$ 的下界，并由该结果得到 $\text{KRadv}[\mathcal{A}, \mathcal{E}]$。

(c) 基于 (b) 证明，如果 $\mathcal{E}$ 在是语义安全的，并且 $|\mathcal{M}|$ 是超多项式的，那么 $|\mathcal{K}|$ 不可能是多项式边界的。

注意：当 $|\mathcal{M}|$ 是多项式边界的时候，$|\mathcal{K}|$ 可以是多项式边界的，比如在一次性密码本中。

2.12 (对消息恢复的安全性). 在 2.2.3.1 小节中，我们提出了针对消息恢复的安全性的概念。构建一个对消息恢复安全，但不是语义安全的密码。

2.13 (在简单环境下计算优势). 考虑以下两个实验：实验 0 和实验 1：

- 在实验 0 中，挑战者抛出一枚公平的硬币（即抛出正面和反面的概率都是 1/2），并将结果发送给对手 $\mathcal{A}$。
- 在实验 1 中，挑战者总是向对手发送 “反面”。

对手的目标是区分这两个实验：在每次实验结束时，对手输出一个比特 0 或 1，作为其对所处实验的猜测。对于 $b = 0, 1$，令 W_b 为实验 b 中对手输出 1 的事件。对手试图使其区分优势：

$$\big|\Pr[W_0] - \Pr[W_1]\big| \in [0, 1]$$

最大化。如果这个优势对所有有效对手来说都是可忽略不计的，那么我们就称这两个实验是无法区分的。

(a) 计算下列对手的优势：

(i) $\mathcal{A}_1$：总是输出 1。

(ii) $\mathcal{A}_2$：忽略挑战者报告的结果，以相等的概率随机输出 0 或 1。

(iii) $\mathcal{A}_3$：如果从挑战者收到 “正面”，则输出 1，否则输出 0。

(iv) $\mathcal{A}_4$：如果从挑战者收到 “正面”，则输出 0，否则输出 1。

(v) $\mathcal{A}_5$：如果收到 “正面”，则输出 1；如果收到 “反面”，则以相等的概率随机输出 0 或 1。

(b) 区分这两个实验的最大优势是多少？给出解释。

2.14 (置换密码). 考虑一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的密码 (E, D)，其中 $\mathcal{C} = \mathcal{M} = \{0,1\}^\ell$，$\mathcal{K}$ 是集合 $\{0, \ldots, \ell\}$ 的所有 $\ell!$ 个置换排列的集合。对于密钥 $k \in \mathcal{K}$ 和消息 $m \in \mathcal{M}$，定义 $E(k, m)$ 为使用置换 k 对 m 的所有比特进行重排列的结果，即 $E(k, m) = m[k(0)] \ldots m[k(\ell - 1)]$。通过展示一个优势为 1 的对手来证明改密码不是语义安全的。

2.15 (嵌套加密). 对于一个密码 $\mathcal{E} = (E, D)$，定义嵌套密码 $\mathcal{E}' = (E', D')$ 为：

$$E'\big((k_0, k_1), m\big) = E\big(k_1, E(k_0, m)\big), \quad D'\big((k_0, k_1), c\big) = D\big(k_0, D(k_1, c)\big)$$

我们的目标是证明，如果 $\mathcal{E}$ 是语义安全的，那么就算对手得到了 k_0 或 k_1 中的任意一个密钥，$\mathcal{E}'$ 也是语义安全的。

(a) 考虑以下语义安全实验，即实验 0 和实验 1：在实验 b 中，对于 $b = 0, 1$，对手产生两条消息 m_0 和 m_1，并得到 k_1 和 $E'\big((k_0, k_1), m_b\big)$。对手输出一个 $\{0,1\}$ 中的值 $\hat{b}$。同之前在语义安全的定义中类似，我们定义对手的优势为 $\mathrm{NEadv}[\mathcal{A}, \mathcal{E}]$。证明对于每个攻击 $\mathcal{E}'$ 的嵌套加密对手 $\mathcal{A}$，都存在一个攻击 $\mathcal{E}$ 的语义安全对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足：

$$\mathrm{NEadv}[\mathcal{A}, \mathcal{E}] = \mathrm{SSadv}[\mathcal{B}, \mathcal{E}]$$

画一个图，$\mathcal{A}$ 在右边，$\mathcal{B}$ 在中间，$\mathcal{B}$ 的挑战者在左边。在你的安全证明中，展示发生在上述三方之间的信息流。

(b) 如果在 (a) 中，对手在实验 0 和实验 1 中得到的是 k_0（而不是 k_1）和 $E'\big((k_0, k_1), m_b\big)$，重复 (a) 中的证明。像 (a) 一样，画一张图展示安全证明中的信息流。

这个问题出现在了 2.4 节中讨论的互联网匿名路由中。

2.16 (自指加密). 证明用一个密钥加密其自身是危险的。令 $\mathcal{E}$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的语义安全密码，其中 $\mathcal{K} \subseteq \mathcal{M}$，并令 $k \xleftarrow{R} \mathcal{K}$。一个密文 $c_* := E(k, k)$，即用 k 加密 k，被称为**自指加密 (self-referential encryption)**。

(a) 构建一个派生自 $\mathcal{E}$ 的密码 $\tilde{\mathcal{E}} = (\tilde{E}, \tilde{D})$，使得 $\tilde{E}$ 在是语义安全的，但如果对手得到了 $\tilde{E}(k,k)$，则变得不安全。这证明了语义安全并不意味着加密自己的密钥也是安全的。

(b) 构建一个派生自 $\mathcal{E}$ 的密码 $\hat{\mathcal{E}} = (\hat{E}, \hat{D})$，使得 $\hat{E}$ 是语义安全的，并且即使对手得到了 $\hat{E}(k,k)$，也能（被证明）保持语义安全性。为了证明 $\hat{E}$ 是语义安全的，你应该证明：对于每个攻击 $\hat{\mathcal{E}}$ 的对手 $\mathcal{A}$，都存在一个攻击 $\mathcal{E}$ 的对手 $\mathcal{B}$，使得 (i) $\mathcal{B}$ 的运行时间与 $\mathcal{A}$ 差不多，(ii) $\mathsf{SSadv}[\mathcal{A}, \hat{\mathcal{E}}] \leq \mathsf{SSadv}[\mathcal{B}, \mathcal{E}]$。

2.17 (压缩并加密). 有两个标准委员会提议将压缩与加密相结合以节省带宽（如 zip 和 gzip 程序中使用的 Lempel-Ziv 算法）。这两个委员会都计划使用变长一次性密码本进行加密。

- 一个委员会提议先压缩后加密。解释一下为什么这是个坏主意。

  提示：回顾一下，压缩可以大大缩减某些消息的大小，而对某些消息的长度影响不大。

- 另一个委员会提议先加密后压缩。解释一下为什么这也是个坏主意。

多年来，在结合加密和压缩时出现过许多问题。CRIME 和 BREACH 攻击就是很有代表性的例子。

2.18 (投票协议). 这个练习提供了一种基于加性一次性密码本的简易投票协议（例 2.4）。假设我们有 t 个投票者和一个计票中心。每个投票者都要投 0 或 1 票，计票中心要统计票数并公布总和 S。然而，他们将使用一个协议，保证没有任何一方（投票者或计票中心）能知道 S 之外的信息（但我们假定每一方都忠实地遵守该协议）。

该协议的工作方式如下。令 $n > t$ 是一个整数。计票中心产生一个对 0 的加密：$c_0 \xleftarrow{R} \{0, \ldots, n-1\}$，并将 c_0 发送给投票者 1。投票者 1 将他的票 v_1 加到 c_0 上，计算 $c_1 \leftarrow c_0 + v_1 \mod n$，并将 c_1 发送给投票者 2。这样继续下去，每个投票者 i 把 v_i 加到 c_{i-1} 上，计算 $c_i \leftarrow c_{i-1} + v_i \mod n$，并把 c_i 发送给投票者 $i+1$，最后投票者 t 把 c_t 发回计票中心。计票中心计算总和为 $S \leftarrow c_t - c_0 \mod n$，并将 S 广播给所有投票者。

(a) 证明该协议能正确计算出总和。

(b) 证明该协议在以下意义上是完美安全的。对于投票者 $i = 1, \ldots, t$，定义 $View_i := (S, c_{i-1})$，它代表投票者 i 的 "观点"。我们还定义 $View_0 := (c_0, c_t)$ 代表计票中心的 "观点"。证明对于每个 $i = 0, \ldots, t$ 和 $S = 0, \ldots, t$，以下结论成立：

> 无论 $v_1, \ldots, v_t$ 的选择如何变化，在 $v_j \in \{0,1\}$ 和 $\sum_{j=1}^{t} v_j = S$ 的约束下，$View_i$ 的分布保持不变。

(c) 证明如果两个投票者 i 和 j 串通，他们就可以确定第三个投票者 k 的投票。你可以自由选择索引 i，j 和 k。

2.19 (双向分割密钥). 令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ 上的语义安全密码，其中 $\mathcal{K} = \{0,1\}^d$。假设我们希望将解密密文的能力分给两方，比如 Alice 和 Bob。这样，想要解密密文，这两方都必不可少。对于一个 $\mathcal{K}$ 上的随机密钥 k，随机从 $\mathcal{K}$ 中选择一个 r 并定义 $k_a := r$ 和 $k_b := k \oplus r$。现在，如果 Alice 和 Bob 在一起，他们就可以重建密钥 $k = k_a \oplus k_b$，然后计算 $D(k, c)$ 来解密密文 c。我们的目标是证明 Alice 和 Bob 都不能单独解密密文。

(a) 构建一个安全概念以捕捉对手在给定 Bob 的密钥 k_b 时打破语义安全性的优势。将这种双向密钥分割优势记为 $\mathsf{2KSadv}[\mathcal{A},\mathcal{E}]$。

(b) 证明对于每一个双向密钥分割对手 $\mathcal{A}$，都存在一个语义安全对手 $\mathcal{B}$，使得 $\mathsf{2KSadv}[\mathcal{A},\mathcal{E}] = \mathsf{SSadv}[\mathcal{B},\mathcal{E}]$。

2.20 (简单的秘密共享). 令 $\mathcal{E} = (E, D)$ 是一个语义安全的密码，密钥空间为 $\mathcal{K} = \{0,1\}^L$。一家银行希望将一个解密密钥 $k \in \{0,1\}^L$ 分成 p_0，p_1 和 p_2 三份，并且解密至少需要其中的两份。每一份都可以交给不同的银行主管保管，并且每次解密都必须要获得三人中的至少两人的部分。这样一来，即使其中一位高管病休，解密也能进行。

(a) 为了实现这个方案，银行产生两个随机数对 (k_0, k_0') 和 (k_1, k_1')，它们满足 $k_0 \oplus k_0' = k_1 \oplus k_1' = k$。银行应如何分配密钥份额，才能使得任何两个部分都能用于解密，但任何单独部分都无法完成任务？

提示： 第一个执行者得到的部分是 $p_0 := (k_0, k_1)$。

(b) 推广 (a) 的方案，使得解密需要 5 份中的 3 份。重组密钥只需要使用各部分的异或。任何两部分都不能透露关于密钥 k 的信息。

(c) 进一步推广，对于任何 $t < w$，设计出一个 "w 中选择 t 份" 的系统。我们将在 11.1 节中看到一个更好的解决该问题的方法。

2.21 (简单的门限解密). 令 $\mathcal{E} = (E, D)$ 是一个语义安全的密码，其密钥空间为 $\mathcal{K}$。在这个练习中，我们将设计一个系统，让银行能将一个密钥 k 分成 p_0，p_1 和 p_2 三份，并且解密至少需要其中的两份，就像练习 2.20 中那样。然而，解密时不需要在一个地方重新组建完整的密钥。

我们使用练习 2.15 中介绍的嵌套加密。在 $\mathcal{K}^4$ 中随机选择一个密钥 $k := (k_0, k_1, k_2, k_3)$，并对消息 m 进行加密，即：

$$c \xleftarrow{R} \Big(E\big(k_1, E(k_0, m)\big),\ E\big(k_3, E(k_2, m)\big)\Big)$$

(a) 构建 p_0，p_1 和 p_2 的分配方案，使得任何两个部分都能用于解密，但任何单独部分都无法完成任务。

提示： 第一个部分是 $p_0 := (k_0, k_3)$。

讨论： 假设持有 p_0 和 p_2 的实体可以进行解密。要解密密文 c，首先将 c 发送给持有 p_2 的实体以部分解密 c，然后将结果转发给持有 p_0 的实体以完成解密。这样一来解密就完成了，并不需要在一个地方先组建完整的密钥 k。

(b) 推广 (a) 的方案，使得解密需要 5 份中的 3 份。解释如何在不重新组合密钥的情况下进行解密。

一种加密方案，如果它的密钥可以被分成若干份，从而使解密需要 w 份中的 t 份，并且解密不需要在单一地点重新装配密钥，就被称为提供**门限解密 (threshold decryption)**。我们将在 11.1 节中看到一个更好的解决该问题的方法。

2.22 (偏差校正). 再考虑一下语义安全攻击游戏的比特猜测版本（即攻击游戏 2.4）。假设一个有效对手 $\mathcal{A}$ 能以 $1/2 + \epsilon$ 的概率赢得游戏（即猜中隐藏比特 b），其中 ϵ 不可忽略不计。请注意，ϵ 可以是正数或负数（可忽略不计的定义在绝对值上起作用）。我们的目标是证明存在另一个有效对手 $\mathcal{B}$ 能以 $1/2 + \epsilon'$ 的概率赢得游戏，其中 ϵ' 是一个不可忽略不计的正数。

(a) 考虑以下对手 $\mathcal{B}$，它将 $\mathcal{A}$ 作为攻击游戏 2.4 中的一个子程序，进行以下的两阶段攻击。在第一阶段，$\mathcal{B}$ 扮演 $\mathcal{A}$ 的挑战者，但 $\mathcal{B}$ 生成自己的隐藏比特 b_0 和自己的密钥 k_0，最终 $\mathcal{A}$ 输出其猜测比特 $\hat{b}_0$。注意，在这个阶段，$\mathcal{B}$ 在攻击游戏 2.4 中的挑战者根本没有参与。在第二阶段，$\mathcal{B}$ 重新启动 $\mathcal{A}$，并让 $\mathcal{A}$ 与攻击游戏 2.4 中 "真正的" 挑战者互动，最终 $\mathcal{A}$ 输出一个猜测比特 $\hat{b}$。当这种情况发生时，$\mathcal{B}$ 输出 $\hat{b} \oplus \hat{b}_0 \oplus b_0$。请注意，$\mathcal{A}$ 的这次运行完全独立于第一次运行——$\mathcal{A}$ 的硬币和系统参数在这两次运行中是独立产生的。

证明 $\mathcal{B}$ 赢得攻击游戏 2.4 的概率为 $1/2 + 2\epsilon^2$。

(b) 人们可能会倾向于如下论证。只要构造一个运行 $\mathcal{A}$ 的对抗者 $\mathcal{B}$，当 $\mathcal{A}$ 输出 $\hat{b}$ 时，对手 $\mathcal{B}$ 输出 $\hat{b} \oplus 1$。现在，我们不知道 ϵ 是正的还是负的。如果它是正的，那么 $\mathcal{A}$ 就满足我们的要求。如果它是负的，那么 $\mathcal{B}$ 就满足我们的要求。虽然我们不知道这两个对手中的哪一个满足我们的要求，但我们知道其中一个肯定满足，所以存在性就被证明了。

这个论证有什么问题？想要解释这一点，你需要了解有关安全参数的数学细节（见 2.3 节）。

(c) 你能想出另一个能以至少 $1/2 + |\epsilon|/2$ 的概率赢得比特猜测游戏的有效对手 $\mathcal{B}'$ 吗？你的对手 $\mathcal{B}'$ 的效率可以比 $\mathcal{B}$ 低。

提示：尝试多次运行对手 $\mathcal{B}$ 的第一阶段。

第三章　流密码

在上一章中，我们介绍了完美安全加密和语义安全加密的概念。完美安全性的问题在于，要实现它必须使用很长的密钥。语义安全性是作为一个较弱的安全概念被提出来的，它或许可以让我们建立使用合理的短密钥的安全密码；然而，我们还没有产生任何这样的具体密码。本章研究一种能做到这一点的密码：流密码。

3.1　伪随机生成器

回顾一下一次性密码本。在这里，密钥、消息和密文都是 L 比特字符串。然而，我们实际希望使用一个更短的密钥。因此，我们的想法是使用一个短的、ℓ 比特的 "种子" s 作为加密密钥，其中 ℓ 的长度比 L 短得多，然后将这个种子 "拉伸" 成一个较长的、L 比特的字符串，用来掩盖消息（和解除对密文的掩盖）。拉伸种子 s 需要使用某个有效的确定性算法 G，该算法能够将任意 ℓ 比特字符串映射成 L 比特字符串。因此，这种修改后的一次性密码本的密钥空间为 $\{0,1\}^\ell$，而消息空间与密文空间仍为 $\{0,1\}^L$。对于 $s \in \{0,1\}^\ell$ 和 $m, c \in \{0,1\}^L$，加密和解密的定义如下：

$$E(s,m) := G(s) \oplus m, \quad D(s,c) := G(s) \oplus c$$

这种修改后的一次性密码本被称作一种**流密码 (stream cipher)**，而函数 G 被称为**伪随机生成器 (pseudo-random generator)**。

如果 $\ell < L$，那么根据香农定理，这种流密码肯定不能实现完美安全性。但是如果 G 满足适当的安全属性，这种密码也能达到语义安全。假设 s 是一个随机的 ℓ 比特字符串，r 是一个随机的 L 比特字符串。直观地说，如果一个对手无法有效区分出 $G(s)$ 与 r 的区别，他应该就不能分辨出流密码与传统的一次性密码本之间的区别；此外，既然后者的密码是语义安全的，那么前者自然也是语义安全的。为了使上述推理更加严谨，我们需要将对手不能 "有效区分 $G(s)$ 与 r 之间的差异" 这一概念形式化。

一种用于区分伪随机字符串 $G(s)$ 和真随机字符串 r 的算法称为**统计检验 (statistical test)**。它把一个字符串作为输入，并输出 0 或 1。如果它在伪随机输入上输出 1 的概率与在真正随机输入上输出 1 的概率有显著差异，那么这样的检验就被称作是**有效的**。即便是相对较小的概率差异，比如说 1%，也被认为是显著的；事实上，即使只有 1% 的差异，如果我们能够获得几百个独立的样本，这些样本要么都是伪随机的，要么就都是真随机的，那么我们就能够很有把握地推断出我们所看到的到底是伪随机字符串还是真随机字符串。但是，一个非零但是可忽略不计的概率差异，比如 2^{-100}，就是没有帮助的。

如何去设计一个有效的统计检验呢？一个基本的方法是：给定一个 L 比特字符串，计算一些统计量，然后看看如果我们认为这个字符串是真随机的话，这个统计量与我们的预期是否存在较大的差异。

例如，一个非常简单的、很容易计算的统计量是字符串中出现的 1 的次数 k。对于一个真随机的字

符串，我们认为 $k \approx L/2$。如果 PRG G 对 1 或者 0 有一些偏向性，我们就可以通过一个统计检验有效地检测到这一点，比如说，如果 $|k - 0.5L| < 0.01L$，则输出 1，否则就输出 0。

前面例子中的检验还可以加强，我们不仅仅可以考虑单个比特，还能够考虑成对的比特。我们可以将 L 比特字符串分解成 $\approx L/2$ 个比特对，并统计 00 对出现的次数 k_{00}，01 对出现的次数 k_{01}，10 对出现的次数 k_{10}，以及 11 对出现的次数 k_{11}。对于一个真随机字符串，我们会预期上面的这四个统计量都 $\approx L/8$。因此，一个自然的统计检验是检验这些统计量与 $L/8$ 的偏差是否小于某个指定界限。或者，我们也可以将这些偏差的平方相加，并检验这个和是否小于某个指定的界限，这就是统计学中经典的 χ 方检验。

我们还可以检查其他的许多简单的统计量。然而，像这样的简单检验并没有倾向于利用算法 G 的更深层次的数学特性，而恶意对手在设计专门针对 G 的攻击时可能会利用这些特性。比如说，对于有一些 PRG，上面的简单检验对其完全无效，但当给定足够多的输出比特时，其模式是完全可以预测的；也就是说，给定 $G(s)$ 的一个足够长的前缀，对手就可以计算出 $G(s)$ 的所有剩余比特，甚至能够计算出种子 s 本身。

我们对 PRG 的安全性的定义正式给出了不应该有有效的（和可有效计算的）统计检验的概念。

3.1.1 伪随机生成器的定义

伪随机生成器 (pseudo-random generator)，简称 **PRG**，是一个有效确定性算法 G，以一个种子 s 作为输入，计算并输出一个值 r。种子 s 来自一个有限的种子空间 $\mathcal{S}$，而输出 r 属于一个有限的输出空间 $\mathcal{R}$。通常情况下，$\mathcal{S}$ 和 $\mathcal{R}$ 是一些定长比特串的集合（例如，在上面的讨论中，我们有 $\mathcal{S} = \{0,1\}^\ell$ 和 $\mathcal{R} = \{0,1\}^L$）。我们称 G 是一个定义在 $(\mathcal{S}, \mathcal{R})$ 上的 PRG。

我们对 PRG 的安全定义抓住了这样一个直观的概念：如果 s 是从 $\mathcal{S}$ 中随机选择的，而 r 是从 $\mathcal{R}$ 中随机选择的，那么任何有效对手都无法有效地分辨 $G(s)$ 和 r 之间的区别：两者是**计算上不可区分 (computationally indistinguishable)** 的。该定义被表述为一个攻击游戏。

攻击游戏 3.1 (伪随机生成器). 对于一个定义在 $(\mathcal{S}, \mathcal{R})$ 上的给定 PRG G 和一个给定的对手 $\mathcal{A}$，我们定义两个实验：实验 0 和实验 1。对于 $b = 0, 1$，我们定义：

实验 b:

- 挑战者按如下方式计算 $r \in \mathcal{R}$：
  - 如果 $b = 0$：$s \xleftarrow{R} \mathcal{S}$，$r \leftarrow G(s)$；
  - 如果 $b = 1$：$r \xleftarrow{R} \mathcal{R}$。

  然后将 r 发送给对手。
- 给定 r，对手计算并输出一个比特 $\hat{b} \in \{0,1\}$。

对于 $b = 0, 1$，定义 W_b 是 $\mathcal{A}$ 在实验 b 中输出 1 的事件。我们定义 $\mathcal{A}$ 对于 G 的**优势**为：

$$\mathrm{PRGadv}[\mathcal{A}, G] := |\Pr[W_0] - \Pr[W_1]|$$

该攻击游戏如图 3.1 所示。

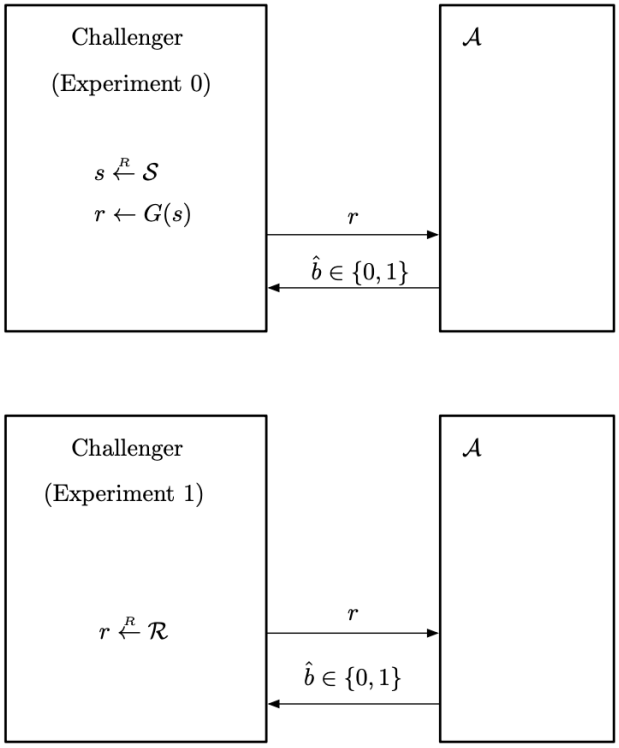


图 3.1: 攻击游戏 3.1 的实验 0 和实验 1

定义 3.1 (安全的 PRG). 如果 $\mathsf{PRGadv}[\mathcal{A}, G]$ 的值对于所有有效对手 $\mathcal{A}$ 来说都可忽略不计，那么 *PRG* G 就是**安全的**。

正如 2.2.5 小节所讨论的，攻击游戏 3.1 可以被重构为一个 “比特猜测” 游戏，挑战者不再进行两个独立的实验，而是随机选择 $b \in \{0,1\}$，然后针对对手 $\mathcal{A}$ 运行实验 b。在这个游戏中，我们记 $\mathcal{A}$ 的比特猜测优势 $\mathsf{PRGadv}^*[\mathcal{A}, G]$ 为 $|\Pr[\hat{b} = b] - 1/2|$。2.2.5 小节中的一般结论（即式 2.11）在此也适用：

$$\mathsf{PRGadv}[\mathcal{A}, G] = 2 \cdot \mathsf{PRGadv}^*[\mathcal{A}, G] \tag{3.1}$$

我们还注意到，只有当种子空间的势 (cardinality) 是超多项式的时候，PRG 才是安全的（见练习 3.5）。

3.1.2 数学细节

如 2.3 节一样，我们在这里给出更多与 PRG 有关的数学细节。就像 2.3 节一样，读者在初读这一小节时可以安全地跳过，在理解上也不会有什么损失。

首先，我们使用定义 2.9 中介绍的术语来精确定义 PRG。

定义 3.2 (伪随机生成器). 一个伪随机生成器包括一个算法 G 和两个具有系统参数化 P 的空间族：

$$\mathbf{S} = \{\mathcal{S}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{R} = \{\mathcal{R}_{\lambda,\Lambda}\}_{\lambda,\Lambda}$$

满足：

1. **S** 和 **R** 是可有效识别和可有效采样的。

2. 算法 G 是一个有效确定性算法，对于输入 λ, Λ, s，其中 $\lambda \in \mathbb{Z}_{\geq 1}$，$\Lambda \in \mathrm{Supp}(P(\lambda))$，$s \in \mathcal{S}_{\lambda,\Lambda}$，它输出 $\mathcal{R}_{\lambda,\Lambda}$ 中的一个元素。

接下来，我们需要对定义 3.1 进行正确的解释。首先，在攻击游戏 3.1 中，需要理解的是，对于安全参数 λ 的每个可能取值，我们都能得到一个不同的概率空间，该空间由挑战者的随机选择和对手的随机选择共同决定。其次，挑战者会产生一个系统参数 Λ，并在游戏一开始时就将其发送给对手。第三，优势 $\mathrm{PRGadv}[\mathcal{A}, G]$ 是安全参数 λ 的一个函数，安全性则意味着该函数是一个可忽略不计函数。

3.2 流密码：使用 PRG 进行加密

令 G 是一个定义在 $(\{0,1\}^{\ell}, \{0,1\}^{L})$ 上的 PRG；也就是说，G 将一个 ℓ 比特的种子拉伸为一个 L 比特的输出。由 G 构建的**流密码** $\mathcal{E} = (E, D)$ 定义在 $(\{0,1\}^{\ell}, \{0,1\}^{\leq L}, \{0,1\}^{\leq L})$ 上；对于 $s \in \{0,1\}^{\ell}$ 和 $m, c \in \{0,1\}^{\leq L}$，加密和解密定义如下：如果 $|m| = v$，则：

$$E(s, m) = G(s)[0 \ldots v-1] \oplus m$$

如果 $|c| = v$，则：

$$D(s, c) = G(s)[0 \ldots v-1] \oplus c$$

读者很容易验证，上述加解密方案能够满足我们对密码的定义（特别是满足了正确性属性）。

请注意，为了分析 $\mathcal{E}$ 的语义安全性，在攻击游戏 2.1 中与信息 m 相关的长度就是 m 的自然比特长度 $|m|$。此外还需注意，如果 v 比 L 小得多，那么对于许多实际的 PRG 来说，计算 $G(s)$ 的前 v 位有可能比计算出 $G(s)$ 的所有位然后截断要快得多。

本节的主要结论如下：

定理 3.1. *如果 G 是一个安全的 PRG，那么由 G 构建的流密码 $\mathcal{E}$ 就是一个语义安全的密码。*

特别地，对于每个按照攻击游戏 2.1 攻击 $\mathcal{E}$ 的语义安全对手 $\mathcal{A}$ 来说，都存在一个按照攻击游戏 3.1 攻击 G 的 PRG 对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足：

$$\mathrm{SSadv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \mathrm{PRGadv}[\mathcal{B}, G] \tag{3.2}$$

证明思路. 基本思想是，我们可以用一个真随机字符串来替换 PRG 的输出，而不会使得对手的优势增长超过一个不可忽略不计的量。然而，在进行这种替换后，对手的优势为零。 □

证明. 令 $\mathcal{A}$ 是一个如攻击游戏 2.1 中那样的攻击 $\mathcal{E}$ 的有效对手。我们想要证明，如果 G 是一个安全的 PRG，那么 $\mathrm{SSadv}[\mathcal{A}, \mathcal{E}]$ 可以忽略不计。用语义安全攻击游戏的比特猜测版本更为方便。我们证明：

$$\mathrm{SSadv}^{*}[\mathcal{A}, \mathcal{E}] = \mathrm{PRGadv}[\mathcal{B}, G] \tag{3.3}$$

对于某个有效对手 $\mathcal{B}$ 成立。那么式 3.2 就可以由定理 2.10 推出。此外，由于我们假设 G 是一个安全的 PRG，那么 $\mathrm{PRGadv}[\mathcal{B}, G]$ 的值一定是可忽略不计的，因此 $\mathrm{SSadv}[\mathcal{A}, \mathcal{E}]$ 也是可忽略不计的。

所以，考虑对手 $\mathcal{A}$ 在攻击游戏 2.1 的比特猜测版本中对 $\mathcal{E}$ 的攻击。在这个游戏中，$\mathcal{A}$ 向挑战者提供两个相同长度的消息 m_0 和 m_1，然后挑战者选择一个随机密钥 s 和一个随机比特 b，并在 s 下对 m_b 进行加密，将得到的密文 c 交给 $\mathcal{A}$；最后，$\mathcal{A}$ 输出一个比特 $\hat{b}$。如果 $\hat{b} = b$，则对手 $\mathcal{A}$ 赢得游戏。

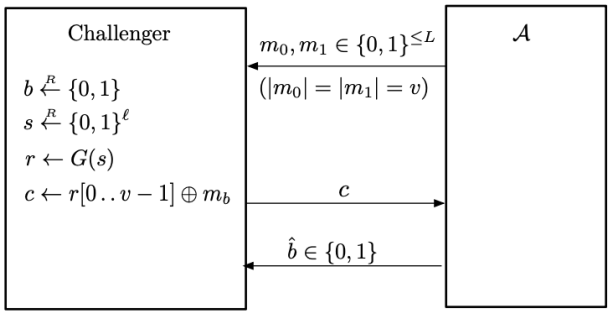


图 3.2: 定理 3.1 的证明中的游戏 0

让我们称这个游戏为**游戏 0**。挑战者在这个游戏中的逻辑可以表示如下：

当收到来自 $\mathcal{A}$ 的 $m_0, m_1 \in \{0,1\}^v$，其中 $v \leq L$ 时：
计算 $b \xleftarrow{R} \{0,1\}$
计算 $s \xleftarrow{R} \{0,1\}^l$，$r \leftarrow G(s)$
计算 $c \leftarrow r[0 \ldots v-1] \oplus m_b$
将 c 发送给 $\mathcal{A}$。

图 3.2 展示了游戏 0 的流程。

记 W_0 为游戏 0 中 $\hat{b} = b$ 的事件。根据定义，我们有：

$$\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}] = |\Pr[W_0] - 1/2| \tag{3.4}$$

接下来，我们修改游戏 0 中挑战者的逻辑，得到一个新的游戏，我们称之为**游戏 1**。游戏 1 与游戏 0 基本相同，只是挑战者现在会用一个真随机字符串代替伪随机字符串。游戏 1 中挑战者的逻辑如下：

当收到来自 $\mathcal{A}$ 的 $m_0, m_1 \in \{0,1\}^v$，其中 $v \leq L$ 时：
计算 $b \xleftarrow{R} \{0,1\}$
计算 $r \xleftarrow{R} \{0,1\}^L$
计算 $c \leftarrow r[0 \ldots v-1] \oplus m_b$
将 c 发送给 $\mathcal{A}$。

与之前一样，$\mathcal{A}$ 在该游戏结束时输出一个比特 $\hat{b}$。我们用灰色强调了和游戏 0 之间的差别。图 3.3 展示了游戏 1 的流程。

记 W_1 为游戏 1 中 $\hat{b} = b$ 的事件。我们声称：

$$\Pr[W_1] = 1/2 \tag{3.5}$$

这是因为在游戏 1 中，对手事实上攻击的是变长一次性密码本。特别地，很容易看到，对手的输出 $\hat{b}$ 和挑战者的隐藏比特 b 是相互独立的。

最后，我们展示如何构建一个有效 PRG 对手 $\mathcal{B}$，它将 $\mathcal{A}$ 作为一个子程序，使得：

$$|\Pr[W_0] - \Pr[W_1]| = \mathrm{PRGadv}[\mathcal{B}, G] \tag{3.6}$$

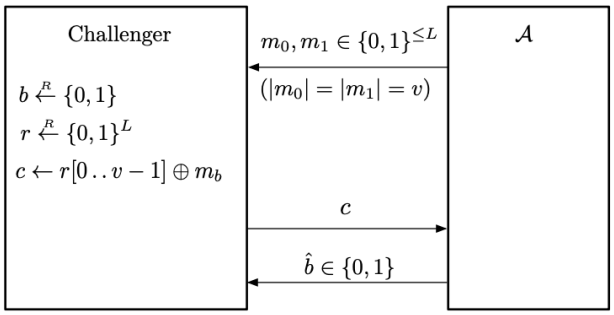


图 3.3: 定理 3.1 的证明中的游戏 1

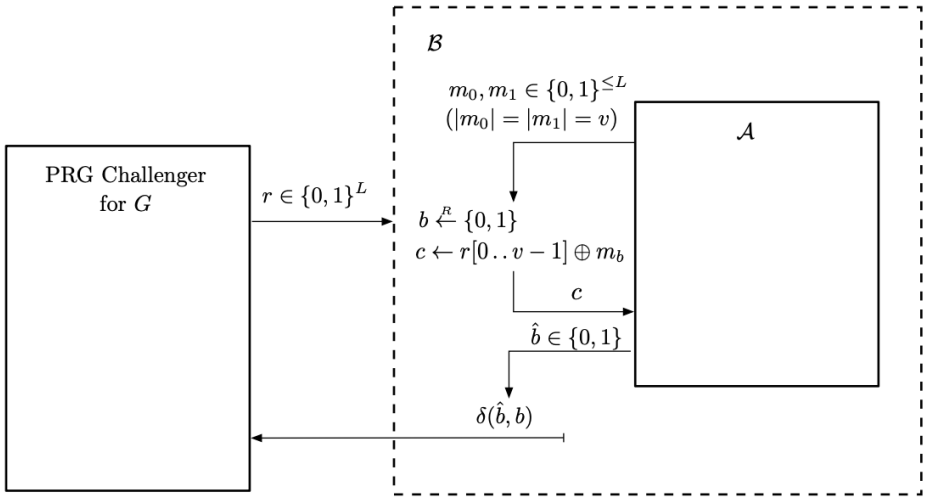


图 3.4: 定理 3.1 的证明中的 PRG 对手 $\mathcal{B}$

成立。这实际上非常简单。我们的新对手 $\mathcal{B}$ 的逻辑如图 3.4 所示。这里，δ 的定义如下：

$$\delta(x,y) = \begin{cases} 1, & x = y \\ 0, & x \neq y \end{cases} \tag{3.7}$$

另外，标有 “PRG 挑战者” 的方框扮演攻击游戏 3.1 中 G 的挑战者的角色。

换句话说，对手 $\mathcal{B}$ 是一个（如攻击游戏 3.1 中那样）旨在攻击 G 的 PRG 对手，它从它的 PRG 挑战者那里收到 $r \in \{0,1\}^L$，然后扮演 $\mathcal{A}$ 的挑战者角色，如下所示：

当收到来自 $\mathcal{A}$ 的 $m_0, m_1 \in \{0,1\}^v$，其中 $v \leq L$ 时：

计算 $b \xleftarrow{\mathrm{R}} \{0,1\}$

计算 $c \leftarrow r[0 \ldots v-1] \oplus m_b$

将 c 发送给 $\mathcal{A}$。

最后，当 $\mathcal{A}$ 输出一个比特 $\hat{b}$ 时，$\mathcal{B}$ 输出比特 $\delta(\hat{b}, b)$。

令 p_0 为 PRG 挑战者在运行攻击游戏 3.1 的实验 0 时，$\mathcal{B}$ 输出 1 的概率，令 p_1 为 PRG 挑战者运行攻击游戏 3.1 的实验 1 时，对手 $\mathcal{B}$ 输出 1 的概率。根据定义，$\mathsf{PRGadv}[\mathcal{A}, G] = |p_1 - p_0|$。此外，如果 PRG 挑战者正在运行实验 0，那么对手 $\mathcal{A}$ 本质上就是在进行我们这里的游戏 0，所以有 $p_0 = \Pr[W_0]$；而如果

PRG 挑战者正在进行实验 1，那么对手 $\mathcal{A}$ 本质上就是在进行我们这里的游戏 1，所以有 $p_1 = \Pr[W_1]$。于是我们立即就能得到式 3.6。

结合式 3.4，3.5 和 3.6，我们就可得到式 3.3。 □

在上述定理中，我们将 $\mathcal{E}$ 的安全性归约到 G 的安全性上，并表明如果 $\mathcal{A}$ 是一个攻击 $\mathcal{E}$ 的有效语义安全对手，那么存在一个攻击 G 的有效 PRG 对手 $\mathcal{B}$，使得：

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] \leq 2 \cdot \mathsf{PRGadv}[\mathcal{B}, G]$$

（实际上，我们还证明了等号是成立的，但这并不重要。）在证明中，我们认为如果 G 是安全的，那么 $\mathsf{PRGadv}[\mathcal{B}, G]$ 的值可忽略不计。因此，根据上述不等式，我们得出结论，$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ 也是可忽略不计的。由于这对所有的有效对手 $\mathcal{A}$ 都成立，于是我们可以得出结论，$\mathcal{E}$ 是语义安全的。

类似于定理 2.7 之后的讨论，另一种结构化的证明方法是反证法：事实上，如果我们假设 $\mathcal{E}$ 是不安全的，那么一定有一个有效对手 $\mathcal{A}$ 能使得 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ 的值不可忽略不计。而安全归约（以及上述不等式）赋予我们一个有效对手 $\mathcal{B}$，使得 $\mathsf{PRGadv}[\mathcal{B}, G]$ 的值也不可忽略不计。也就是说，如果我们能破解 $\mathcal{E}$，我们也就能破解 G。虽然这在逻辑上是等价的，但这样的证明有一种不同的“感觉”：我们从破解 $\mathcal{E}$ 的对手 $\mathcal{A}$ 开始，并说明如何使用 $\mathcal{A}$ 来构造一个能破解 G 的新对手 $\mathcal{B}$。

读者应该注意到，上述定理的证明与我们在 2.2.4 小节中对网络电子轮盘赌的分析遵循基本相同的模式。在这两种情况下，我们都从一个攻击游戏开始（图 2.2 和图 3.2），我们对其进行修改，从而得到一个新的攻击游戏（图 2.3 和图 3.3）；而在这个新的攻击游戏中，计算对手的优势非常容易。此外，我们还使用了一个适当的安全假设，以证明对手在原始游戏和修改后的游戏中的优势差异是可忽略不计的。这是通过引入一个新的对手（图 2.4 和图 3.4）来实现的，该对手攻击底层的密码学原语（密码或 PRG），其优势等于这一差异。假设底层原语是安全的，那么这个差异一定是可忽略不计的；或者，我们可以反过来说：如果这个差异是不可忽略不计的，那么新对手就会“破坏”底层密码学原语。

这是一个将在本文中反复出现的模式。因此读者应该仔细研究这种两种分析方法，以确保能够完全理解其中的原理。

3.3 流密码的局限性：对一次性密码本的攻击

尽管流密码是语义安全的，但它非常脆弱，如果使用不当就会完全丧失安全性。

3.3.1 两次密码本是不安全的

一个流密码可以很好地加密从 Alice 到 Bob 的单一消息。然而，Alice 可能希望向 Bob 发送多条消息。简单起见，假设 Alice 希望对两条消息 m_1 和 m_2 进行加密。她天真的解决方案是使用相同的流密码密钥 s 对这两条消息进行加密，即：

$$c_1 \leftarrow m_1 \oplus G(s), \quad c_2 \leftarrow m_2 \oplus G(s) \tag{3.8}$$

稍微思考一下就会发现，这种结构根本就是不安全的。对手如果能够截获 c_1 和 c_2，就能计算出：

$$\Delta := c_1 \oplus c_2 = \big(m_1 \oplus G(s)\big) \oplus \big(m_2 \oplus G(s)\big) = m_1 \oplus m_2$$

并得到 m_1 和 m_2 的异或。考虑到英文文本中往往会包含足够多的冗余，因此给定 $\Delta = m_1 \oplus m_2$，对手可以轻易地恢复出 m_1 和 m_2。因此，只要能够获得足够长的密文，式 3.8 中的构造就会泄露明文。

式 3.8 中的构造被戏称为**两次密码本 (two-time pad)**。我们上面论证了两次密码本是完全不安全的，特别是，**一个流密码的密钥不应该被用来加密多条消息**。在这本书中，我们将看到很多例子，在这些例子中，使用一个一次性密码本就足够满足要求。例如像在 5.4.1 小节中，我们为每个消息选择一个新的随机密钥。然而，在单一密钥可能会被多次使用的环境中，绝不能直接使用流密码。我们会在第五章构建可以多次使用的密码。

不正确地重复使用流密码密钥是部署系统中的一个常见错误。例如，一个叫做 PPTP 的协议使 A 和 B 两方能够互相发送加密的消息。微软在 Windows NT 中使用了一种叫做 RC4 的流密码来实现 PPTP 协议。在微软最初的实现中，从 B 到 A 的加密密钥与从 A 到 B 的加密密钥是完全相同的。因此，通过窃听两个相反方向的加密信息，攻击者就可以恢复这两条消息的明文。

另一个关于两次密码本的有趣故事来自 Klehr 的转述，他非常详细地描述了二战期间在美国的俄罗斯间谍是如何使用一次性密码本把消息送回莫斯科的。正如 Klehr 所解释的，该系统由一个致命的缺陷：

> 在二战期间，苏联无法生产足够多的一次性密码本... 来满足庞大的密码需求...。因此，他们把一些一次性密码本用了两次，还认为这不会影响到他们的系统。二战期间，美国的反间谍部门收集了所有进出的国际电报。从 1946 年开始，在英国人的帮助下，他们开始大力破解苏联的信息，由于... 苏联把一些一次性密码本当作两次密码本使用的错误，在接下来的 25 年里，它们破获了大约 2900 条消息，包含 1941 年至 1946 年（苏联在此期间改用另外一套系统）之间发送的 5000 多页共计数十万条消息。

解密工作的代号为维诺那 (Venona) 计划。维诺那计划因揭发了朱利叶斯 · 罗森堡和艾瑟尔 · 罗森堡夫妇而声名鹊起，他们参与了苏联间谍集团的证据被维诺那计划完整地揭露出来。从 1995 年开始，所有 3000 多份由维诺那计划解密的消息都被公开了。

3.3.2 一次性密码本是易被控制的

尽管语义安全性能够确保对手无法读取明文，但它没有提供完整性的保证。在使用流密码时，对手可以改变密文，而这种修改永远不会被解密者发现。更糟糕的是，我们将表明，通过改变密文，攻击者甚至可以控制解密后的明文。

假设攻击者截获了一条密文 $c := E(s, m) = m \oplus G(s)$。攻击者将 c 改为 $c' := c \oplus \Delta$，其中的 Δ 是攻击者选取的某个值。因此，解密者收到修改后的消息为：

$$D(s, c') = c' \oplus G(s) = (c \oplus \Delta) \oplus G(s) = m \oplus \Delta$$

因此，即使攻击者不知道 m 或 s，他也能将解密后的消息变成 $m \oplus \Delta$，而 Δ 是由攻击者选择的。因此我们称流密码是**易被控制的 (malleable)**，因为攻击者可以对明文造成可预测的变化。我们将在第九章中构建能够同时满足机密性和完整性的密码。

一个简单的例子是，易被控制的特性可以帮助攻击者攻击加密文件系统。为了具体说明这一问题，假设 Bob 是一位教授，Alice 和 Molly 是两个学生。Bob 的学生通过电子邮件提交他们的作业，然后 Bob 将这些电子邮件存储在一个用流密码加密的磁盘上。一封电子邮件总是以一个标准头开始。简化一下，我们可以假设一封来自 Alice 的电子邮件总是以 `From:Alice` 为开头。

现在，假设 Molly 能够进入 Bob 的磁盘，并找到了 Alice 发出的包含她的作业的电子邮件的密文。于是 Molly 可以窃取 Alice 的作业，方法如下。她只需将适当的五个字符的字符串异或到密码文本的第 6 至 10 位，以便将邮件首部 `From:Alice` 改为 `From:Molly`。Molly 只需要对密文进行操作，并且不需要知道 Bob 的私钥。Bob 永远不会知道文件被修改过，当他给 Alice 的作业打分时，还以为这份作业是 Molly 交的，因此 Molly 就窃取了 Alice 的成果。

当然，为了使这种攻击有效，Molly 必须能以某种方式在 Bob 的加密磁盘上找到 Alice 的电子邮件。然而，在一些加密文件系统的实现中，文件的元数据（如文件名、修改时间等）是不加密的。有了这些元数据，Molly 就可以直接找到 Alice 的加密邮件并进行攻击了。

3.4 组合 PRG

在本节中，我们将讨论两种构造，以允许我们基于旧的 PRG 建立新的 PRG。这些构造允许我们扩展原始 PRG 的输出空间的大小，同时保留其安全性。更重要的是，在本节中我们将介绍一种非常重要的论证技术，称为**混合论证 (hybrid argument)**，该技术在现代密码学中的应用非常广泛。

3.4.1 一种并行构造

令 G 是一个定义在 $(\mathcal{S}, \mathcal{R})$ 上的 PRG。假设在某些应用中，我们想多次使用 G。我们希望 G 的所有输出与 R 上的随机元素是计算上无法区分的。如果 G 是一个安全的 PRG，并且种子是独立产生的，那么这样的 G 就能满足这个要求。

我们可以将 G 的多个应用建模为一个新的 PRG G'。也就是说，我们构建一个新的 PRG G'，它将 G 应用于 n 个种子，并将输出连接起来。因此，G' 定义在 $(\mathcal{S}^n, \mathcal{R}^n)$ 上，对于 $s_1, \ldots, d_n \in \mathcal{S}$，我们有：

$$G'(s_1, \ldots, s_n) := \big(G(s_1), \ldots, G(s_n)\big)$$

我们称 G' 为 G 的 n **次并行组合 (n-wise parallel composition)**，n 称为**重复参数 (repetition parameter)**，并且，我们要求 n 是一个多项式边界的值。

定理 3.2. *如果 G 是一个安全的 PRG，那么 G 的 n 次并行组合 G' 也是一个安全的 PRG。*

> *特别地，对于每一个按照攻击游戏 3.1 攻击 G' 的 PRG 对手 $\mathcal{A}$，都存在一个按照攻击游戏 3.1 攻击 G 的 PRG 对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，使得：*
>
> $$\mathrm{PRGadv}[\mathcal{A}, G'] = n \cdot \mathrm{PRGadv}[\mathcal{B}, G]$$

作为热身，我们首先在 $n = 2$ 的特殊情况下证明这个定理。令 $\mathcal{A}$ 是一个有效 PRG 对手，在攻击游戏 3.1 中攻击 G' 时有优势 ϵ。我们想证明，如果 G 是一个安全 PRG，则 ϵ 可以忽略不计。为此，我们把**游戏 0** 定义为攻击游戏 3.1 中的实验 0。这个游戏中的挑战者的工作方式如下：

> 计算 $s_1 \xleftarrow{R} \mathcal{S}$，$r_1 \leftarrow G(s_1)$
> 计算 $s_2 \xleftarrow{R} \mathcal{S}$，$r_2 \leftarrow G(s_2)$
> 将 (r_1, r_2) 发送给 $\mathcal{A}$。

令 p_0 为 $\mathcal{A}$ 在该游戏中输出 1 的概率。

下面，我们定义**游戏 1**，它在 $\mathcal{A}$ 和一个挑战者之间进行，其工作方式如下：

计算 $r_1 \xleftarrow{R} \mathcal{R}$
计算 $s_2 \xleftarrow{R} \mathcal{S}$，$r_2 \leftarrow G(s_2)$
将 (r_1, r_2) 发送给 $\mathcal{A}$。

请注意，游戏 1 既不对应于攻击游戏 3.1 中的实验 0，也不对应于实验 1；相反，它是一个“混合”实验，对应于实验 0 和实验 1 之间的某个状态。我们所做的就是在游戏 0 中用一个真随机值取代伪随机值 r_1（见上面的高亮行）。直观地说，在假设 G 是一个安全 PRG 的情况下，对手 $\mathcal{A}$ 应该注意不到这种差异。为了更精确地论证这一点，令 p_1 是 $\mathcal{A}$ 在游戏 1 中输出 1 的概率。

令 $\delta_1 := |p_1 - p_0|$。我们声称，假设 G 是一个安全的 PRG，则 δ_1 是可忽略不计的。事实上，我们很容易构建一个有效 PRG 对手 $\mathcal{B}_1$，他在攻击游戏 3.1 中攻击 G 的优势正好等于 δ_1。对手 $\mathcal{B}_1$ 的工作方式如下：

当收到来自挑战者的 $r \in \mathcal{R}$，$\mathcal{B}_1$ 扮演 $\mathcal{A}$ 的挑战者的角色：
令 $r_1 \leftarrow r$
计算 $s_2 \xleftarrow{R} \mathcal{S}$，$r_2 \leftarrow G(s_2)$
将 (r_1, r_2) 发送给 $\mathcal{A}$。
最后，$\mathcal{B}_1$ 输出 $\mathcal{A}$ 所输出的任何东西。

请注意，当 $\mathcal{B}_1$ 在其攻击游戏的实验 0 中时，它完全模仿挑战者在游戏 0 中的行为，而在实验 1 中，它完全模仿挑战者在游戏 1 中的行为。因此，p_0 等于 $\mathcal{B}_1$ 在攻击游戏 3.1 的实验 0 中输出 1 的概率，而 p_1 等于 $\mathcal{B}_1$ 在攻击游戏 3.1 的实验 1 中输出 1 的概率。因此，$\mathcal{B}_1$ 在攻击 G 时的优势恰好是 $|p_1 - p_0|$，正如我们上面声称的。

接下来，我们定义**游戏 2**，它在 $\mathcal{A}$ 和一个挑战者之间进行，其工作方式如下：

计算 $r_1 \xleftarrow{R} \mathcal{R}$
计算 $r_2 \xleftarrow{R} \mathcal{R}$
将 (r_1, r_2) 发送给 $\mathcal{A}$。

我们所做的就是把游戏 1 中的伪随机值 r_2 替换成一个真正的随机值（见上面的高亮行）。令 p_2 为 $\mathcal{A}$ 在游戏 2 中输出 1 的概率。请注意，游戏 2 对应于攻击游戏 3.1 的实验 1，因此 p_2 等于攻击游戏 3.1 的实验 1 中 $\mathcal{A}$ 相对于 PRG G' 输出 1 的概率。

令 $\delta_2 := |p_2 - p_1|$。通过与上面类似的论证，很容易看到当 G 是一个安全的 PRG 时，δ_2 是可忽略不计的。事实上，我们很容易构建一个有效 PRG 对手 $\mathcal{B}_2$，他在攻击游戏 3.1 中攻击 G 的优势正好等于 δ_2。对手 $\mathcal{B}_2$ 的工作方式如下：

当收到来自挑战者的 $r \in \mathcal{R}$，$\mathcal{B}_2$ 扮演 $\mathcal{A}$ 的挑战者的角色：
计算 $r_1 \xleftarrow{R} \mathcal{R}$
令 $r_2 \leftarrow r$
将 (r_1, r_2) 发送给 $\mathcal{A}$。
最后，$\mathcal{B}_2$ 输出 $\mathcal{A}$ 所输出的任何东西。

p_1 显然就等于 $\mathcal{B}_2$ 在攻击游戏 3.1 的实验 0 中输出 1 的概率，而 p_2 等于 $\mathcal{B}_2$ 在攻击游戏 3.1 的实验 1 中输出 1 的概率。

回顾一下，我们有 $\epsilon = \mathrm{PRGadv}[\mathcal{A}, G']$，那么从上面的讨论中，我们有：

$$\epsilon = |p_2 - p_0| = |p_2 - p_1 + p_1 - p_0| \le |p_1 - p_0| + |p_2 - p_1| = \delta_1 + \delta_2$$

由于 δ_1 和 δ_2 都可忽略不计，那么 ϵ 也可忽略不计（见事实 2.6）。

这就完成了 G' 在 $n = 2$ 情况下的安全证明。在给出一般情况下的证明之前，我们先给出 $n = 2$ 情况下的另一种证明方式。我们的第一个证明采用了构建两个对手 $\mathcal{B}_1$ 和 $\mathcal{B}_2$ 的方法，而第二个证明会把这两个对手合并为一个单一的如同攻击游戏 3.1 中攻击 G 的 PRG 对手 $\mathcal{B}$，其工作方式如下：

当收到来自挑战者的 $r \in \mathcal{R}$ 时，$\mathcal{B}$ 进行以下操作：

随机选取 $\omega \in \{1, 2\}$

将 r 发送给 $\mathcal{B}_\omega$。

最后，$\mathcal{B}$ 输出 $\mathcal{B}_\omega$ 所输出的任何东西。

记 W_0 为 $\mathcal{B}$ 在攻击游戏 3.1 的实验 0 中输出 1 的事件，W_1 为 $\mathcal{B}$ 在攻击游戏 3.1 的实验 1 中输出 1 的事件。以 $\omega = 1$ 和 $\omega = 2$ 为条件，我们有：

$$\begin{aligned}\Pr[W_0] &= \Pr[W_0 \,|\, \omega = 1] \Pr[\omega = 1] + \Pr[W_0 \,|\, \omega = 2] \Pr[\omega = 2] \\ &= \frac{1}{2}\Big(\Pr[W_0 \,|\, \omega = 1] + \Pr[W_0 \,|\, \omega = 2] \Big) \\ &= \frac{1}{2}(p_0 + p_1)\end{aligned}$$

类似地，我们也有：

$$\begin{aligned}\Pr[W_1] &= \Pr[W_1 \,|\, \omega = 1] \Pr[\omega = 1] + \Pr[W_1 \,|\, \omega = 2] \Pr[\omega = 2] \\ &= \frac{1}{2}\Big(\Pr[W_1 \,|\, \omega = 1] + \Pr[W_1 \,|\, \omega = 2] \Big) \\ &= \frac{1}{2}(p_1 + p_2)\end{aligned}$$

因此，如果 δ 是 $\mathcal{B}$ 在攻击游戏 3.1 中相对于 G 的优势，我们就有：

$$\delta = \big|\Pr[W_1] - \Pr[W_0]\big| = \left|\frac{1}{2}(p_1 + p_2) - \frac{1}{2}(p_0 + p_1)\right| = \frac{1}{2}|p_2 - p_0| = \epsilon/2$$

即 $\epsilon = 2\delta$。由于 δ 可忽略不计，ϵ 也可以忽略不计（见事实 2.6）。

下面，我们正式介绍当 n 为一个普通的，多项式边界的值时，定理 3.2 的证明。

证明思路. 我们可以尝试将上述第一种策略从 $n = 2$ 的情况扩展到任意的 n 的情况。也就是说，我们可以构建一个由 $n + 1$ 个游戏构成的游戏序列。挑战者在开始时产生一个伪随机序列 $G(s_1), \ldots, G(s_n)$，每次用 $\mathcal{R}$ 上的真随机元素替换其中的一个元素，最终得到一个 $\mathcal{R}$ 上的真随机序列 $(r_1, \ldots, r_n)$。直觉上，对手应该注意不到这些替换中的哪怕任何一个，因为 G 是一个安全的 PRG；然而，正式证明这一点需要构建 n 个不同的对手，每个对手以稍微不同的方式去攻击 G。事实上，当 n 不是一个绝对的常数，而单纯只是一个多项式边界的值时，这会导致一些恼人的技术困难；而扩展上面概述的第二种策略则要简单得多，因为这只需要构建一个对手，就能够 “一击即中” 地攻击 G。 □

证明. 令 $\mathcal{A}$ 是一个有效 PRG 对手，他对 G' 进行攻击游戏 3.1 中的攻击。我们首先引入一个大小为 $n + 1$ 的**混合游戏 (hybrid games)** 序列，称为混合 0，混合 1，...，混合 n。对于 $j = 0, 1, \ldots, n$，混

Hybrid 0:	$G(s_1)$	$G(s_2)$	$G(s_3)$	$\cdots$	$G(s_n)$
Hybrid 1:	r_1	$G(s_2)$	$G(s_3)$	$\cdots$	$G(s_n)$
Hybrid 2:	r_1	r_2	$G(s_3)$	$\cdots$	$G(s_n)$
$\vdots$					
Hybrid $n-1$:	r_1	r_2	r_3	$\cdots$	$G(s_n)$
Hybrid n:	r_1	r_2	r_3	$\cdots$	r_n

图 3.5: 挑战者在混合 $0, 1, \ldots, n$ 中准备的数值。每个 r_i 都是 $\mathcal{R}$ 上的一个随机元素，每个 s_i 都是 $\mathcal{S}$ 上的一个随机元素。

合 j 是一个 $\mathcal{A}$ 和挑战者之间的游戏。挑战者会准备一个 n 个值的元组，其中的前 j 个元素是真随机值，其余的 $n-j$ 个元素是由 G 输出的伪随机值；也就是说，挑战者的工作方式如下：

$r_1 \xleftarrow{R} \mathcal{R}$
$\vdots$
$r_j \xleftarrow{R} \mathcal{R}$
$s_{j+1} \xleftarrow{R} \mathcal{S},\ r_{j+1} \leftarrow G(s_{j+1})$
$\vdots$
$s_n \xleftarrow{R} \mathcal{S},\ r_n \leftarrow G(s_{j+1})$
将 $(r_1, \ldots, r_n)$ 发送给 $\mathcal{A}$。

像之前一样，$\mathcal{A}$ 在游戏结束时输出 0 或 1。图 3.5 展示了挑战者在这 $n+1$ 个游戏中的每个游戏中准备的数值。令 p_j 为 $\mathcal{A}$ 在混合 j 中输出 1 的概率。注意，p_0 等于 $\mathcal{A}$ 在攻击游戏 3.1 的实验 0 中输出 1 的概率，而 p_n 则等于 $\mathcal{A}$ 在实验 1 中输出 1 的概率。因此，我们有：

$$\mathrm{PRGadv}[\mathcal{A}, G'] = |p_n - p_0| \tag{3.9}$$

接下来我们定义一个 PRG 对手 $\mathcal{B}$，它对 G 进行攻击游戏 3.1 中的攻击，其工作方式如下：

当收到来自挑战者的 $r \in \mathcal{R}$，$\mathcal{B}$ 扮演 $\mathcal{A}$ 的挑战者的角色：
$\omega \xleftarrow{R} \{1, \ldots, n\}$
$r_1 \xleftarrow{R} \mathcal{R}$
$\vdots$
$r_{\omega-1} \xleftarrow{R} \mathcal{R}$
$r_\omega \leftarrow r$
$s_{\omega+1} \xleftarrow{R} \mathcal{S},\ r_{\omega+1} \leftarrow G(s_{\omega+1})$
$\vdots$
$s_n \xleftarrow{R} \mathcal{S},\ r_n \leftarrow G(s_{j+1})$
将 $(r_1, \ldots, r_n)$ 发送给 $\mathcal{A}$。
最后，$\mathcal{B}$ 输出 $\mathcal{A}$ 所输出的任何东西。

令 W_0 是 $\mathcal{B}$ 在攻击游戏 3.1 的实验 0 中输出 1 的事件，W_1 是 $\mathcal{B}$ 在攻击游戏 3.1 的实验 1 中输出 1 的事件。一个关键的观察是：

对于每个固定的 $j = 1, \ldots, n$，以 $\omega = j$ 为条件，$\mathcal{B}$ 的攻击游戏的实验 0 就相当于混合 $j-1$，而 $\mathcal{B}$ 的攻击游戏的实验 1 则相当于混合 j。

因此：

$$\Pr[W_0 \,|\, \omega = j] = p_{j-1}, \quad \Pr[W_1 \,|\, \omega = j] = p_j$$

所以，我们有：

$$\Pr[W_0] = \sum_{j=1}^{n} \Pr[W_0 \,|\, \omega = j] \Pr[\omega = j] = \frac{1}{n} \sum_{j=1}^{n} \Pr[W_0 \,|\, \omega = j] = \frac{1}{n} \sum_{j=1}^{n} p_{j-1}$$

类似地：

$$\Pr[W_1] = \sum_{j=1}^{n} \Pr[W_1 \,|\, \omega = j] \Pr[\omega = j] = \frac{1}{n} \sum_{j=1}^{n} \Pr[W_1 \,|\, \omega = j] = \frac{1}{n} \sum_{j=1}^{n} p_j$$

最终，我们有：

$$\begin{aligned} \mathsf{PRGadv}[\mathcal{B}, G] &= |\Pr[W_1] - \Pr[W_0]| \\ &= \left| \frac{1}{n} \sum_{j=1}^{n} p_j - \frac{1}{n} \sum_{j=1}^{n} p_{j-1} \right| \\ &= \frac{1}{n} |p_n - p_0| \end{aligned}$$

将其与式 3.9 相结合，我们可以得到：

$$\mathsf{PRGadv}[\mathcal{A}, G'] = n \cdot \mathsf{PRGadv}[\mathcal{B}, G]$$

由于我们假设 G 是一个安全 PRG，因此 $\mathsf{PRGadv}[\mathcal{B}, G]$ 可以忽略不计，而由于 n 是多项式边界的，$\mathsf{PRGadv}[\mathcal{B}, G']$ 也是可忽略不计的（见事实 2.6）。这就证明了该定理。 □

定理 3.2 表明，PRG 的安全性随着我们使用它的次数增多而最多呈线性下降。有人可能会问，这个约束是否是严格的？也即，安全性是否真的会随着使用次数的增加而线性下降？答案其实是肯定的（见练习 3.14）。

3.4.2 一种串行构造：Blum-Micali 方法

我们下面介绍一种由 Blum 和 Micali 发明的串行构造，它使用一个只稍做延展的 PRG，建立一个可以伸展到任意长度的 PRG。

令 G 是一个定义在 $(\mathcal{S}, \mathcal{R} \times \mathcal{S})$ 上的 PRG，其中 $\mathcal{S}$ 和 $\mathcal{R}$ 是有限集。对于每个多项式边界的值 $n \geq 1$，我们可以构造一个定义在 $(\mathcal{S}, \mathcal{R}^n \times \mathcal{S})$ 上的新的 PRG G'。对于 $s \in \mathcal{S}$，我们令：

$G'(s) :=$
　$s_0 \leftarrow s$
　对于 $i \leftarrow 1$ 到 n：
　　$(r_i, s_i) \leftarrow G(s_{i-1})$
　输出 $(r_1, \ldots, r_n, s_n)$

我们称 G' 为 G 的 n **次串行组合 (n-wise sequential composition)**。图 3.6 是 $n = 3$ 时 G' 的示意图。

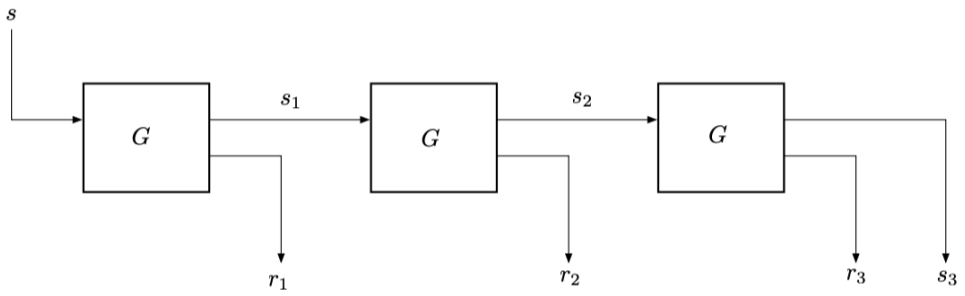


图 3.6: $n = 3$ 时的串行构造

我们将在下面的定理 3.3 中证明，如果 G 是一个安全 PRG，那么 G' 也是一个安全 PRG。作为这个构造的一个特例，假设 G 是一个定义在 $(\{0,1\}^{\ell}, \{0,1\}^{t+\ell})$ 上的 PRG，其中 ℓ 和 t 是正整数；也就是说，G 把 ℓ 比特字符串拉伸为 $t+\ell$ 比特字符串。我们可以很自然地把 G 的输出空间看作是 $\{0,1\}^t \times \{0,1\}^{\ell}$。应用上述构造，并把输出解释为比特串，我们就能得到一个 PRG G'，它能把 ℓ 位比特串拉伸为 $nt + \ell$ 位比特串。

定理 3.3. *如果 G 是一个安全 PRG，那么 G 的 n 次串行组合 G' 也是一个安全 PRG。*

> *特别地，对于每个对 G' 进行攻击游戏 3.1 中的攻击的 PRG 对手 $\mathcal{A}$，都存在一个对 G 进行攻击游戏 3.1 中的攻击的 PRG 对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足：*
>
> $$\mathsf{PRGadv}[\mathcal{A}, G'] = n \cdot \mathsf{PRGadv}[\mathcal{B}, G]$$

证明思路. 该定理的证明是一个混合论证，其思路与定理 3.2 的证明非常相似。该证明背后的直觉如下所述。考虑一个 PRG 对手 $\mathcal{A}$，他在攻击游戏 3.1 的实验 0 中收到 $(r_1, \ldots, r_n, s_n)$。由于 $s = s_0$ 是随机的，而且 G 是一个安全 PRG，我们可以用 $\mathcal{R} \times \mathcal{S}$ 中的一个完全随机的元素来代替 (r_1, s_1)，且 $\mathcal{A}$ 在这个新的混合游戏中输出 1 的概率应该只会发生可忽略不计的变化。现在，由于 s_1 是随机的（同样是因为 G 是一个安全 PRG），我们就可以用 $\mathcal{R} \times \mathcal{S}$ 上的一个完全随机的元素来代替 (r_2, s_2)，而 $\mathcal{A}$ 在这第二个混合游戏中输出 1 的概率应该也只会发生可忽略不计的变化。继续这样下去，我们可以用 $\mathcal{R} \times \mathcal{S}$ 中的随机元素逐步替换 (r_3, s_3) 到 (r_n, s_n)，在做出这些改变之后，$\mathcal{A}$ 输出 1 的概率应该都只会发生可忽略不计的变化（假设 n 是多项式边界的）。然而，在这一点上，$\mathcal{A}$ 输出 1 的概率与他在攻击游戏 3.1 的实验 1 中输出 1 的概率相同，因此，这个概率与 $\mathcal{A}$ 在攻击游戏 3.1 的实验 0 中输出 1 的概率接近得可以忽略不计。

这就是我们的想法；然而，就像在定理 3.2 的证明中一样，由于技术原因，我们设计了一个攻击 G 的 PRG 对手。 □

证明. 令 $\mathcal{A}$ 是一个对 G' 进行攻击游戏 3.1 中的攻击的 PRG 对手。我们首先引入一个大小为 $n+1$ 的混合游戏序列，称为混合 0，混合 1，...，混合 n。对于 $j = 0, 1, \ldots, n$，我们定义混合 j 为 $\mathcal{A}$ 和以下挑战者之间的游戏。挑战者的工作方式如下：

$$r_1 \xleftarrow{R} \mathcal{R}$$
$$\vdots$$
$$r_j \xleftarrow{R} \mathcal{R}$$

$s_j \xleftarrow{R} \mathcal{S}$
$(r_{j+1}, s_{j+1}) \leftarrow G(s_j)$
$\vdots$
$(r_n, s_n) \leftarrow G(s_{n-1})$
将 $(r_1, \ldots, r_n, s_n)$ 发送给 $\mathcal{A}$。

像之前一样，$\mathcal{A}$ 在游戏结束时输出 0 或 1。图 3.7 展示了挑战者在 $n = 3$ 的情况下的工作流程。请注意，p_0 也等于 $\mathcal{A}$ 在攻击游戏 3.1 的实验 0 中输出 1 的概率，而 p_n 则等于 $\mathcal{A}$ 在攻击游戏 3.1 的实验 1 中输出 1 的概率。因此，我们有：

$$\mathsf{PRGadv}[\mathcal{A}, G'] = |p_n - p_0| \tag{3.10}$$

接下来我们定义一个 PRG 对手 $\mathcal{B}$，它对 G 进行攻击游戏 3.1 中的攻击，其工作方式如下：

当收到来自挑战者的 $(r, s) \in \mathcal{R} \times \mathcal{S}$，$\mathcal{B}$ 扮演 $\mathcal{A}$ 的挑战者的角色：
$\omega \xleftarrow{R} \{1, \ldots, n\}$
$r_1 \xleftarrow{R} \mathcal{R}, \ldots, r_{\omega-1} \xleftarrow{R} \mathcal{R}$
$(r_\omega, s_\omega) \leftarrow (r, s)$
$(r_{\omega+1}, s_{\omega+1}) \leftarrow G(s_\omega), \ldots, (r_n, s_n) \leftarrow G(s_{n-1})$
将 $(r_1, \ldots, r_n, s_n)$ 发送给 $\mathcal{A}$。
最后，$\mathcal{B}$ 输出 $\mathcal{A}$ 所输出的任何东西。

令 W_0 是 $\mathcal{B}$ 在攻击游戏 3.1 的实验 0 中输出 1 的事件，W_1 是 $\mathcal{B}$ 在攻击游戏 3.1 的实验 1 中输出 1 的事件。一个关键的观察是：

对于每个固定的 $j = 1, \ldots, n$，以 $\omega = j$ 为条件，$\mathcal{B}$ 的攻击游戏的实验 0 等同于混合 $j - 1$，而 $\mathcal{B}$ 的攻击游戏的实验 1 则等同于混合 j。

因此：

$$\Pr[W_0 \,|\, \omega = j] = p_{j-1}, \quad \Pr[W_1 \,|\, \omega = j] = p_j$$

接下来的证明只是简单的计算，与定理 3.2 的证明的最后一段相同，不再赘述。 □

评价 PRG 的一个标准是它的**扩展率 (expansion rate)**：一个将 n 比特种子扩展到 m 比特输出的 PRG 的扩展率为 m/n。更一般地说，如果种子空间是 $\mathcal{S}$，输出空间是 R，我们将扩展率定义为 $\log|\mathrm{R}|/\log|\mathcal{S}|$。串行组合构造比并行组合构造提供了更好的扩展率，但是它有一个缺点，那就是它不能被并行化。事实上，存在一种构造可以兼顾两种优点：大的扩展率和高度可并行的结构，见 4.4.4 小节。

3.4.3 数学细节

在定理 3.2 和 3.3 的证明中，有一些微妙的地方值得讨论。

首先，在这两个构造中，底层的 PRG G 可能有系统参数。也就是说，可能存在一个概率性算法，它将安全参数 λ 作为输入，并输出一个系统参数 Λ。回顾一下，系统参数是能够完全将构造实例化的公共数据（在这个场景中，它可能定义了种子空间和输出空间）。对于并行和串行构造，我们可以对 G 的所有 n 个实例使用相同的系统参数；事实上，对于串行构造来说，这是必须的，因为我们要确保一轮

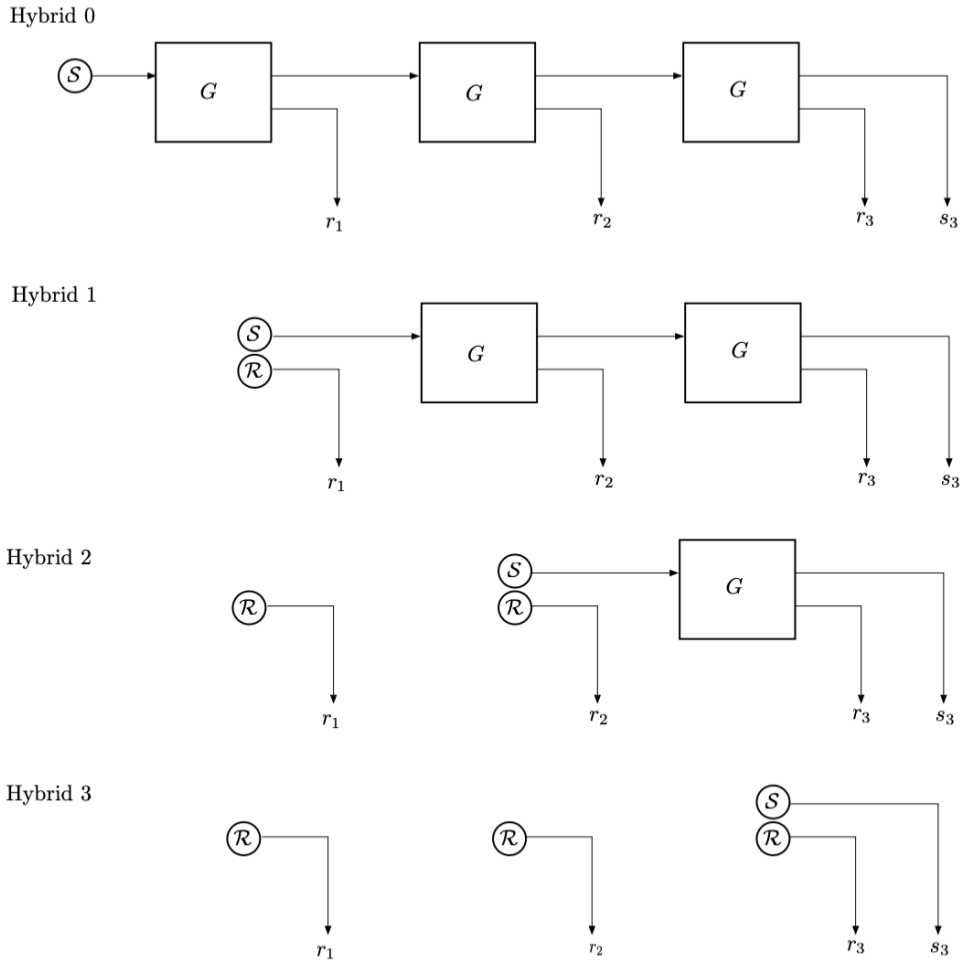


图 3.7: $n = 3$ 时，混合游戏中挑战者的计算。圆圈表示随机产生的 $\mathcal{S}$ 或 $\mathcal{R}$ 上的元素，如标签所示。

的输出可以被用作下一轮的输入。无论是对所有 G 的实例使用相同的系统参数，还是对不同的实例使用不同的系统参数，这些安全定理的证明都是完全有效的。

其次，我们要简要讨论关于混合论证的一个相当深奥的问题。为了更具体一点，我们把注意力集中在定理 3.2 的证明上（尽管类似的讨论也适用于定理 3.3 的证明，或任何其他的混合论证）。在证明该定理时，我们最终想要证明，如果存在一个能破解 G' 的有效对手 $\mathcal{A}$，那么也必然存在一个能破解 G 的有效对手。假设 $\mathcal{A}$ 是一个能破解 G' 的有效对手，那么它相对于 G' 的优势 $\epsilon(\lambda)$（我们这里明确地把它表示成安全参数 λ 的一个函数）是不可忽略不计的。这意味着存在一个常数 c 使得 $\epsilon(\lambda) \geq 1/\lambda^c$ 适用于无限多的 λ。

现在，在定理 3.2 证明之前的讨论中，我们考虑了 $n = 2$ 的特殊情况，并表明存在有效对手 $\mathcal{B}_1$ 和 $\mathcal{B}_2$，使得 $\epsilon(\lambda) \leq \delta_1(\lambda) + \delta_2(\lambda)$ 对于任意 λ 都成立，其中 $\delta_j(\lambda)$ 表示 $\mathcal{B}_j$ 相对于 G 的优势。由此可见，要么 $\delta_1(\lambda) \geq 1/2\lambda^c$ 无限频繁，要么 $\delta_2(\lambda) \geq 1/2\lambda^c$ 无限频繁。因此我们可以得出结论，要么是 $\mathcal{B}_1$ 打破 G，要么是 $\mathcal{B}_2$ 打破 G（也可能两者皆然）。因此，*存在*一个能破解 G 的有效对手：它要么是 $\mathcal{B}_1$ 要么是 $\mathcal{B}_2$，我们不知道到底是哪一个（也不必分清是哪一个）。然而无论是哪一个，它都是一个固定的对手，对所有的 λ 都是统一定义的；也就是说，它是一个固定的，以 λ 为输入的机器。

这个论证是完全有效的，并且可以扩展到任意*常数* n：我们可以构造 n 个对手 $\mathcal{B}_1, \ldots, \mathcal{B}_n$，并论证对于某个 $j \in \{1, \ldots, n\}$，对手 $\mathcal{B}_j$ 无限频繁地对 G 有优势 $1/n\lambda^c$，从而攻破 G。然而这个论证并没有扩展到 n 是 λ 的函数的情况，我们现在把它明确写成 $n(\lambda)$。问题不在于 $1/(n(\lambda)\lambda^c)$ 也许太小（其实并不是）。这个问题相当微妙，所以在我们讨论它之前，让我们先回顾一下我们所给出的（合法）证明。对于每个 λ，我们定义一个大小为 $n(\lambda)+1$ 的混合游戏序列，因此对于每个 λ，我们实际上得到了一个不同的游戏序列。事实上，我们不能说有一个单一、有限的游戏序列对所有 λ 都有效，因为 $n(\lambda) \to \infty$。尽管如此，我们还是明确地构造了一个固定的对手 $\mathcal{B}$，它是为所有 λ 统一定义的；也就是说，$\mathcal{B}$ 是一个固定的机器，它把 λ 作为输入。我们为每个 λ 定义的混合游戏序列是一个数学对象，我们对其可计算性不做任何主张，它只是在分析 $\mathcal{B}$ 时使用的一个方便的工具。

希望到现在为止，读者至少对我们试图将常数 n 的论证推广到一个函数 $n(\lambda)$ 时所产生的问题有了一个直观感觉。首先，我们甚至不清楚 $n(\lambda)$ 个对手 $\mathcal{B}_1, \ldots, \mathcal{B}_{n(\lambda)}$ 是什么意思：我们的对手应该是将 λ 作为输入的固定机器，而机器本身不应该依赖于 λ。撇开这种语言上的混乱不谈，我们对常数情况的证明只表明存在一个 “对手”，对于无限多的 λ 值来说，它能以某种方式知道 $j = j(\lambda)$ 的 “正确” 值，以便在 $(n(\lambda)+1)$ 游戏混合论证中使用——没有一个 j 常数值一定对无限多的 λ 有效。如果使用非统一的计算模型，我们实际上可以使这种类型的论证有意义，但我们在本文中不会采取这种方法。

当我们使用一个构建单一对手 $\mathcal{B}$ 的混合论证时，所有这些问题都会消失，就像我们在定理 3.2 和 3.3 的证明中所做的。然而我们重申，我们在 $n = 2$，或自然延伸到每一个常数 n 的情况下所做的原始分析是完全有效的。在这种情况下，我们构建了一个单一、固定的 $n+1$ 游戏序列，每个单独的游戏对所有 λ 都是统一定义的（就像我们的安全定义中的攻击游戏一样），我们还定义了一个有限的对手集合，每个对手都是一个固定的机器。我们重申这一点，因为在后面的内容中，我们将经常构建涉及这样的有限序列游戏的证明（事实上，定理 3.1的证明就是这种类型）。在这种情况下，每个游戏将为所有 λ 统一定义，并被称为游戏 0、游戏 1，等等。相反，当我们进行混合论证，使用非均匀的游戏序列时，我们将这些游戏表示为混合 0，混合 1 等，以避免任何可能的混淆。

3.5 下一比特测试

令 G 是一个定义在 $(\{0,1\}^{\ell}, \{0,1\}^{L})$ 上的 PRG，它能将 ℓ 比特字符串拉伸到 L 比特长。有许多方法可以让对手区分 G 的伪随机输出和真正的随机比特序列。事实上，假设一个有效对手能在给定 G 输出的前 $L-1$ 比特的情况下预测出其输出的最后一比特，则 G 从直观上来说就是不安全的，因为给定一个真随机 L 位比特序列的前 $L-1$ 位，任何人最多只有一半的机会猜中最后一比特。事实上，该结论一个有趣的逆命题也是真命题。

我们下面会正式定义 PRG 的**不可预测性 (unpredictability)** 的概念。它本质上指，给定 G 输出的前 i 比特（这里的 i 是一个对手选择的索引），能够以显著高于 $1/2$ 的概率预测出下一个比特（即第 $i+1$ 比特）是困难的。之后，我们将证明不可预测性和安全性是等价的。安全性能够导出不可预测性这一事实是很直观的：如果能够有效预测伪随机序列中的下一比特，那么我们就能直接给出一个有效的统计测试来打破安全性。相反，由不可预测性能够导出安全性，这是相当有趣的（需要花一些精力来证明），这实际上是指，如果存在任何有效的统计测试能够打破 PRG 的安全性，那么必然存在一个方法能够有效预测伪随机序列将要输出的下一比特。

攻击游戏 3.2 (不可预测的 PRG). 对于一个定义在 $(\mathcal{S}, \{0,1\}^{L})$ 上的给定 PRG G 和一个给定对手 $\mathcal{A}$，攻击游戏的过程如下：

- 对手向挑战者发送一个索引 i，其中 $0 \le i \le L-1$。
- 挑战者计算

$$s \xleftarrow{R} \mathcal{S},\ \ r \leftarrow G(s)$$

  并将 $r[0 \ldots i-1]$ 发送给对手。

- 对手输出 $g \in \{0,1\}$。

如果 $r[i] = g$，我们就说 $\mathcal{A}$ **获胜**。我们定义 $\mathcal{A}$ 相对于 G 的**优势**为 $\mathrm{Predadv}[\mathcal{A}, G]$，其值为 $|\Pr[\mathcal{A} \text{ wins}] - 1/2|$。

定义 3.3 (不可预测的 PRG). *如果 $\mathrm{Predadv}[\mathcal{A}, G]$ 对所有有效对手 $\mathcal{A}$ 来说都是可忽略不计的，那么 PRG G 就是**不可预测的** **(unpredictable)**。*

我们下面先证明安全性能够推出不可预测性。

定理 3.4. *令 G 是一个定义在 $(\mathcal{S}, \{0,1\}^{L})$ 上的 PRG。如果 G 是安全的，那么 G 就是不可预测的。*

> *特别地，对于每个如攻击游戏 3.2 那样攻击 G 的不可预测性对手 $\mathcal{A}$，必然存在一个如攻击游戏 3.1 那样攻击 G 的安全性对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足：*
>
> $$\mathrm{Predadv}[\mathcal{A}, G] = \mathrm{PRGadv}[\mathcal{B}, G]$$

证明. 令 $\mathcal{A}$ 是一个攻击 G 的不可预测性的对手，令 i 表示 $\mathcal{A}$ 所选择的索引。同时，假设 $\mathcal{A}$ 能以 $1/2+\epsilon$ 的概率赢得攻击游戏 3.2，则有 $\mathrm{Predadv}[\mathcal{A}, G] = |\epsilon|$。

我们下面以 $\mathcal{A}$ 为子程序，建立一个攻击 G 的安全性的对手 $\mathcal{B}$，其运行方式如下：

当收到来自挑战者的 $r \in \{0,1\}^L$ 时，$\mathcal{B}$ 进行以下操作：
$\mathcal{B}$ 将 $r[0 \ldots i-1]$ 发送 $\mathcal{A}$，获得 $\mathcal{A}$ 的输出 $g \in \{0,1\}$；
如果 $r[i] = g$，$\mathcal{B}$ 输出 1，否则输出 0。

对于 $b = 0, 1$，令 W_b 为 $\mathcal{B}$ 在攻击游戏 3.1 的实验 b 中输出 1 的事件。在实验 0 中，r 是 G 的一个伪随机输出，当且仅当 $r[i] = g$ 时 W_0 才会发生，因此根据定义，有：

$$\Pr[W_0] = 1/2 + \epsilon$$

在实验 1 中，r 是一个真随机比特序列。同样地，当且仅当 $r[i] = g$ 时 W_1 才会发生；然而，在这种情况下，随机变量 $r[i]$ 和 g 的值相互独立，因此有：

$$\Pr[W_1] = 1/2$$

因此可以得到：

$$\mathrm{Predadv}[\mathcal{B}, G] = |\Pr[W_1] - \Pr[W_0]| = |\epsilon| = \mathrm{PRGadv}[\mathcal{A}, G]$$

□

更有趣也更有挑战性的任务是证明不可预测性能够推出着安全性。在详细介绍具体的证明之前，我们先勾勒出高层次的想法。

首先，我们会采用一个混合论证，目的是论证如果 $\mathcal{A}$ 是一个能够有效区分伪随机 L 比特序列和真随机 L 比特序列的有效对手，那么我们必然可以构造一个有效对手 $\mathcal{B}$，他能够有效地区分：

$$x_1 \cdots x_j \; r$$

和：

$$x_1 \cdots x_j \; x_{j+1}$$

其中 j 是一个随机选出的索引，$x_1, \ldots, x_L$ 是伪随机输出，而 r 是一个随机比特。因此，对手 $\mathcal{B}$ 可以在给定 $x_1, \ldots, x_j$ 这个 “侧信息” 的情况下有效区分伪随机比特 x_{j+1} 和真随机比特 r。

我们想把 $\mathcal{B}$ 的区分优势变成预测优势，大致的想法是这样的：给定 $x_1, \ldots, x_j$，我们给 $\mathcal{B}$ 提供一个序列 $x_1 \cdots x_j \; r$，其中的 r 是一个随机选出的比特；如果 $\mathcal{B}$ 输出 1，我们对 x_{j+1} 的预测值就是 r；否则我们对 x_{j+1} 的预测值就是 $\overline{r}$（r 的补码）。

这一预测策略的有效性由以下的一般结论给出，我们称之为**区分者/预测者引理**。我们有一半设置如下：

- 一个随机变量 X，它对应于上面的 “侧信息” $x_1, \ldots, x_j$ 以及对手 $\mathcal{B}$ 所使用的任何随机硬币；
- 一个取值为 0 或 1 的随机变量 B，它对应于上面的 x_{j+1}，并可能与 X 相关；
- 一个取值为 0 或 1 的随机变量 R，它对应于上面的 r，并且与 (X, B) 无关。
- 一个函数 d，它对应于使得 $\mathcal{B}$ 的区分优势为 $|\epsilon|$ 的策略，其中 $\epsilon = \Pr[d(\mathsf{X}, \mathsf{B}) = 1] - \Pr[d(\mathsf{X}, \mathsf{R}) = 1]$。

该引理表明，如果我们用上述预测策略定义 B'，即如果 $d(\mathsf{X}, \mathsf{R}) = 1$，就有 $\mathsf{B}' = \mathsf{R}$，否则有 $\mathsf{B}' = \overline{\mathsf{R}}$，那么预测 B' 等于实际值 B 的概率正好是 $1/2 + \epsilon$。下面是该引理的精确陈述：

引理 3.5 (区分者/预测者引理). 令 X 是一个在某个集合 S 中取值的随机变量，令 B 和 R 是取值为 0 或 1 的随机变量，其中 R 在 $\{0,1\}$ 上均匀分布，且与 (X,B) 无关。令 $d : S \times \{0,1\} \to \{0,1\}$ 是一个任意的函数，并令:

$$\epsilon := \Pr[d(\mathsf{X},\mathsf{B}) = 1] - \Pr[d(\mathsf{X},\mathsf{R}) = 1]$$

随机变量 B' 的定义如下:

$$\mathsf{B}' := \begin{cases} \mathsf{R}, & d(\mathsf{X},\mathsf{R}) = 1 \\ \overline{\mathsf{R}}, & d(\mathsf{X},\mathsf{R}) \neq 1 \end{cases}$$

则有:

$$\Pr[\mathsf{B}' = \mathsf{B}] = 1/2 + \epsilon$$

证明. 我们首先以事件 $\mathsf{B} = \mathsf{R}$ 和 $\mathsf{B} = \overline{\mathsf{R}}$ 为条件计算 $\Pr[\mathsf{B}' = \mathsf{B}]$:

$$\begin{aligned} \Pr[\mathsf{B}' = \mathsf{B}] &= \Pr[\mathsf{B}' = \mathsf{B} \,|\, \mathsf{B} = \mathsf{R}] \cdot \Pr[\mathsf{B} = \mathsf{R}] + \Pr[\mathsf{B}' = \mathsf{B} \,|\, \mathsf{B} = \overline{\mathsf{R}}] \cdot \Pr[\mathsf{B} = \overline{\mathsf{R}}] \\ &= \Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \mathsf{R}] \cdot \frac{1}{2} + \Pr[d(\mathsf{X},\mathsf{R}) = 0 \,|\, \mathsf{B} = \overline{\mathsf{R}}] \cdot \frac{1}{2} \\ &= \frac{1}{2}\Big(\Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \mathsf{R}] + \big(1 - \Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \overline{\mathsf{R}}]\big)\Big) \\ &= \frac{1}{2} + \frac{1}{2}(\alpha - \beta) \end{aligned}$$

其中:

$$\alpha := \Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \mathsf{R}], \quad \beta := \Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \overline{\mathsf{R}}]$$

根据独立性，我们有:

$$\alpha = \Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \mathsf{R}] = \Pr[d(\mathsf{X},\mathsf{B}) = 1 \,|\, \mathsf{B} = \mathsf{R}] = \Pr[d(\mathsf{X},\mathsf{B}) = 1]$$

想要知道最后一个等式为什么成立，可以参考练习 3.25 的结论。

因此，我们可以计算出:

$$\begin{aligned} \epsilon &= \Pr[d(\mathsf{X},\mathsf{B}) = 1] - \Pr[d(\mathsf{X},\mathsf{R}) = 1] \\ &= \alpha - \Big(\Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \mathsf{R}] \cdot \Pr[\mathsf{B} = \mathsf{R}] + \Pr[d(\mathsf{X},\mathsf{R}) = 1 \,|\, \mathsf{B} = \overline{\mathsf{R}}] \cdot \Pr[\mathsf{B} = \overline{\mathsf{R}}] \Big) \\ &= \alpha - \frac{1}{2}(\alpha + \beta) \\ &= \frac{1}{2}(\alpha - \beta) \end{aligned}$$

这就证明了该引理。 □

定理 3.6. 令 G 是一个定义在 $(\mathcal{S}, \{0,1\}^L)$ 上的 *PRG*。如果 G 是不可预测的，那么 G 就是安全的。

> 特别地，对于每个像攻击游戏 *3.1* 那样攻击 G 的安全性的对手 $\mathcal{A}$，必然存在一个像攻击游戏 *3.2* 那样攻击 G 的不可预测性的对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足:
>
> $$\mathrm{PRGadv}[\mathcal{A}, G] = L \cdot \mathrm{Predadv}[\mathcal{B}, G]$$

证明. 令 $\mathcal{A}$ 像攻击游戏 3.1 中那样攻击 G。利用 $\mathcal{A}$，我们建立一个预测器 $\mathcal{B}$，它像攻击游戏 3.2 那样攻击 G。$\mathcal{B}$ 的工作方式如下:

- 随机选出 $\omega \in \{1, \ldots, L\}$。
- 向挑战者发送 $L - \omega$，得到一个字符串 $x \in \{0,1\}^{L-\omega}$。
- 随机生成 ω 个比特 $r_1, \ldots, r_\omega$，并将 L 比特序列 $x \,||\, r_1 \cdots r_\omega$ 发送给 $\mathcal{A}$。
- 如果 $\mathcal{A}$ 输出 1，则 $\mathcal{B}$ 输出 r_1，否则 $\mathcal{B}$ 输出 $\bar{r}_1$。

为了分析 $\mathcal{B}$，我们考虑 $L+1$ 个混合游戏，称为混合 0，混合 1，...，混合 L。对于 $j = 0, \ldots, L$，我们定义混合 j 为 $\mathcal{A}$ 和挑战者之间的游戏，挑战者生成一个由 $L - j$ 个伪随机比特和 j 个真随机比特组成的序列 r；也就是说，挑战者随机选择 $s \in \mathcal{S}$ 和 $t \in \{0,1\}^j$，并向 $\mathcal{A}$ 发送比特序列：

$$r = G(s)[0 \ldots L - j - 1] \,||\, t$$

和之前一样，$\mathcal{A}$ 在游戏结束时输出 0 或 1，我们定义 p_j 为 $\mathcal{A}$ 在混合 j 中输出 1 的概率。注意 p_0 是 $\mathcal{A}$ 在攻击游戏 3.1 的实验 0 中输出 1 的概率，而 p_L 是 $\mathcal{A}$ 在攻击游戏 3.1 的实验 1 中输出 1 的概率。

令 W 为 $\mathcal{B}$ 在攻击游戏 3.2 中获胜的事件（即他正确预测了下一个比特），那么我们有：

$$\begin{aligned}
\Pr[W] &= \sum_{j=1}^{L} \Pr[W|\omega = j] \cdot \Pr[\omega = j] \\
&= \frac{1}{L} \sum_{j=1}^{L} \Pr[W \,|\, \omega = j] \\
&= \frac{1}{L} \sum_{j=1}^{L} \left(\frac{1}{2} + p_{j-1} - p_j \right) \quad \text{（根据引理 3.5）} \\
&= \frac{1}{2} + \frac{1}{L}(p_0 - p_L)
\end{aligned}$$

这就证明了该定理。 □

3.6 案例研究：Salsa 和 ChaCha PRG

在实践中，有许多建立 PRG 和流密码的方法。一种方法是使用 3.4.2 小节中介绍的 Blum-Micali 范式来构建 PRG。另一种方法，我们将在第五章中介绍，它用一种更加通用的，称作伪随机函数的原语在计数器模式下构建 PRG 和流密码。我们下面介绍一种使用后者的实际构造。

Salsa20/12 和 Salsa20/20 是由丹尼尔 · 伯恩斯坦于 2005 年设计的快速流密码。Salsa20/12 是被选入 eStream 流密码组合的四种 Profile 1 流密码之一。eStream 是一个选定适合实际使用场景的快速安全的流密码的项目。伯恩斯坦又于 2008 年提出了 Salsa20/12 和 Salsa20/20 的变体，分别被称作 ChaCha12 和 ChaCha20。这些流密码已被纳入一些广泛部署的协议，如 TLS 和 SSH。

让我们先简单介绍一下 Salsa 和 ChaCha 流密码家族的底层 PRG。这些 PRG 将一个 256 比特种子和一个 64 比特的 nonce 作为输入。现在我们忽略 nonce，先简单地将其设置为 0，我们会在本节末尾讨论 nonce 的作用。Salsa 和 ChaCha PRG 都遵循图 3.8 所示的顶层设计结构。它们都包含两个组件：

- 一个填充函数 $\text{pad}(s, j, 0)$，将 256 比特的种子 s 和 64 比特的计数器 j 结合，形成一个 512 比特的分组。第三个输入是一个 64 比特的 nonce，我们目前先把它置为 0。

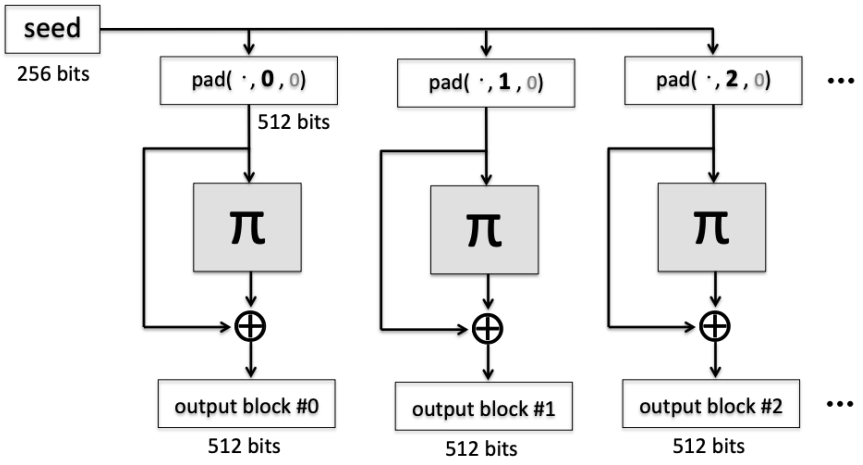


图 3.8: Salsa 和 ChaCha PRG 的示意图

- 一个固定、公开的置换 $\pi : \{0,1\}^{512} \to \{0,1\}^{512}$。

这两个组件使用以下算法（见图 3.8）输出 $L < 2^{64}$ 个伪随机分组，每个分组 512 比特长：

输入：种子 $s \in \{0,1\}^{256}$
1. 对于 $j \leftarrow 0$ 到 $L-1$：
2. $\quad h_j \leftarrow \text{pad}(s, j, 0) \in \{0,1\}^{512}$
3. $\quad r_j \leftarrow \pi(h_j) \oplus h_j$
4. 输出 $(r_0, \ldots, r_{L-1})$。

最终的 PRG 输出是 $512 \cdot L$ 比特长的序列。我们注意到，在 Salsa 和 ChaCha 中，第 3 行中的异或运算是一个稍微复杂的操作：512 比特的操作数 h_j 和 $\pi(h_j)$ 被分成 16 个字，每个字长 32 比特，然后逐字计算模 2^{32} 加法。

Salsa 和 ChaCha 的设计是高度可并行的，并且可以利用多处理器核心来加快加密速度。此外，它还能实现对输出分组的随机访问：想要输出分组的编号 j，并不需要先把所有前序分组都计算出来。基于 Blum-Micali 范式的生成器则不具备这些特性。

我们将在下一章的练习 4.23 中分析 Salsa 和 ChaCha 的安全性，为此我们还需要一些其他的工具。

一些细节。 我们下面简单介绍填充函数 $\text{pad}(s, j, n)$ 和 ChaCha20 中使用的置换 π。填充函数的输入是一个 256 比特的种子 $s_0, \ldots, s_7 \in \{0,1\}^{32}$，一个 64 比特的计数器 $j_0, j_1 \in \{0,1\}^{32}$ 和一个 64 比特的 nonce $n_0, n_1 \in \{0,1\}^{32}$。它输出一个 512 比特的分组，表示为 $x_0, \ldots, x_{15} \in \{0,1\}^{32}$。输出被组织为一个由 32 比特长的字所构成的 4×4 矩阵，如下所示：

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \longleftarrow \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ j_0 & j_1 & n_0 & n_1 \end{pmatrix} \tag{3.11}$$

其中 c_0, c_1, c_2, c_3 是固定的 32 比特常数。

置换 $\pi : \{0,1\}^{512} \to \{0,1\}^{512}$ 是通过对一个简单的置换进行固定次数的迭代来实现的。π 的 512 比

特输入被当作一个由 32 比特的字构成的 4×4 数组，记作 $x_0, \ldots, x_{15}$。在 ChaCha20 中，函数 π 是通过重复 10 次以下步骤来实现的：

(1) QuarterRound(x_0, x_4, x_8, x_{12}), (2) QuarterRound(x_1, x_5, x_9, x_{13}),
(3) QuarterRound$(x_2, x_6, x_{10}, x_{14})$, (4) QuarterRound$(x_3, x_7, x_{11}, x_{15})$,
(5) QuarterRound$(x_0, x_5, x_{10}, x_{15})$, (6) QuarterRound$(x_1, x_6, x_{11}, x_{12})$,
(7) QuarterRound(x_2, x_7, x_8, x_{13}), (8) QuarterRound(x_3, x_4, x_9, x_{14})

这里，QuarterRound(a, b, c, d) 的定义可参见下面的 C 程序所表示的步骤：

```
a += b;  d ^= a;  d <<<= 16;
c += d;  b ^= c;  b <<<= 12;
a += b;  d ^= a;  d <<<= 8;
c += d;  b ^= c;  b <<<= 7;
```

注意到 QuarterRound 的前四个调用，即步骤 (1-4)，从左到右分别应用于 4×4 矩阵的四列上。接下来的四次调用，即步骤 (5-8)，分别应用于矩阵的四条对角线。这就完成了我们对 ChaCha20 的描述，只是我们还需要讨论一下 nonce 的使用。

使用 nonce。 虽然我们到目前为止讨论的 PRG 只把种子作为输入，但在实践中使用的许多 PRG 还需要一个额外的输入，称为 *nonce*。也就是说，PRG 是一个函数 $G : \mathcal{S} \times \mathcal{N} \to \mathcal{R}$，其中 $\mathcal{S}$ 和 $\mathcal{R}$ 和之前一样，而 $\mathcal{N}$ 被称为 *nonce* 空间。nonce 能让我们由一个种子 s 产生多个伪随机输出。也就是说，$G(s, n_0)$ 是一个伪随机输出，而当 $n_1 \neq n_0$ 时，$G(s, n_1)$ 就是另一个伪随机输出。nonce 将 PRG 变成了一个更强大的原语，称为伪随机函数 *(pseudo-random function)*，我们将在下一章更详细地讨论它。正如我们将看到的，安全的伪随机函数能让我们使用同一个种子安全地加密多条消息。

3.7 案例研究：线性生成器

在这一节中，我们将看到两个由线性函数构建的 PRG 的例子。这两个生成器都遵循 3.4.2 小节中介绍的 Blum-Micali 范式。我们的第一个例子称为线性同构生成器，它是完全不安全的。我们之所以以它为例，是想举例说明攻击 PRG 时可能会出现的一些优美的数学构造。我们的第二个例子称为子集和生成器，如果我们假设经典子集和问题的某个特定版本是困难的，就可以证明它是一个安全的 PRG。

3.7.1 一个密码分析的例子：线性同构生成器

线性同构生成器 (Linear congruential generators, LCG) 在统计模拟中被用于产生伪随机值。它们速度快，容易实现，并且被广泛部署。LCG 的变体也在 glibc、Microsoft Visual Basic 和 Java 运行时的早期版本中被用于引入随机性。虽然这些生成器可能足以用于模拟，但它们绝不应该用在密码学应用中，因为它们作为 PRG 是不安全的。特别是它们是可预测的：给定 LCG 的几个连续输出，很容易计算出所有后续的输出。在本节中，我们通过展示一种预测算法来描述对 LCG 的攻击。

基本的线性同构生成器由四个公共系统参数指定：一个整数 q，两个常数 $a, b \in \{0, \ldots, q-1\}$ 和一个正整数 $w \leq q$。选取的常数 a 应与 q 互素。我们用 $\mathcal{S}_q$ 和 $\mathcal{R}$ 来表示集合：

$$\mathcal{S}_q := \{0, \ldots, q-1\}; \qquad \mathcal{R} := \{0, \ldots, \lfloor (q-1)/w \rfloor\}$$

这里的 $\lfloor\cdot\rfloor$ 是向下取整函数：对于一个实数 x，$\lfloor x\rfloor$ 是小于或等于 x 的最大整数。现在，以 $s\in\mathcal{S}_q$ 为种子的生成器 $G_{\text{lcg}}:\mathcal{S}_q\to\mathcal{R}\times\mathcal{S}_q$ 的定义如下：

$$G_{\text{lcg}}(s):=\big(\lfloor s/w\rfloor,\ as+b \bmod q\big)$$

当 w 是 2 的幂，比如 $w=2^t$ 时，$\lfloor s/w\rfloor$ 的计算其实就是简单地抹除 s 的 t 个最小有效位。因此，$G_{\text{lcg}}(s)$ 的左边就是抹去 s 的 t 个最小有效位的结果。

生成器 $G_{\text{lcg}}(s)$ 显然是不安全的，因为只要给定 $s':=as+b \bmod q$，我们就可以直接重建 s，然后将 $\lfloor s/w\rfloor$ 从随机值中区分出来。然而，考虑下面的一种 Blum-Micali 构造的变体，其中，最终的 $\mathcal{S}_q$ 中的值不被输出：

$G_{\text{lcg}}^{(n)}(s):=$
$s_0\leftarrow s$
对于 $j\leftarrow 0$ 到 $L-1$：
$\quad r_i\leftarrow\lfloor s_{i-1}/w\rfloor,\quad s_i\leftarrow as_{i-1}+b \bmod q$
输出 $(r_0,\ldots,r_n)$。

我们将每一次循环称为 LCG 的一次迭代，并称每一个 $r_1,\ldots,r_n$ 为一次迭代的输出。

不同的实现会使用不同的系统参数 q，a，b 和 w。例如，Java 8 中的 `Math.random` 函数使用 $q=2^{48}$，$w=2^{22}$ 以及十六进制常数 $a=$ `0x5DEECE66D`，$b=$ `0x0B`。因此，LCG 的每次迭代都会输出 48 比特的状态 s_i 的前 $48-22=26$ 比特。

这个 Java 8 生成器所使用的参数对于安全应用来说显然太小了，因为生成器的第一次迭代输出就会揭示 s 中除了 22 比特之外的所有其他比特。攻击者可以通过穷举搜索轻易地恢复未知的这 22 比特：对于这 22 比特的每一个可能的值，它都生成一个候选种子 $\hat{s}$。它可以从 $\hat{s}$ 出发计算若干后续的输出，并将其与从实际的生成器中观察到的后续比特进行对比，以此来测试 $\hat{s}$ 是否是正确的种子。只要遍历所有 2^{22} 个候选种子（约 400 万个），攻击者就能最终找到正确的种子 s，然后就可以预测胜生成器的所有后续输出。这种攻击在现代处理器上的运行时间只有不到一秒。

就算 LCG 的参数大到足以抵抗穷举搜索，比如说 $q=2^{512}$，生成器 G_{lcg} 也是不安全的。就算你可以从各种软件库中找到它，也永远不要把它用在安全应用中。已知的针对 LCG 的攻击表明，即使生成器每次迭代只输出几个比特，我们仍有可能基于几个连续的输出预测整个序列。让我们看看这种攻击的一个优雅的版本。

密码分析。 假设 q 很大（例如 $q=2^{512}$），LCG G_{lcg} 每次迭代都会输出状态 s 中大约一半的比特，就像 Java 8 中的 `Math.random` 生成器一样。鉴于种子 s 的大小，对其进行穷举式搜索是不可能的。然而，我们下面展示，如何用仅仅两个连续迭代的输出来快速地预测生成器。

更确切地说，假设对于某个固定的 $c>0$，例如 $c=32$，有 $w<\sqrt{q}/c$。这意味着在每次迭代中，生成器所输出的比特数都只略长于当前内部状态的比特数的一半。假设攻击者得到了生成器的连续两个输出 $r_i,r_{i+1}\in\mathcal{R}$。我们下面展示它预测剩余序列的方法。对于某个未知的 $s_i\in\mathcal{S}_q$，攻击者知道：

$$r_i=\lfloor s_i/w\rfloor,\qquad r_{i+1}=\lfloor s_{i+1}/w\rfloor=\lfloor(as_i+b \bmod q)/w\rfloor$$

我们有：

$$r_i\cdot w+e_0=s_i,\qquad r_{i+1}\cdot w+e_1=(as_i+b \bmod q)$$

其中，e_0 和 e_1 是 s_i 和 s_{i+1} 除以 w 后的余数；特别地，我们有 $0\le e_0,e_1<w<\sqrt{q}/c$。$e_0$ 和 e_1 小于 $\sqrt{q}$ 这一事实是攻击能够成功的一个重要因素。接下来，我们用 s 代换 s_i，并引入一个整数变量 x 来消

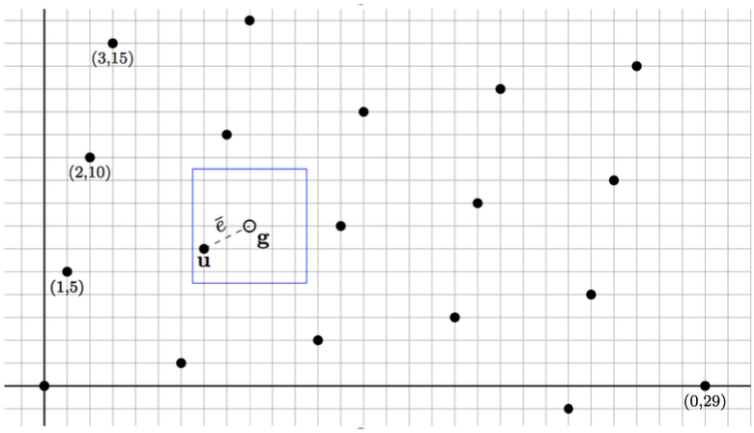


图 3.9: 与攻击 LCG 相关的二维网格。这里的网格是由向量 $(1,5)^{\mathrm{T}}$ 和 $(0,29)^{\mathrm{T}}$ 生成的。攻击者持有向量 $\boldsymbol{g}=(9,7)^{\mathrm{T}}$，并希望找到最接近的网格向量 $\boldsymbol{u}$。在这本图中确实只有一个 “接近” $\boldsymbol{g}$ 的网格向量。

除 mod q，得到：

$$r_i \cdot w + e_0 = s, \qquad r_{i+1} \cdot w + e_1 = as + b + qx$$

攻击者不知道 x，s，e_0 和 e_1 的值，但它知道 r_i，r_{i+1}，w，a 和 b。最后，重新排列各项，把涉及 x 和 s 的项放到左边，得到：

$$s = r_i \cdot w + e_0, \qquad as + qx = r_{i+1}w - b + e_1 \tag{3.12}$$

我们可以将式 3.12 重写为向量形式：

$$s \cdot \begin{pmatrix} 1 \\ a \end{pmatrix} + x \cdot \begin{pmatrix} 0 \\ q \end{pmatrix} = \boldsymbol{g} + \boldsymbol{e} \qquad \text{where} \qquad \boldsymbol{g} := \begin{pmatrix} r_i w \\ r_{i+1}w - b \end{pmatrix}, \quad \boldsymbol{e} := \begin{pmatrix} e_0 \\ e_1 \end{pmatrix} \tag{3.13}$$

令 $\boldsymbol{u} \in \mathbb{Z}^2$ 表示未知向量 $\boldsymbol{u} := \boldsymbol{g} + \boldsymbol{e} = s \cdot (1,a)^{\mathrm{T}} + x \cdot (0,q)^{\mathrm{T}}$。如果攻击者能够找到 $\boldsymbol{u}$，他就可以通过线性代数轻松地从 $\boldsymbol{u}$ 中恢复 s 和 x。利用 s，他就可以预测 PRG 的其余输出。因此，想要破解生成器，只需要找到这样的向量 $\boldsymbol{u}$ 即可。攻击者知道向量 $\boldsymbol{g} \in \mathbb{Z}^2$，此外，他也知道 $\boldsymbol{e}$ 很短，即 $\|\boldsymbol{e}\|_\infty$ 最大为 $\sqrt{q}/c$。因此，他知道 $\boldsymbol{u}$ 是 “接近” $\boldsymbol{g}$ 的。

我们下面展示如何由 $\boldsymbol{g}$ 找到 $\boldsymbol{u}$。考虑向量 $(1,a)^{\mathrm{T}}$ 和 $(0,q)^{\mathrm{T}}$ 的所有整系数线性组合所构成的集合。我们用 $\mathcal{L}_a$ 表示这个集合，它是 $\mathbb{Z}^2$ 的一个子集，包含像 $(1,a)^{\mathrm{T}}$，$(2,2a)^{\mathrm{T}}$，$(3,3a-2q)^{\mathrm{T}}$ 这样的向量。图 3.9 展示了集合 $\mathcal{L}_a$，图中的实心点是向量 $(1,a)^{\mathrm{T}}$ 和 $(0,q)^{\mathrm{T}}$ 的整系数线性组合。我们称集合 $\mathcal{L}_a$ 为由向量 $(1,a)^{\mathrm{T}}$ 和 $(0,q)^{\mathrm{T}}$ 所生成的二维**网格 (lattice)**。

现在，攻击者有一个向量 $\boldsymbol{g} \in \mathbb{Z}^2$，并且知道他的目标向量 $\boldsymbol{u} \in \mathcal{L}_a$ 接近 $\boldsymbol{g}$。如果他能在 $\mathcal{L}_a$ 中找到与 $\boldsymbol{g}$ 最接近的向量，那么这个向量很有可能就是所需的向量 $\boldsymbol{u}$。下面的定理将表明，对于大多数的 $a \in \mathcal{S}_q$ 来说，情况确实如此。

引理 3.7. *对于 $\mathcal{S}_q$ 中的至少 $(1-16/c^2)\cdot q$ 个 a，网格 $\mathcal{L}_a \subseteq \mathbb{Z}^2$ 有如下性质：对于每个 $\boldsymbol{g} \in \mathbb{Z}^2$，最多只有一个向量 $\boldsymbol{u} \in \mathcal{L}_a$ 使得 $\|\boldsymbol{g}-\boldsymbol{u}\|_\infty < \sqrt{q}/c$。*

在引理 3.7 中取 $c = 32$（因此 $w = \sqrt{q}/30$），那么对于 98% 的 $a \in \mathcal{S}_q$ 来说，$\mathcal{L}_a$ 中离 $\boldsymbol{g}$ 最近的向量就恰好是所需的向量 $\boldsymbol{u}$。在证明该定理之前，我们首先完成对攻击的描述。

剩下的工作就是有效地找到 $\mathcal{L}_a$ 中离 $\boldsymbol{g}$ 最近的向量。这个问题是一个一般问题的特例，它叫做**最近向量问题 (closest vector problem)**：给定一个网格 $\mathcal{L}$ 和一个向量 $\boldsymbol{g}$，找到 $\mathcal{L}$ 中离 $\boldsymbol{g}$ 最近的向量。有了这个算法，攻击者就可以根据生成器的两个输出 r_i 和 r_{i+1} 恢复 LCG 的内部状态 s_i，并预测剩余的序列。这种攻击对 98% 的 $a \in \mathcal{S}_q$ 都有效。

完整起见，我们注意到，还有 2% 的 $a \in \mathcal{S}_q$ 所发起的攻击会失败，比如 $a = 1$ 和 $a = 2$。对于这些 a，在 $\mathcal{L}_a$ 中可能有多个接近给定的 $\boldsymbol{g}$ 的网格向量。我们把设计一个对于 $\mathcal{S}_q$ 中那些不适用于引理 3.7 的 a 有效的攻击作为一个有趣的练习。最后，我们以对引理 3.7 的证明结束本节。

引理 3.7 的证明. 令 $\boldsymbol{g} \in \mathbb{Z}^2$，假设 $\mathcal{L}_a$ 中存在两个接近 $\boldsymbol{g}$ 的向量 $\boldsymbol{u}_0$ 和 $\boldsymbol{u}_1$，即 $\|\boldsymbol{u}_i - \boldsymbol{g}\|_\infty < \sqrt{q}/c$ 对于 $i = 0, 1$ 成立。那么 $\boldsymbol{u}_0$ 和 $\boldsymbol{u}_1$ 一定是相互接近的。事实上，根据三角不等式，我们有：

$$\|\boldsymbol{u}_0 - \boldsymbol{u}_1\|_\infty \le \|\boldsymbol{u}_0 - \boldsymbol{g}\|_\infty + \|\boldsymbol{g} - \boldsymbol{u}_1\|_\infty \le 2\sqrt{q}/c$$

由于任何网格在加法下都是封闭的，因此我们可以看出 $\boldsymbol{u} := \boldsymbol{u}_0 - \boldsymbol{u}_1$ 是网格 $\mathcal{L}_a$ 中的一个向量，并且我们可以得出结论：$\mathcal{L}_a$ 中一定包含一个 “短” 向量，即一个范数最大为 $B := 2\sqrt{q}/c$ 的非零向量。因此，我们对使得 $\mathcal{L}_a$ 包含这样一个短向量的 “坏的” a 的数量进行约束。

我们首先考虑 q 是素数的情况。我们声称，每个短向量最多包含在一个网格 $\mathcal{L}_a$ 中，因此坏的 a 的数量最多就是短向量的数量。假设 $\boldsymbol{t} = (s, y)^{\mathrm{T}} \in \mathbb{Z}^2$ 是某个非零向量，满足 $\|\boldsymbol{t}\|_\infty \le B$。假设对于某个 $a \in \mathcal{S}_q$ 有 $\boldsymbol{t} \in \mathcal{L}_a$，那么存在整数 s_a 和 x_a 使得 $s_a \cdot (1, a)^{\mathrm{T}} + x_a \cdot (0, q)^{\mathrm{T}} = \boldsymbol{t} = (s, y)^{\mathrm{T}}$。由此，我们得到 $s = s_a$ 和 $y = as \bmod q$。此外，我们有 $s \neq 0$，因为如果不是这样，就有 $\boldsymbol{t} = \boldsymbol{0}$。因为 $y = as \bmod q$，$s \neq 0$，所以 a 的值是唯一确定的，即 $a = ys^{-1} \bmod q$。因此，当 q 是素数时，每个非零的短向量 $\boldsymbol{t}$ 都最多只包含在一个 $a \in \mathcal{S}_q$ 的网格中。进而可知，坏的 a 的数量最多就是短向量的数量，也就是 $(2B)^2 = 16q/c^2$。

当 q 不是素数时，对坏的 a 的数量的约束同样成立。为了说明原因，考虑一个特定的非零 $s \in \mathcal{S}_q$，令 $d = \gcd(s, q)$。如上所述，只有当有一个 $a \in \mathcal{S}_q$ 满足 $as \equiv y \pmod q$ 时，向量 $\boldsymbol{t} = (s, y)^{\mathrm{T}}$ 才包含在某个网格 $\mathcal{L}_a$ 中。这意味着 y 必须是 d 的倍数，所以我们只需要考虑 y 的 $2B/d$ 个可能取值。对于每个这样的 y，向量 $\boldsymbol{t} = (s, y)^{\mathrm{T}}$ 最多包含在 d 个网格中。由于 s 有 $2B$ 个可能取值，这表明坏的 a 的数量以 $d \cdot 2B/d \cdot 2B = (2B)^2$ 为界，这与 q 是素数的情况相同。

总之，$\mathcal{S}_q$ 中最多有 $16q/c^2$ 个坏的 a。因此，对于 $\mathcal{S}_q$ 中的 $(1 - 16/c^2) \cdot q$ 个 a 值，网格 $\mathcal{L}_a$ 中不包含非零短向量，故而该引理得证。 □

3.7.2 子集和生成器

接下来我们展示如何基于简单的线性运算构建一个伪随机生成器。假设经典**子集和问题** *(subset sum problem)* 的某个随机版本是困难的，那么这个生成器是安全的。

模子集问题。 令 q 是一个正整数，令 $\mathcal{S}_q := \{0, \ldots, q-1\}$。在 $\mathcal{S}_q$ 中选择 n 个整数 $\boldsymbol{a} := (a_0, \ldots, a_{n-1})$，并定义子集和函数 $f_{\boldsymbol{a}} : \{0,1\}^n \to \mathcal{S}_q$ 为：

$$f_{\boldsymbol{a}}(\boldsymbol{s}) := \sum_{i : s_i = 1} a_i \bmod q$$

比如说 $f_{\boldsymbol{a}}(101101) = a_0 + a_1 + a_2 + a_4 \bmod q$。现在，对于一个目标整数 $t \in \mathcal{S}_q$，模子集问题的定义如下：

给定 $(q, \boldsymbol{a}, t)$ 作为输入，如果存在一个向量 $\boldsymbol{s} \in \{0,1\}^n$ 使得 $f_{\boldsymbol{a}}(\boldsymbol{s}) = t$，则将其输出。

换句话说，该问题是，如果函数 $f_{\boldsymbol{a}}(\cdot)$ 存在反函数，就通过寻找 t 的原像的方式来求得该反函数。模子集问题目前被认为是 NP 困难的。

子集和与 PRG。 子集问题自然而然地给出了以下的 PRG：在设置时选择一个固定整数 q，并从 $\mathcal{S}_q$ 中随机选择 n 个整数 $\vec{a} := (a_0, \ldots, a_{n-1})$。PRG $G_{q,\vec{a}}$ 将一个种子 $\boldsymbol{s} \in \{0,1\}^n$ 作为输入，并输出一个 $\mathcal{S}_q$ 上的伪随机值。其定义为：

$$G_{q,\vec{a}}(\boldsymbol{s}) := \sum_{i=1}^{n} a_i \cdot s_i \bmod q$$

该 PRG 将一个 n 比特的种子拉伸为一个 $\log_2 q$ 比特的输出。选择 n 和 q 使得 $2n = \log_2 q$，我们就可以得到一个 PRG，其输出长度是输入长度的两倍。我们可以将其插入 Blum-Micali 构造中以进一步扩大输出。

虽然这个 PRG 比 3.6 节中的 ChaCha20 等自定义构造要慢得多，但每一位输出所对应的工作都只是 $\mathcal{S}_q$ 上的一个模加法，这可能适合一些对时间不敏感的应用。

Impagliazzo 和 Naor 表明，攻击基于 $G_{q,\vec{a}}$ 的 PRG 的难度与解决模子集和问题的某个随机化变体一样。虽然已经有很多工作试图解决模子集问题，但对于较大的 n，比如 $n > 1000$，当 $2n = \log_2 q$ 时，该问题似乎仍然是很难的，这就意味着 $G_{q,\vec{a}}$ 作为 PRG 是安全的。

变体。 Fischer 和 Stern 等人提出子集和生成器的一个变体：

$$G_{q,\vec{a}}(\boldsymbol{s}) := A \cdot \boldsymbol{s} \bmod q$$

其中，q 是一个小素数，A 是一个 $\mathcal{S}_q^{n \times m}$ 上的随机矩阵，$n < m$，并且种子 $\boldsymbol{s}$ 均匀分布在 $\{0,1\}^m$ 上。该生成器将 m 比特的种子映射为 $n \log_2 q$ 比特的输出。我们将在第十六章进一步讨论这个生成器。

3.8 案例研究：对 DVD 加密系统的密码学分析

内容加扰系统 (Content Scrambling System, CSS) 是一个用于保护 DVD 光盘上电影的系统。它使用一种称为 CSS 的流密码来加密电影内容。CSS 设计于 20 世纪 80 年代，当时，可出口的加密算法被限制在 40 比特密钥以内。因此，CSS 使用 40 比特的密钥对电影进行加密。虽然我们现在已经知道，使用 40 比特密钥的密码是非常不安全的，但我们将表明，CSS 流密码格外弱，以至于我们可以找到比穷举所有 2^{40} 个密钥耗时更短的破解方法。它为密码分析提供了一个有趣的机会。

线性反馈移位寄存器 (Linear feedback shift register, LFSR)。 CSS 流密码由两个 LFSR 构建。一个 n 比特 LFSR 由一组整数 $V := \{v_1, \ldots, v_d\}$ 定义，其中每个 v_i 都在 $\{0, \ldots, n-1\}$ 中。V 中的元素被称为**抽头位置 (tap position)**。一个 LFSR 能够提供一个 PRG（见图 3.10），其原理如下所示：

输入：$s = (b_{n-1}, \ldots, n_0) \in \{0,1\}^n$，其中 $s \neq 0^n$
输出：$y \in \{0,1\}^\ell$，其中 $\ell > n$
对于 $i \leftarrow 1$ 到 ℓ：
　　输出 b_0　　// 输出一比特

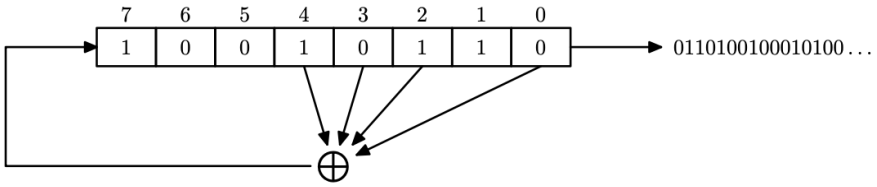


图 3.10: 8 比特的线性反馈移位寄存器 $\{4, 3, 2, 0\}$

$b \leftarrow b_{v_1} \oplus \cdots \oplus b_{v_d}$ // 计算反馈比特
$s \leftarrow (b, b_{n-1}, \ldots, b_1)$ // 将寄存器中的比特右移

LFSR 每个时钟周期输出一个比特。请注意，如果一个 LFSR 在启动时状态为 $s = 0^n$，那么它的输出会退化为全部是 0。由于这个原因，种子中的某一比特必须总是被置为 1。

LFSR 可以用很少的晶体管在硬件上实现。因此，由 LFSR 构建的流密码对于低成本的消费电子产品（如 DVD 播放器、手机和蓝牙设备）很有吸引力。

来自 LSFR 的流密码。 一个单一的 LFSR 作为 PRG 是完全不安全的，因为给定其输出的 n 个连续比特，很容易就能计算出所有的后续比特。然而，使用一个非线性组件把几个 LFSR 组合起来，我们就有可能得到在某种程度上（弱）安全的 PRG。一种属于 eStream 组合的流密码 Trivium 就是这样构建出来的。

从 LFSR 构建流密码的一种方法是并行地运行几个 LFSR，并使用非线性操作组合它们的输出。接下来描述的 CSS 流密码使用整数域上的加法将两个 LFSR 组合起来。而被用于加密 GSM 手机流量的 A5/1 流密码组合了三个 LFSR 的输出。蓝牙 E0 流密码使用一个 2 比特的有限状态机将四个 LFSR 组合起来。所有这些算法都已被证明是不安全的，并且不应该被使用，这是因为对于这些密码，恢复明文所需的时间远远少于对密钥空间进行穷举搜索的耗时。

另一种方法是只运行一个 LFSR，并对其内部状态进行非线性操作来产生输出。用于加密 3GPP 手机流量的 `snow` 3G 密码就是这样操作的。

CSS 流密码。 CSS 流密码是由图 3.11 所示的 PRG 建立的。该 PRG 的工作原理如下：

输入：种子 $s \in \{0,1\}^{40}$
输出：ℓ 个比特

令 $s = s_1 \| s_2$，其中 $s_1 \in \{0,1\}^{16}$，$s_2 \in \{0,1\}^{24}$
将 $1 \| s_1$ 加载到一个 17 比特的 LFSR 中
将 $1 \| s_2$ 加载到一个 25 比特的 LFSR 中
令 $c \leftarrow 0$ // 进位
对于 $i = 1, \ldots, \ell$:
    将两个 LFSR 运行 8 个周期，得到 $x_i, y_i \in \{0,1\}^8$
    将 x_i 和 y_i 视作 $\{0, \ldots, 255\}$ 中的两个整数
    输出 $z_i := x_i + y_i + c \bmod 256$
    如果 $x_i + y_i > 255$，则令 $c \leftarrow 1$，否则令 $c \leftarrow 0$ // 进位

该 PRG 每次迭代输出一个字节。在 s_1 和 s_2 中预留的 1 确保 LFSR 不会被初始化为全 0 状态。两个

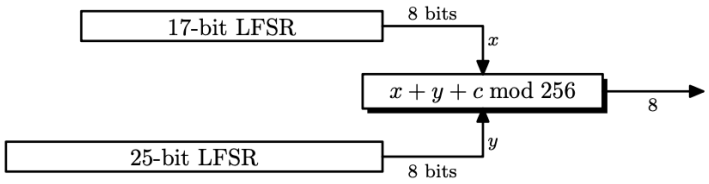


图 3.11: CSS 流密码

LFSR 的抽头是固定的。17 比特的 LFSR 使用的是抽头 $\{14, 0\}$，25 比特的 LFSR 使用抽头 $\{12, 4, 3, 0\}$。

我们展示的 CSS PRG 是 CSS 的一个小变体，它描述起来比较容易，但和真正的 CSS 具有同等的安全性。在真正的 CSS 中，对于 17 比特的 LFSR，我们不是在初始种子中预置 1，而是在第 9 比特的位置插入 1；而对于 25 比特的 LFSR，是在第 22 比特处插入 1。此外，真正的 CSS 会丢弃 17 比特 LFSR 输出的第一个字节和 25 比特 LFSR 输出的前两个字节。这两个问题都不会影响到接下来的分析。

CSS 的不安全性。 给定一个 PRG 的输出，通过对种子空间进行穷举搜索，我们显然可以在 2^{40} 次计算内恢复秘密的种子。我们下面展示一种更快的攻击方法，它只需要进行 2^{16} 次猜测。假设我们得到了 PRG 输出的前 100 个字节 $\bar{z} := (z_1, z_2, \ldots)$。该攻击基于以下观察：

> 令 (x_1, x_2, x_3) 和 (y_1, y_2, y_3) 分别为 17 比特和 25 比特的 LFSR 所输出的前三个字节。那么：
>
> $$(2^{16}x_3 + 2^8 x_2 + x_1) + (2^{16}y_3 + 2^8 y_2 + y_1) \equiv (2^{16}z_3 + 2^8 z_2 + z_1) \pmod{2^{24}}$$
>
> 因此，一旦 (z_1, z_2, z_3) 和 (x_1, x_2, x_3) 都已经知道了，我们就可以很容易地计算出 (y_1, y_2, y_3)，进而可以很容易地得到 25 比特的 LFSR 的初始状态 s_2。

有了这个观察，攻击者就可以通过尝试所有可能的 16 比特的 s_1 的值来恢复种子 s。对于每个猜测的 s_1，它计算出 17 比特 LFSR 对应的输出 (x_1, x_2, x_3)。利用上面的观察，它就能够获得一个 25 比特 LFSR 的候选种子 s_2。然后，为了确认 $\hat{s} := s_1 \| s_2$ 是否是正确的秘密种子，它使用种子 $\hat{s}$ 运行 PRG 的 100 次迭代，并将输出的结果与给定的序列 $\bar{z}$ 进行比较。如果序列不匹配，就换一个 s_1 重新进行计算。一旦攻击者找到了正确的 s_1，生成的序列将与给定的 $\bar{z}$ 相匹配，在这种情况下，攻击者就得到了正确的秘密种子 $\hat{s} := s_1 \| s_2$。

我们上面的分析表明，在期望上对 s_1 进行 2^{15} 次猜测后，就可以找到整个种子 s。这比单纯地进行 2^{40} 次穷举搜索攻击要快得多。

3.9 案例研究：对 RC4 流密码的密码学分析

RC4 流密码由 Ron Rivest 在 1987 年设计，历史上曾用于保护网络流量（在 SSL/TLS 协议中）和无线流量（在 802.11b WEP 协议中）的安全。它被设计为可在内存很小的 8 位处理器上运行。虽然 RC4 仍在使用，但它已被证明容易遭受一些显著的攻击，因而不应该被应用在新的项目中。我们对 RC4 的讨论可以作为流密码分析的一个优雅的例子。

RC4 密码的核心是一个 PRG，称为 RC4 PRG。PRG 维护一个内部状态，包含一个 256 字节的数

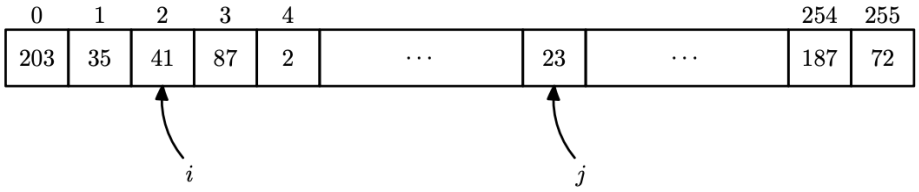


图 3.12: RC4 内部状态的一个例子

组 S 和两个额外的字节 i, j，作为指入 S 的两个指针。数组 S 包含 $\{0, \ldots, 255\}$ 中的所有的数字，且每个数字正好出现一次。图 3.12 给出了一个 RC4 状态的例子。

RC4 流密码的密钥 s 也是 PRG 的种子，用于将数组 S 初始化为数字 $0 \ldots 255$ 一个伪随机置换排列。初始化是使用下面的**设置算法 (setup algorithm)** 进行的：

输入：字节序列 s

对于 $i = 1, \ldots, 255$：
    令 $S[i] \leftarrow i$
令 $j \leftarrow 0$
对于 $i = 1, \ldots, 255$：
    令 $k \leftarrow s[i \bmod |s|]$ // 从种子中提取一个字节
    令 $j \leftarrow (j + S[i] + k) \bmod 256$
    $\text{swap}(S[i], S[j])$

在循环过程中，索引 i 在数组中线性增长，而索引 j 则跳来跳去。在每次迭代中，指针 i 所指向的内容都会与 j 所指向的内容互换。

一旦数组 S 被初始化，PRG 就可以使用下面的**流生成器 (stream generator)** 一次生成一个字节的伪随机输出：

令 $i \leftarrow 0$，$j \leftarrow 0$
重复：
    令 $i \leftarrow (i + 1) \bmod 256$
    令 $j \leftarrow (j + S[i]) \bmod 256$
    $\text{swap}(S[i], S[j])$
    输出 $S\big[\,(S[i] + S[j]) \bmod 256\,\big]$
直到永远

该程序的运行时间视需要而定。同样地，索引 i 在数组中线性增长，而索引 j 则跳来跳去。交换 $S[i]$ 和 $S[j]$ 会不断地打乱数组 S。

RC4 的加密速度。 RC4 很适合用软件实现。其他流密码，如 Grain 和 Trivium，是为硬件设计的，在软件实现下的性能表现很差。表 **??** 提供了 RC4 和其他一些软件实现的流密码的运行时间对比。现代处理器运行在 64 比特字长上，使得基于 8 比特设计的 RC4 在这些架构上稍显缓慢。

密码	速度[1](MB/s)
RC4	126
SEAL	375
Salsa20	408
Sosemanuk	727

表 3.1: 软件实现的流密码的速度比较（速度越高越好）。

3.9.1 RC4 的安全性

RC4 一度被认为是一个安全的流密码，并被广泛部署在应用程序中。在一些攻击表明它的输出有一定的偏差后，该密码就失宠了。我们下面提出两种攻击，它们都能将 RC4 的输出与随机字符串区分开来。在本小节中，我们用 n 表示数组 S 的大小。对于 RC4，我们有 $n = 256$。

初始 RC4 输出中的偏差。 RC4 的设置算法基于给定的随机种子将数组 S 初始化为 $0 \ldots 255$ 的一个置换。我们先暂且假设 RC4 的设置算法是完美的，它能从所有可能的 256! 个排列组合中产生一个均匀的置换排列。Mantin 和 Shamir 表明，即使假设初始化是完美的，RC4 的输出也是有偏差的。

引理 3.8 (Mantin-Shamir 定理). *假设数组 S 被设置为 $0 \ldots n-1$ 的一个随机排列，并且 i 和 j 都被置为 0，那么 RC4 输出的第二个字节等于 0 的概率为 $2/n$。*

证明思路. 令 z_2 是 RC4 输出的第二个字节。令 P 是 $S[2] = 0$ 和 $S[1] \neq 2$ 成立的事件。关键的观察是，当事件 P 发生时，$z_2 = 0$ 的概率为 1，见图 3.13。然而，当 P 没有发生时，z_2 均匀分布在 $0 \ldots n-1$ 上，因此它等于 0 的概率为 $1/n$。由于 $\Pr[P]$ 约为 $1/n$，我们可以得到以下（近似）结果：

$$\begin{aligned}\Pr[z_2 = 0] &= \Pr[(z_2 = 0) \mid P] \cdot \Pr[P] + \Pr[(z_2 = 0) \mid \neg P] \cdot \Pr[\neg P] \\ &\approx 1 \cdot (1/n) + (1/n) \cdot (1 - 1/n) \approx 2/n \quad \square\end{aligned}$$

该引理表明，RC4 输出的第二个字节为 0 的概率是它本应有的两倍。这就导出了一个简单的 RC4 PRG 区分器。给定一个序列 $x \in \{0, \ldots, 255\}^\ell$，对于 $\ell \geq 2$，如果 x 的第二个字节是 0，区分器输出 0，否则就输出 1。根据引理 3.8，这个区分器的优势约为 $1/n$，对于 RC4 来说就是 0.39%。

Mantin-Shamir 区分器表明 RC4 输出的第二字节是有偏差的。AlFardan 等人推广了这一结论，他们通过测量许多随机密钥上的偏差，表明输出的前 256 个字节中的每一个都有偏差：每个字节的分布都与均匀性相差甚远。这种偏差不像第二字节那样明显，但它仍是不可忽略不计的，足以用来对密码发动攻击。例如，他们表明，给定用 2^{30} 个随机密钥加密的单一明文的对应密文，我们有可能以接近 1 的概率恢复明文的前 128 字节。这种攻击很容易在网络上进行，因为在网络上，一个秘密的 cookie 通常被嵌入到一个消息的前几个字节中。每次浏览器连接到受害者的网络服务器时，这个 cookie 都会用新的密钥重新加密。攻击者可以使用 Javascript 脚本令用户的浏览器反复重连到目标网站，以向攻击者提供发动攻击和暴露 cookie 所需的 2^{30} 个密文。

作为回应，RSA 实验室发布了一项建议，建议放弃 RC4 流生成器输出的前 1024 字节，而只使用第 1025 字节及以后的字节。这可以防御最初的密钥流偏差区分器，但不能防御我们接下来将要讨论的

[1]性能数字是使用 Crypto++ 5.6.0 benchmark 在 1.83 Ghz Intel Core 2 处理器上运行获得的。

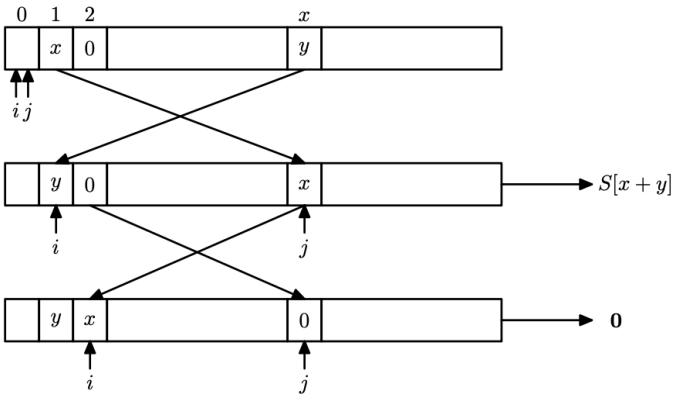


图 3.13: 引理 3.8 的证明

攻击。

RC4 流生成器中的偏差。 假设 RC4 设置算法已经被修改，从而使得上面介绍的攻击无效。Fluhrer 和 McGrew 给出了一个直接针对流生成器的攻击。他们声称字节对 $(0,0)$ 在 RC4 的输出中出现的次数多于它应该在随机序列中出现的次数。这足以将 RC4 的输出与随序列区分开来。

令 $\mathrm{ST}_{\mathrm{RC4}}$ 为 RC4 所有可能的内部状态的集合。由于数组 S 有 n 个可能的设置，i 和 j 各有 n 个可能的设置，所以 $\mathrm{ST}_{\mathrm{RC4}}$ 的大小为 $n! \cdot n^2$。由于在 RC4 中 $n = 256$，$\mathrm{ST}_{\mathrm{RC4}}$ 的大小是非常巨大的，大约为 10^{511}。

引理 3.9 (Fluhrer-McGrew 定理). *假设 RC4 使用一个 $\mathrm{ST}_{\mathrm{RC4}}$ 中的随机状态 T 初始化。令 (z_1, z_2) 是 RC4 在状态 T 下启动时输出的前两个字节。则有:*

$$i \neq n-1 \implies \Pr[(z_1, z_2) = (0,0)] \geq (1/n^2) \cdot \big(1 + (1/n)\big)$$
$$i \neq 0, 1 \implies \Pr[(z_1, z_2) = (0,1)] \geq (1/n^2) \cdot \big(1 + (1/n)\big)$$

我们将一对连续输出 (z_1, z_2) 称为一个**二重字 (digraph)**。在一个真正的随机序列中，所有二重字 (x, y) 出现的概率都应当正好是 $1/n^2$。上面的引理表明，对于 RC4，$(0,0)$ 出现的概率比它应有的值大 $1/n^3$。$(0,1)$ 也是如此。事实上，除了引理 3.9 所述的两个二重字，Fluhrer-McGrew 还确定了其他几个异常的二重字。

该引理导出了一个简单的区分 RC4 的输出和随机序列的区分器 D。如果区分器在给定的序列中发现的 $(0,0)$ 的数量大于随机序列中应有的数量，它就输出 1，否则就输出 0。

输入：序列 $s \in \{0, \ldots, n\}^{\ell}$
输出：0 或 1
令 q 为 x 中出现二重字 $(0,0)$ 的次数
如果 $(q/\ell) - (1/n^2) > 1/(2n^3)$ 则输出 0，否则输出 1

使用定理 24.1，我们可以估计出 D 的优势与输入长度 ℓ 的关系。特别地，区分器 D 能获取下述优势:

$$\ell = 2^{14} \text{ 字节}: \qquad \mathsf{PRGadv}[D, RC4] \geq 2^{-8}$$
$$\ell = 2^{34} \text{ 字节}: \qquad \mathsf{PRGadv}[D, RC4] \geq 0.5$$

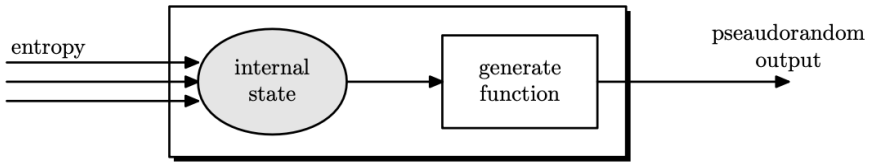


图 3.14: 一个随机数生成器

使用 Fluhrer 和 McGrew 提供的所有异常二重字，我们就可以建立一个区分器，只用 $2^{30.6}$ 字节的输出就能实现 0.8 的优势。

对 RC4 的相关密钥攻击。 Fluhrer、Mantin 和 Shamir 表明，RC4 与相关密钥一起使用时是不安全的。我们将在 9.10 节的攻击 2 中讨论这种攻击及其对 802.11b WiFi 协议的影响。

3.10 在实践中生成随机比特

在密码学中，许多任务都需要随机比特，例如生成密钥和其他被称为 nonce 的短时值。在整本书中，我们假设所有参与方都能获得良好的随机源，否则许多理想的密码学目标就都不可能实现。到目前为止，我们使用 PRG 将一个短的均匀分布的秘密种子拉伸成一个长的伪随机序列。虽然 PRG 是生成伪随机比特序列的一个重要工具，但它只是故事的一部分。

在实践中，随机比特序列通常是使用**随机数生成器 (random number generator, RNG)** 生成的。RNG 和 PRG 一样输出一串随机或伪随机的比特。然而 RNG 有一个额外的接口，用于不断向 RNG 的内部状态添加熵，如图 3.14 所示。其原理是，每当系统有更多的随机熵贡献给 RNG 时，这些熵就被添加到 RNG 的内部状态中。每当有人从 RNG 中读取比特时，这些比特都是用当前的内部状态生成的。

一个典型的例子是 Linux 操作系统中的 RNG，它被实现成一个名为 `/dev/random` 的设备。任何人都可以从该设备中读取到随机比特。你可以在 UNIX shell 中输出 `cat /dev/random` 来试着玩一玩这个工具，你将会看到一串无休止的、看起来很随机的字符。UNIX RNG 从一些硬件来源获得随机熵，包括：

- 键盘事件：按键时间间隔能够提供随机熵；
- 鼠标事件：中断时间和鼠标位置都能够提供随机熵；
- 硬件中断：硬件中断的时间间隔也能提供较高质量的随机熵。

这些随机源能够产生连续的随机流，并会被定期异或到 RNG 的内部状态上。请注意，具体的键盘输入内容不会被用作熵的来源，通常使用的只是按键的时间，这是为了确保用户的输入内容不会通过 RNG 泄露给系统中的其他用户。

高熵的随机生成。 上述熵源产生随机流的速度相对较慢。为了以更快的速度生成真随机比特，英特尔从 2012 年的 Ivy Bridge 处理器系列开始，增加了一个硬件随机数生成器。使用 `RdRand` 指令即可读取这个生成器的输出，它旨在提供一个快速且均匀的随机比特生成器。

为了减少生成器输出中的偏差，原始比特首先通过一个被称为“调节器”的函数，以确保在提供足够熵源作为输入时，输出是一个均匀分布的比特序列。我们在 8.10 节讨论密钥推导问题时会更详细地讨论这个问题。

RdRand 发生器不应该取代其他的熵源，比如上面描述的几个熵源；它只应该作为 RNG 的一个额外的熵源来增强它们。这样一来，如果生成器有缺陷，就不会完全影响到加密应用。

英特尔的方法的一个困难是，随着时间的推移，被采样的硬件元素可能由于硬件故障而停止产生随机流。例如，被采样的比特可能总是 "0"，这会导致高度非随机的输出。为了防止这种情况的发生，RNG 的输出会被不断地用一套固定的统计方法来测试。如果任何一项测试报告为 "非随机"，发生器就会被认为有缺陷。

3.11 一个更广阔的视角：计算上不可区分性

我们对伪随机数发生器 G 的安全性的定义将下面这个直观的想法进行了形式化，即对手不应该能够有效区分 $G(s)$ 和 r，其中 s 是一个随机选出的种子，而 r 是输出空间中的一个随机元素。

这个想法可以很自然地推广到其他场合。假设 P_0 和 P_1 是有限集 $\mathcal{R}$ 上的两个概率分布。我们的目标是定义一个直观的概念，即对手无法有效地区分 P_0 和 P_1。与之前一样，这通过设计一个攻击游戏来完成。对于 $b = 0, 1$，我们记 $x \xleftarrow{R} P_b$ 为根据概率分布 P_b 从集合 $\mathcal{R}$ 中随机选出一个值赋给 x。

攻击游戏 3.3 (区分 P_0 和 P_1). 对于有限集 $\mathcal{R}$ 上的给定概率分布 P_0 和 P_1 以及一个给定对手 $\mathcal{A}$，我们定义两个实验：实验 0 和实验 1。对于 $b = 0, 1$，我们定义：

实验 b:

- 挑战者计算 $x \xleftarrow{R} P_b$，并将 x 发送给对手。
- 给定 x，对手计算并输出一个比特 $\hat{b} \in \{0, 1\}$。

对于 $b = 0, 1$，令 W_b 为对手 $\mathcal{A}$ 在实验 b 中输出 1 的事件。我们定义 $\mathcal{A}$ 相对于 P_0 和 P_1 的**优势**为：

$$\mathrm{Distadv}[\mathcal{A}, P_0, P_1] := \big|\Pr[W_0] - \Pr[W_1]\big|$$

定义 3.4 (计算上不可区分性). *如果 $\mathrm{Distadv}[\mathcal{A}, P_0, P_1]$ 的值对任意有效对手 $\mathcal{A}$ 来说均可忽略不计，则称概率分布 P_0 和 P_1 **在计算上不可区分** **(computationally indistinguishable)**。*

利用该定义，我们可以更简单地重新表述安全 PRG 的定义：如果 P_1 是 $\mathcal{R}$ 上的均匀分布，P_0 是对 $r \in \mathcal{R}$ 的赋值的概率分布：

$$P_0(r) := \frac{|\{s \in \mathcal{S} : G(s) = r\}|}{|\mathcal{S}|}$$

当且仅当 P_0 和 P_1 在计算上不可区分时，定义在 $(\mathcal{S}, \mathcal{R})$ 上的 PRG 是安全的。

与 2.2.5 中讨论的相同，攻击游戏 3.3 可以被改写为一个 "比特猜测" 游戏，其中挑战者不再有两个独立的实验，而是随机选择一个 $b \in \{0, 1\}$，然后与对手 $\mathcal{A}$ 运行实验 b。在这个游戏中，我们将 $\mathcal{A}$ 的比特猜测优势 $\mathrm{Distadv}^*[\mathcal{A}, P_0, P_1]$ 记为 $|\Pr[\hat{b} = b] - 1/2|$。那么 2.2.5 中的推广结论（即式 2.11）也适用于此：

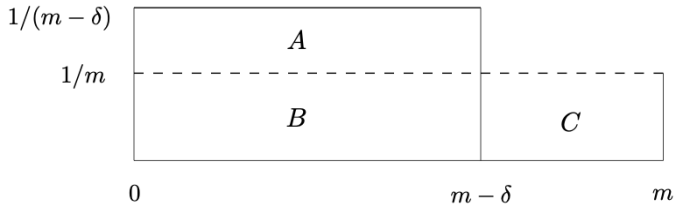
$$\mathrm{Distadv}[\mathcal{A}, P_0, P_1] = 2 \cdot \mathrm{Distadv}^*[\mathcal{A}, P_0, P_1] \tag{3.14}$$

通常情况下，为了证明两个分布在计算上是不可区分的，我们不得不做某些其他的计算上的假设。然而有时两个分布真的极其相似，以至于无论对手有多么强大的计算能力，都无法有效区分它们。为了准确表述这种 "相似性" 的概念，我们下面将引入一个有用的工具，称为**统计距离 (statistical distance)**：

定义 3.5 (统计距离). 假设 P_0 和 P_1 是有限集 $\mathcal{R}$ 上的概率分布，那么它们的**统计距离**定义为：

$$\Delta[P_0, P_1] := \frac{1}{2} \sum_{r \in \mathcal{R}} \left| P_0(r) - P_1(r) \right|$$

例 3.1. 假设 P_0 是 $\{1, \ldots, m\}$ 上的均匀分布，P_1 是 $\{1, \ldots, m-\delta\}$ 上的均匀分布，其中 $\delta \in \{0, \ldots, m-1\}$。我们下面试着计算 $\Delta[P_0, P_1]$。我们固然可以直接使用统计距离的定义来计算 $\Delta[P_0, P_1]$；但是，不妨考虑下面这张关于 P_0 和 P_1 的图：



P_0 和 P_1 的统计距离就是图中区域 A 和区域 C 面积和的一半。此外，由于概率分布的总和为 1，我们必然有：

$$\text{area of } B + \text{area of } A = 1 = \text{area of } B + \text{area of } C$$

因此区域 A 和区域 C 的面积相等。所以：

$$\Delta[P_0, P_1] = \text{area of } A = \text{area of } C = \delta/m$$

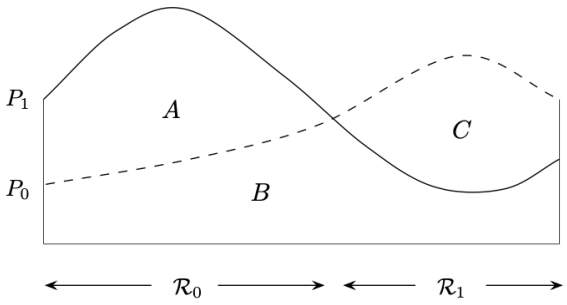
下面的定理使我们能够在计算上不可区分性和统计距离这两个概念之间建立起联系。

定理 3.10. 令 P_0 和 P_1 是有限集 $\mathcal{R}$ 上的两个概率分布，那么我们有：

$$\max_{\mathcal{R}' \subseteq \mathcal{R}} |P_0[\mathcal{R}'] - P_1[\mathcal{R}']| = \Delta[P_0, P_1]$$

其中，最大值在 $\mathcal{R}$ 的所有子集 $\mathcal{R}'$ 上都能取得。

证明. 假设我们把 $\mathcal{R}$ 分成两个互不相干的子集：由使得 $P_0(r) < P_1(r)$ 的 $r \in \mathcal{R}$ 组成的集合 $\mathcal{R}_0$，以及由使得 $P_0(r) \geq P_1(r)$ 的 $r \in \mathcal{R}$ 组成的集合 $\mathcal{R}_1$。考虑下面的 P_0 和 P_1 分布的示意图，其中 $\mathcal{R}_0$ 的元素被放在 $\mathcal{R}_1$ 的元素的左边：



现在，与例 3.1 中一样，我们有：

$$\Delta[P_0, P_1] = \text{area of } A = \text{area of } C$$

注意到，对于 $\mathcal{R}$ 的每个子集 $\mathcal{R}'$，我们都有：

$$P_0[\mathcal{R}'] - P_1[\mathcal{R}'] = \text{area of } C' - \text{area of } A'$$

其中 C' 指位于 $\mathcal{R}'$ 上的 C 的子区域，A' 指位于 $\mathcal{R}'$ 上的 A 的子区域。由此可知，当 $\mathcal{R}' = \mathcal{R}_0$ 或 $\mathcal{R}' = \mathcal{R}_1$ 时，$|P_0[\mathcal{R}'] - P_1[\mathcal{R}']|$ 取得最大值，此时最大值就等于 $\Delta[P_0, P_1]$。 □

与计算上不可分性的联系如下：

定理 3.11. *假设 P_0 和 P_1 是有限集 $\mathcal{R}$ 上的概率分布，那么对于任意对手 $\mathcal{A}$，我们都有：*

$$\text{Distadv}[\mathcal{A}, P_0, P_1] \leq \Delta[P_0, P_1]$$

证明. 考虑一个如攻击游戏 3.3 中那样试图区分 P_0 和 P_1 的对手 $\mathcal{A}$。

首先，我们考虑 $\mathcal{A}$ 是确定性算法的情况。在这种情况下，$\mathcal{A}$ 的输出是一个 $r \in \mathcal{R}$ 的函数 $f(r)$，它是由挑战者发送给 $\mathcal{A}$ 的。令 $\mathcal{R}' := \{r \in \mathcal{R} : f(r) = 1\}$。如果 W_0 和 W_1 是攻击游戏 3.3 中定义的两个事件，那么对于 $b = 0, 1$，我们有：

$$\Pr[W_b] = P_b[\mathcal{R}']$$

根据定理 3.10，我们有：

$$\text{Distadv}[\mathcal{A}, P_0, P_1] = |P_0[\mathcal{R}'] - P_1[\mathcal{R}']| \leq \Delta[P_0, P_1]$$

我们下面考虑 $\mathcal{A}$ 是概率性算法的情况。我们可以认为 $\mathcal{A}$ 接受一个辅助输入 t，代表它的随机选择。我们认为 t 是从某个有限集 $\mathcal{T}$ 中均匀随机选出的。因此，$\mathcal{A}$ 的输出是挑战者交给他的值 $r \in \mathcal{R}$ 和代表其随机选择的值 $t \in \mathcal{T}$ 的函数 $g(r, t)$。对于一个给定的 $t \in \mathcal{T}$，令 $\mathcal{R}'_t := \{r \in \mathcal{R} : g(r, t) = 1\}$。然后，对 t 的随机选择进行平均化，我们有：

$$\Pr[W_b] = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} P_b[\mathcal{R}'_t]$$

由此可得：

$$\begin{aligned}
\text{Distadv}[\mathcal{A}, P_0, P_1] &= |P_0[\mathcal{R}'] - P_1[\mathcal{R}']| \\
&= \frac{1}{|\mathcal{T}|} \left| \sum_{t \in \mathcal{T}} (P_0[\mathcal{R}'_t] - P_1[\mathcal{R}'_t]) \right| \\
&\leq \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} |P_0[\mathcal{R}'_t] - P_1[\mathcal{R}'_t]| \\
&\leq \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \Delta[P_0, P_1] \\
&= \Delta[P_0, P_1] \quad \square
\end{aligned}$$

作为该定理的一个结论，我们可以看到，如果 $\Delta[P_0, P_1]$ 可忽略不计，那么 P_0 和 P_1 是计算上不可区分的。

我们还可以将两个随机变量之间的统计距离定义为它们相应分布之间的统计距离。也就是说，如果 X 和 Y 是在一个有限集 $\mathcal{R}$ 中取值的随机变量，那么它们的**统计距离**是：

$$\Delta[\mathsf{X}, \mathsf{Y}] := \frac{1}{2} \sum_{r \in \mathcal{R}} |\Pr[\mathsf{X} = r] - Pr[\mathsf{Y} = r]|$$

在这种情况下，定理 3.10 表明：

$$\max_{\mathcal{R}' \subseteq \mathcal{R}} |\Pr[\mathsf{X} \in \mathcal{R}'] - \Pr[\mathsf{Y} \in \mathcal{R}']| = \Delta[\mathsf{X}, \mathsf{Y}]$$

其中，最大值在 $\mathcal{R}$ 的所有子集 $\mathcal{R}'$ 上都能取得。

类似地，我们还可以对随机变量而非分布来定义区分优势。使用随机变量的好处是，我们可以更方便地处理彼此相关的分布，正如下面的定理所例证的那样。

定理 3.12. *如果 $\mathcal{S}$ 和 $\mathcal{T}$ 是有限集，X 和 Y 是在 $\mathcal{S}$ 上取值的随机变量，并且 $f : \mathcal{S} \to \mathcal{T}$ 是一个函数，那么 $\Delta[f(\mathsf{X}), f(\mathsf{Y})] \leq \Delta[\mathsf{X}, \mathsf{Y}]$。*

证明. 对于某个 $\mathcal{T}' \subseteq \mathcal{T}$，我们有：

$$\begin{aligned}\Delta[f(\mathsf{X}), f(\mathsf{Y})] &= |\Pr[f(\mathsf{X}) \in \mathcal{T}'] - \Pr[f(\mathsf{Y}) \in \mathcal{T}']| \quad \text{（根据定理 3.10）} \\ &= |\Pr[\mathsf{X} \in f^{-1}(\mathcal{T}')] - \Pr[\mathsf{Y} \in f^{-1}(\mathcal{T}')]| \\ &\leq \Delta[\mathsf{X}, \mathsf{Y}] \quad \text{（根据定理 3.10）} \quad \square\end{aligned}$$

例 3.2. 令 X 均匀分布在集合 $\{0, \ldots, m-1\}$ 上，Y 均匀分布在集合 $\{0, \ldots, N-1\}$ 上，且 $N \geq m$。令 $f(t) := t \bmod m$。我们想计算 X 和 $f(\mathsf{Y})$ 之间的统计距离的上界。我们可以这样做。令 $N = qm - r$，其中 $0 \leq r < m$，因此 $q = \lceil N/m \rceil$。同时，令 Z 均匀分布在集合 $\{0, \ldots, qm-1\}$ 上。那么 $f(\mathsf{Z})$ 就均匀分布在集合 $\{0, \ldots, m-1\}$ 上，这是因为 $\{0, \ldots, m-1\}$ 中的每个元素在函数 f 下都有相同个数的原像（即 q 个），这些原像都落在集合 $\{0, \ldots, qm-1\}$ 中。由于统计距离只取决于随机变量的分布，根据定理 3.12，我们有：

$$\Delta[\mathsf{X}, f(\mathsf{Y})] = \Delta[f(\mathsf{Z}), f(\mathsf{Y})] \leq \Delta[\mathsf{Z}, \mathsf{Y}]$$

正如我们在例 3.1 中所看到的：

$$\Delta[\mathsf{Z}, \mathsf{Y}] = \frac{r}{qm} < \frac{1}{q} \leq \frac{m}{N}$$

因此有：

$$\Delta[\mathsf{X}, f(\mathsf{Y})] < \frac{m}{N}$$

例 3.3. 现在我们想要生成一个给定区间 $\{0, \ldots, m-1\}$ 上的伪随机数。假设我们有一个 PRG G，它可以输出 L 比特的序列。当然，一个 L 比特的序列可以被看作是 $\{0, \ldots, N-1\}$ 中的一个数，其中 $N := 2^L$。让我们假设 $N \geq m$。

为了生成一个区间 $\{0, \ldots, m-1\}$ 上的伪随机数，我们可以把 G 的输出看作是 $\{0, \ldots, N-1\}$ 中的一个数，并将其模 m 后输出。我们下面将表明，只要 G 是安全的，且 m/N 可忽略不计，上述方法所产生的数和从区间 $\{0, \ldots, m-1\}$ 中随机挑选的真随机数在计算上不可区分。

为此，令 P_0 为 G 输出并模 m 后的分布，P_1 为 $\{0, \ldots, m-1\}$ 上的均匀分布，令 $\mathcal{A}$ 是一个试图区分 P_0 和 P_1 的对手，就像在攻击游戏 3.3 中的那样。

令游戏 0 为攻击游戏 3.3 中的实验 0，在这个实验中，$\mathcal{A}$ 被赋予了一个按照 P_0 分布的随机样本，记 W_0 是 $\mathcal{A}$ 在游戏 0 中输出 1 的事件。

现在，定义游戏 1 与游戏 0 基本相同，只是我们用一个从区间 $\{0, \ldots, N-1\}$ 中随机选出的真随机数代替 G 的输出。记 W_1 为 $\mathcal{A}$ 在游戏 1 中输出 1 的事件。我们很容易构建出一个有效对手 $\mathcal{B}$，他可以像攻击游戏 3.1 中那样攻击 G，并使得：

$$\mathrm{PRGadv}[\mathcal{B}, G] = |\Pr[W_0] - \Pr[W_1]|$$

思路是 $\mathcal{B}$ 获取到他的挑战值并模 m，然后将这个值交给 $\mathcal{A}$，然后原样输出 $\mathcal{A}$ 输出的任何东西。

最后，我们定义游戏 2 为攻击游戏 3.3 中的实验 1，在这个实验中，$\mathcal{A}$ 被赋予了一个按照 P_1 分布的随机样本，也就是 $\{0,\ldots,m-1\}$ 上的均匀分布。记 W_2 为 $\mathcal{A}$ 在游戏 2 中输出 1 的事件。如果 P 是游戏 1 中交给 $\mathcal{A}$ 的值的分布，那么根据定理 3.11，我们就有 $|\Pr[W_1]-\Pr[W_2]| \leq \Delta[P,P_1]$；此外，根据例 3.2，我们还有 $\Delta[P,P_1] \leq m/N$。

将以上结论综合起来，可以得到：

$$\begin{aligned}\mathsf{Distadv}[\mathcal{A},P_0,P_1] &= |\Pr[W_0]-\Pr[W_2]| \leq |\Pr[W_0]-\Pr[W_1]| + |\Pr[W_1]-\Pr[W_2]| \\ &\leq \mathrm{PRGadv}[\mathcal{B},G] + \frac{m}{N}\end{aligned}$$

而根据假设，该值可忽略不计。

3.11.1 数学细节

和之前一样，我们下面会详述相关的数学细节，以便从渐进复杂性理论的角度解释本节的定义和结论。

在定义计算上不可区分性（定义 3.4）时，我们应该考虑两个概率分布族 $P_0=\{P_{0,\lambda}\}_\lambda$ 和 $P_1=\{P_{1,\lambda}\}_\lambda$，它们都由安全参数 λ 索引。对于每个 λ，分布 $P_{0,\lambda}$ 和 $P_{1,\lambda}$ 都应该在有限比特序列集合 $\mathcal{R}_\lambda$ 中取值，其中 $\mathcal{R}_\lambda$ 中的序列长度以 λ 的多项式为界。在攻击游戏 3.3 中，安全参数 λ 是挑战者和对手的输入，而在实验 b 中，挑战者产生一个根据 $P_{b,\lambda}$ 分布的样本。优势应当被正确地表记为 $\mathsf{Distadv}[\mathcal{A},P_0,P_1](\lambda)$，它是 λ 的一个函数，而计算上不可区分性意味着该函数可以忽略不计。

在某些情况下，引入一个概率生成的系统参数可能是很自然的；然而，从技术角度来看，这不是必要的，因为这样的系统参数可以被纳入到分布 $P_{0,\lambda}$ 和 $P_{1,\lambda}$ 中。我们还可以要求 $P_{0,\lambda}$ 和 $P_{1,\lambda}$ 是可有效采样的；然而，为了保持定义的简单性，我们不会强制要求这样做。

统计距离的定义（定义 3.5）从非渐进的角度来看是完全合理的，不需要任何修改或阐述。如前所述，定理 3.10 对特定的分布 P_0 和 P_1 成立。定理 3.11 可以渐进地看作是说，对于所有分布族 $P_0=\{P_{0,\lambda}\}_\lambda$ 和 $P_1=\{P_{1,\lambda}\}_\lambda$，对于任意对手（甚至是计算上无界的对手），以及对于所有 λ，我们都有：

$$\mathsf{Distadv}[\mathcal{A},P_0,P_1](\lambda) \leq \Delta[P_{0,\lambda},P_{1,\lambda}]$$

3.12 一个有趣的应用：抛掷硬币与比特承诺

Alice 和 Bob 要出去约会了。Alice 想看某一部电影，而 Bob 想看另一部。他们决定抛一枚硬币来随机选择电影。如果抛硬币的结果是正面，他们就去看 Alice 选择的电影，否则就去看 Bob 选择的电影。当 Alice 和 Bob 离得很近的时候，这很容易操作：他们中的一个人，比如说 Bob，抛出一枚硬币，他们都能验证这个结果。但当他们相距甚远，并且在电话中交谈时，这就比较难了。Bob 可以在他那边掷硬币，然后告诉 Alice 结果，但是 Alice 却没有理由相信这个结果。Bob 可以简单地声称抛硬币的结果是反面，而 Alice 则没有办法证实这一点。这不是一个开始约会的好方法。

解决这个问题的一个简单方法是利用一种叫做**比特承诺 (bit commitment)** 的密码学原语。它能够让 Bob 承诺一个他选择的比特 $b \in \{0,1\}$。稍后，Bob 可以打开承诺，让 Alice 相信 b 是 Bob 所承诺的值。承诺一个比特 b 的结果是一个**承诺字符串 (commitment string)** c，它和一个**打开字符串**

(opening string) 被一起发送给 Alice，由 Bob 在以后打开承诺时使用。如果一个承诺方案满足以下两个特性，它就是安全的：

- **隐藏 (hiding)**：承诺字符串 c 不会透露关于承诺比特 b 的任何信息。更确切地说，当承诺比特为 0 时 c 的分布与承诺比特为 1 时 c 的分布是不可区分的。在我们提出的比特承诺方案中，隐藏属性取决于特定的 PRG G 的安全性。
- **绑定 (binding)**：令 c 是 Bob 输出的承诺字符串。如果 Bob 可以将该承诺打开为某个 $b \in \{0,1\}$，那么他就不能将其打开为 $\bar{b}$。这就保证了一旦 Bob 承诺了一个比特 b，他就只可以把它作为 b 打开，而不能是其他的值。在我们提出的承诺方案中，绑定属性无条件成立。

抛掷硬币。 使用一个承诺方案，Alice 和 Bob 就可以生成一个随机比特 $b \in \{0,1\}$，这样，假设协议成功终止，任何一方都不能使结果偏向他们的倾向。这样的协议被称为**抛掷硬币协议 (coin flipping protocols)**。结果比特 b 将决定他们去看什么电影。

Alice 和 Bob 使用下面这个简单的抛掷硬币协议：

步骤 1：Bob 随机选择一个比特 $b_0 \xleftarrow{R} \{0,1\}$。

Alice 和 Bob 执行承诺协议。

通过该协议，Alice 获得对 b_0 的承诺 c，Bob 获得一个打开字符串 s。

步骤 2：Alice 随机选择一个比特 $b_1 \xleftarrow{R} \{0,1\}$，并将其明文发送给 Bob。

步骤 3：Bob 通过向 Alice 揭露 b_0 和 s 来打开承诺。

Alice 验证 c 是否是对 b_0 的承诺，如果验证失败则终止。

输出：结果比特 $b := b_0 \oplus b_1$。

我们声称，如果协议成功终止，并且一方诚实地遵守协议，那么另一方就不能使结果朝他的倾向偏移。根据隐藏属性，Alice 在步骤 1 结束时不会得到任何关于 b_0 的信息，因此她对 b_1 的选择与 b_0 的值无关。根据绑定属性，Bob 只能在步骤 3 中对他在步骤 1 中选择的 b_0 打开承诺 c。而由于他在 Alice 选择 b_1 之前就选择了 b_0，所以 Bob 对 b_0 的选择与 b_1 无关。我们可以得出结论，输出比特 b 是两个独立比特的异或。因此，如果一方诚实地遵守协议，另一方就不可能使产生的比特有所偏向。

这个协议的一个问题是，Bob 在步骤 2 结束时，在 Alice 获取比特之前就知道了生成的比特。原则上，如果结果不是 Bob 想要的，他也可以在步骤 2 结束时中止协议，并试图重新启动协议，希望下一次运行能得到他想要的结果。更复杂的抛掷硬币协议能够避免这个问题，但代价是需要更多回合的交互。

来自安全 PRG 的比特承诺。 剩下的工作就是构建一个让 Bob 承诺他的比特 $b_0 \in \{0,1\}$ 的安全比特承诺方案。我们使用 Naor 提出的一个优雅的构造来完成这个工作。

令 $G : \mathcal{S} \to \mathcal{R}$ 是一个安全的 PRG，其中 $|\mathcal{R}| \geq |\mathcal{S}|^3$ 和 $\mathcal{R} = \{0,1\}^n$ 对某个 n 成立。为了承诺比特 b_0，Alice 和 Bob 参与以下协议：

Bob 承诺比特 $b0 \in \{0,1\}$：

步骤 1：Alice 随机选择一个 $r \in \mathcal{R}$，并将 r 发送给 Bob。

步骤 2：Bob 随机选择一个 $s \in \mathcal{S}$ 并计算 $c \leftarrow \mathrm{com}(s, r, b_0)$，其中 $\mathrm{com}(s, r, b_0)$ 是以下函数：

$$c = \mathrm{com}(s, r, b_0) := \begin{cases} G(s) & b_0 = 0 \\ G(s) \oplus r & b_0 = 1 \end{cases}$$

Bob 输出 c 作为承诺字符串，并将 s 作为打开字符串。

当到了打开承诺的时候，Bob 向 Alice 发送 (b_0, s)。如果 $c = \mathrm{com}(s, r, b_0)$，Alice 就接受打开值，否则就拒绝它。

隐藏属性直接来自 PRG 的安全性：因为输出 $G(s)$ 与 $\mathcal{R}$ 上的均匀随机字符串在计算上是不可区分的，所以 $G(s) \oplus r$ 与 $\mathcal{R}$ 上的均匀随机字符串在计算上也是不可区分的。因此，无论是 $b_0 = 0$ 还是 $b_0 = 1$，承诺字符串 c 与 $\mathcal{R}$ 上的均匀字符串在计算上是不可区分的，与要求一样。

只要 $1/|\mathcal{S}|$ 可忽略不计，绑定属性就无条件成立。Bob 可以将一个承诺 $c \in \mathcal{R}$ 打开为 0 和 1 的唯一办法是找到两个种子 $s_0, s_1 \in \mathcal{S}$ 使得 $c = G(s_0) = G(s_1) \oplus r$，这意味着 $G(s_0) \oplus G(s_1) = r$。如果存在种子 $s_0, s_1 \in \mathcal{S}$ 使得 $G(s_0) \oplus G(s_1) = r$，我们就称这样的 $r \in \mathcal{R}$ 是 “坏的”。种子对 (s_0, s_1) 的数量是 $|\mathcal{S}|^2$，因此坏的 r 的数量最多也是 $|\mathcal{S}|^2$。由此可见，Alice 选到坏的 r 的概率最多为 $|\mathcal{S}|^2/|\mathcal{R}| < |\mathcal{S}|^2/|\mathcal{S}|^3 = 1/|\mathcal{S}|$，该值可忽略不计。因此，Bob 能把承诺 c 打开为 0 和 1 的概率是可忽略不计的。

这样，我们就完成了对比特承诺方案的介绍。我们将在 8.12 节看到一个更有效的承诺方案以及更多与承诺有关的应用，但在那之前，我们还需要引入更多的密码学工具。

3.13 笔记

对文献的引用有待补充。

3.14 练习

第四章　分组密码

接着上一章，本章继续讨论抵御窃听者并保护隐私的话题。在本章中，我们将研究另外一种密码，被称为**分组密码 (block cipher)**。除此之外，我们还将会考察**伪随机函数 (pseudo-random function)**的相关概念。

分组密码是实用密码学的“老黄牛”：它们不仅可以用来构建流密码，还可以用来构建具有更强安全属性的密码（正如我们将在第五章中所探讨的那样），以及许多其他的密码学原语。

4.1　分组密码：基本定义与性质

从功能上讲，**分组密码**是一种确定性密码 $\mathcal{E} = (E, D)$，其消息空间和密码空间是同一（有限）集 $\mathcal{X}$。如果 $\mathcal{E}$ 的密钥空间是 $\mathcal{K}$，我们就称 $\mathcal{E}$ 是一个**定义在** $(\mathcal{K}, \mathcal{X})$ **上**的分组密码。我们称元素 $x \in \mathcal{X}$ 为一个**数据分组 (data block)**，并称 $\mathcal{X}$ 为 $\mathcal{E}$ 的**数据分组空间**。

对于每个固定的密钥 $k \in \mathcal{K}$，我们可以定义函数 $f_k := E(k, \cdot)$。也就是说，$f_k : \mathcal{X} \to \mathcal{X}$ 可以把 $x \in \mathcal{X}$ 映射到 $E(k, x) \in \mathcal{X}$ 上。密码的正确性要求，对于任意固定密钥 k，函数 f_k 都是一一对应的，并且由于 $\mathcal{X}$ 是有限集，f_k 也必须映射到有限集 $\mathcal{X}$ 上。因此 f_k 本质上是有限集 $\mathcal{X}$ 上的一个置换，而 $D(k, \cdot)$ 是逆置换 f_k^{-1}。

尽管从语法上讲，分组密码只是一种特殊的密码，但我们期望分组密码具有的安全属性实际上要比语义安全性强得多：对于一个随机选择的密钥 k，就所有的实际情况而言，置换 $E(k, \cdot)$ 应该“看起来”像一个随机的置换。我们下面会更精确地定义这一概念。

一个非常重要且流行的分组密码是高级加密标准 (Advanced Encryption Standard, AES)。我们将在后面详细地研究 AES 的内部设计，但现在我们先给出一个非常高层次的描述。AES 的密钥通常是 128 比特的序列（但也可以使用更长的密钥，比如 192 比特或 256 比特）。AES 数据分组是 128 比特的序列，见图 4.1。AES 的设计是相当高效的：使用一台典型的消费级计算机进行一次 AES 加（解）密仅需几百个时钟周期。

分组密码的安全定义被表述为一种“黑盒测试”。大致思路是这样的：给一个有效对手一个“黑盒子”，盒子里是 $\mathcal{X}$ 上的一个置换 f，它来自以下两个随机过程中的一个：

- 对于随机选出的密钥 k，$f = E(k, \cdot)$，或者
- f 是从 $\mathcal{X}$ 的*所有*置换中随机均匀选出的一个真随机置换。

对手看不到盒子的内部，但他可以用提问的方式来“探测”它：他可以给盒子一个值 $x \in \mathcal{X}$ 并得到一个 $y := f(x) \in \mathcal{X}$。我们允许对手多次提问，而且我们允许他以任何方式选择问题；特别地，这些问题甚至可以以某种巧妙的方式依赖于盒子对于之前的某个或某些问题的回答。安全性意味着，对手无论如何

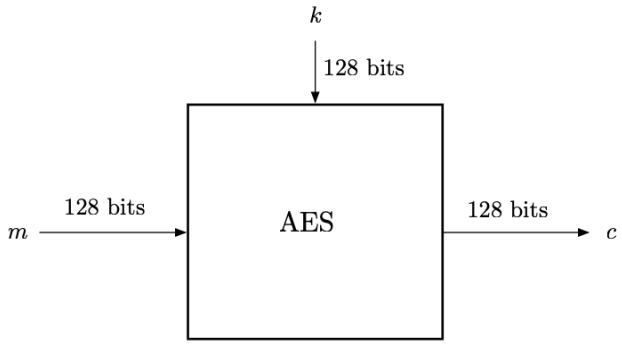


图 4.1: 分组密码 AES

也无法得知盒子里是哪种类型的函数——是随机密钥控制的分组密码，还是一个真的随机置换。换句话说，一个安全的分组密码应该与一个随机置换**在计算上不可区分**。

为了更正式地定义这一概念，我们首先引入一些表记符号。我们用：

$$\mathrm{Perms}[\mathcal{X}]$$

表示 $\mathcal{X}$ 上*所有*置换的集合。需要注意，这是一个非常大的集合：

$$|\mathrm{Perms}[\mathcal{X}]| = |\mathcal{X}|!$$

对于 AES 来说，$|\mathcal{X}| = 2^{128}$，置换的数量约为：

$$\mathrm{Perms}[\mathcal{X}] \approx 2^{2^{135}}$$

而 128 比特 AES 定义的置换的数量最多为 2^{128}。

和之前一样，为了定义安全性，我们会引入一个攻击游戏。就如定义 PRG 的攻击游戏一样，这个攻击游戏也包含两个独立的实验。在这两个实验中，对手将遵循相同的协议，即它会向挑战者提交一连串的查询 $x_1, x_2, \ldots$；挑战者则用 $f(x_i)$ 回应查询 x_i。在第一个实验中，$f = E(k, \cdot)$，其中 $k \in \mathcal{K}$ 是随机选出的一个元素；而在第二个实验中，f 是从 $\mathrm{Perms}[\mathcal{X}]$ 中随机选出的一个置换。在每个实验中，挑战者都只能使用同一 f 来回应所有来自对手的查询。当对手决定终止对挑战者的质询，它就会输出一个比特。

攻击游戏 4.1 (分组密码). 对于一个定义在 $(\mathcal{K}, \mathcal{X})$ 上的给定分组密码 (E, D) 和一个给定对手 $\mathcal{A}$，我们定义两个实验：实验 0 和实验 1。对于 $b = 0, 1$，我们定义：
实验 b:

- 挑战者按如下方式选择 $f \in \mathrm{Perms}[\mathcal{X}]$：

  如果 $b = 0$：$k \xleftarrow{R} \mathcal{K}$，令 $f \leftarrow E(k, \cdot)$；
  如果 $b = 1$：令 $f \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$。

- 对手向挑战者发起一系列查询。
  对于 $i = 1, 2, \ldots$，第 i 个查询是一个数据分组 $x_i \in \mathcal{X}$。
  挑战者计算 $y_i \leftarrow f(x_i) \in \mathcal{X}$，并将 y_i 交给对手。

- 对手计算并输出一个比特 $\hat{b} \in \{0,1\}$。

对于 $b = 0, 1$，令 W_b 为 $\mathcal{A}$ 在实验 b 中输出 1 的事件。我们将 $\mathcal{A}$ 对于 $\mathcal{E}$ 的**优势**定义为：

$$\mathrm{BCadv}[\mathcal{A}, \mathcal{E}] := \big|\Pr[W_0] - \Pr[W_1]\big|$$

最后，如果 $\mathcal{A}$ 最多发出 Q 次查询，我们就称 $\mathcal{A}$ 就是一个 Q **次查询 BC 对手**。

图 4.2 展示了攻击游戏 4.1。

定义 4.1 (安全的分组密码). 如果对于所有有效对手 $\mathcal{A}$，$\mathrm{BCadv}[\mathcal{A}, \mathcal{E}]$ 的值都可忽略不计，那么分组密码 $\mathcal{E}$ 就是安全的。

我们强调，挑战者在攻击游戏 4.1 中的查询是允许自适应的；也就是说，对手不需要事先选择所有的查询；相反，它允许对手以某种巧妙的方式根据挑战者之前的应答构造接下来的查询（见练习 4.6）。

正如 2.2.5 小节所讨论的，攻击游戏 4.1 也可以被重构为一个 “比特猜测” 游戏，此时挑战者不再有两个独立的实验，而是随机选择 $b \in \{0,1\}$，然后针对对手 $\mathcal{A}$ 运行实验 b。在这个游戏中，我们记 $\mathcal{A}$ 的比特猜测优势 $\mathrm{BCadv}^*[\mathcal{A}, \mathcal{E}]$ 为 $|\Pr[\hat{b} = b] - 1/2|$。2.2.5 小节的推广结论（即式 2.11）在此依旧适用：

$$\mathrm{BCadv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \mathrm{BCadv}^*[\mathcal{A}, \mathcal{E}] \tag{4.1}$$

4.1.1 安全性的一些含义

令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{X})$ 上的分组密码。为了进一步了解安全性的含义，我们下面讨论几个简单的结论。简单起见，我们假设 $|\mathcal{X}|$ 是大（超多项式）的。

4.1.1.1 安全的分组密码是不可预测的

我们下面说明，如果密码 $\mathcal{E}$ 在定义 4.1 的意义上是安全的，那么它一定是不可预测的，这意味着，每个有效对手赢得下面的预测游戏的概率都可忽略不计。在这个游戏中，挑战者随机选择一个密钥 k，而对手提交一连串的查询 $x_1, \ldots, x_Q$；对于对手的第 i 次查询，挑战者以 $E(k, x_i)$ 应答。这些查询是自适应的，也就是说对手的每个查询都可能取决于挑战者之前的应答。最后，挑战者会输出一对值 (x_{Q+1}, y)，其中 $x_{Q+1} \notin \{x_1, \ldots, x_Q\}$。如果有 $y = E(k, x_{Q+1})$，我们就说对手赢得了这场游戏。

为了证明这一结论，我们假设 $\mathcal{E}$ 不是不可预测的，这意味着存在一个有效对手 $\mathcal{A}$，它能以不可忽略不计的概率 p 赢得上述预测游戏，于是那么我们可以利用 $\mathcal{A}$ 来打破 $\mathcal{E}$ 在定义 4.1 意义上的安全性。为此，我们可以构造一个对手 $\mathcal{B}$，它一边进行攻击游戏 4.1，一边在上述预测游戏中扮演 $\mathcal{A}$ 的挑战者的角色。每当 $\mathcal{A}$ 发起查询 x_i 时，对手 $\mathcal{B}$ 就将 x_i 转发给自己在攻击游戏 4.1 中的挑战者，然后得到一个应答 y_i，并将其传回给 $\mathcal{A}$。最后，当 $\mathcal{A}$ 输出 (x_{Q+1}, y) 时，对手 $\mathcal{B}$ 将 x_{Q+1} 转交给自己的挑战者，得到 y_{Q+1}。如果 $y = y_{Q+1}$，$\mathcal{B}$ 就输出 1，否则就输出 0。

一方面，如果 $\mathcal{B}$ 的挑战者正在进行实验 0，那么 $\mathcal{B}$ 会以概率 p 输出 1。另一方面，如果 $\mathcal{B}$ 的挑战者正在进行实验 1，那么 $\mathcal{B}$ 以可忽略不计的概率 ϵ 输出 1（这是因为我们假设 $|\mathcal{X}|$ 是超多项式的）。这意味着 $\mathcal{B}$ 在攻击游戏 4.1 中的优势是 $|p - \epsilon|$，而这个值是不可忽略不计的。

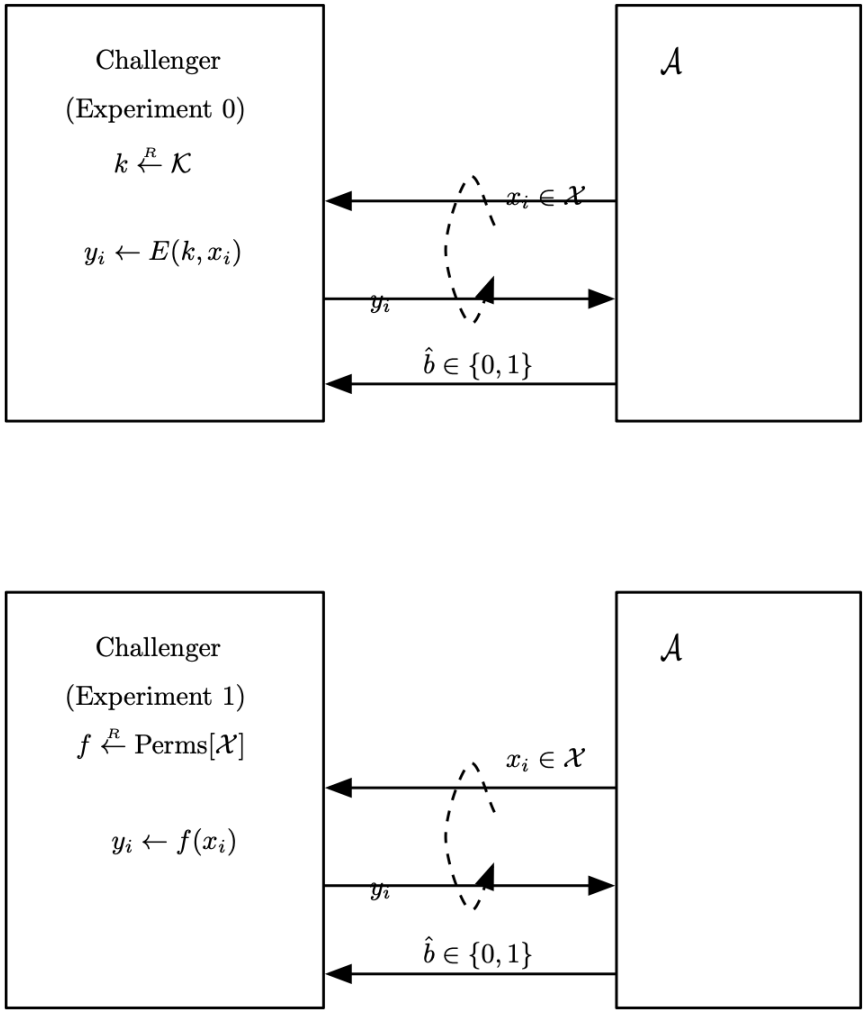

Challenger
(Experiment 0)
$k \xleftarrow{R} \mathcal{K}$
$y_i \leftarrow E(k, x_i)$
$\mathcal{A}$
$x_i \in \mathcal{X}$
y_i
$\hat{b} \in \{0,1\}$
Challenger
(Experiment 1)
$f \xleftarrow{R} \text{Perms}[\mathcal{X}]$
$y_i \leftarrow f(x_i)$
$\mathcal{A}$
$x_i \in \mathcal{X}$
y_i
$\hat{b} \in \{0,1\}$

图 4.2: 攻击游戏 4.1

4.1.1.2 不可预测性意味着对密钥恢复攻击的安全性

我们下面说明，如果 $\mathcal{E}$ 是不可预测的，那么它对*密钥恢复攻击*是安全的，这意味着每个有效对手赢得下面的*密钥恢复游戏*的概率都可忽略不计。在这个游戏中，对手与挑战者的交互方式与上面的预测游戏完全一样，区别只是在最后，对手需要输出一个候选密钥 $k \in \mathcal{K}$，如果 $k = k$，我们就说对手赢得了游戏。

为了证明这一结论，我们假设 $\mathcal{E}$ 对密钥恢复攻击不安全，这意味着存在一个有效对手 $\mathcal{A}$，它能以不可忽略不计的概率 p 赢得密钥恢复游戏。然后我们可以使用 $\mathcal{A}$ 来建立一个有效对手 $\mathcal{B}$，它能以至少 p 的概率赢得上面的预测游戏。对手 $\mathcal{B}$ 只需要运行 $\mathcal{A}$ 的攻击，然后在 $\mathcal{A}$ 输出 k 的时侯任意选择一个 $x_{Q+1} \notin \{x_1, \ldots, x_Q\}$，计算 $y \leftarrow E(k, x_{Q+1})$ 并输出 (x_{Q+1}, y)。

很容易看出，如果 $\mathcal{A}$ 赢得了密钥恢复游戏，$\mathcal{B}$ 就赢得了预测游戏。

4.1.1.3 密钥空间大小和穷举搜索攻击

结合上面的两个结论，我们可以知道：如果 $\mathcal{E}$ 是一个安全的分组密码，那么它对密钥恢复攻击也一定是安全的。此外，如果 $\mathcal{E}$ 对密钥恢复攻击是安全的，那么 $|\mathcal{K}|$ 一定是大的。

有一种方法可以论证这个结论，如下所述。任何一个对手只要从密钥空间 $\mathcal{K}$ 中随机选取一个 k，就能以 $1/|\mathcal{K}|$ 的概率赢得密钥恢复游戏。而如果 $|\mathcal{K}|$ 不是超多项式的，$1/|\mathcal{K}|$ 就不可忽略不计。因此，当 $|\mathcal{K}|$ 不是超多项式的时候，这个简单的密钥猜测对手就能以不可忽略不计的概率赢得密钥恢复游戏。

我们也可以通过另一种称为*穷举搜索攻击*的方式来用运行时间换取成功概率。在这种攻击中，我们的对手在密钥恢复游戏中进行一些任意的查询 $x_1, \ldots, x_Q$，并获得应答 $y_1, \ldots, y_Q$。我们可以认为——至少从启发式的角度来看——假设 $|\mathcal{X}| \geq |\mathcal{K}|$，并且 $|\mathcal{X}|$ 是超多项式的，对于相当小的 Q 值（事实上 $Q = 2$），仅有一个密钥 k 能够以与 1 相差可不略不计的概率使得：

$$y_i = E(k, x_i), \quad i = 1, \ldots, Q \tag{4.2}$$

因此，我们的对手只需尝试所有可能的密钥，必然能够找到一个满足式 4.2 的密钥 k。如果只有一个符合条件的密钥，那么对手找到的密钥就会是挑战者选择的密钥，而对手将赢得游戏。因此，对手能以几乎为 1 的概率赢得密钥恢复游戏，但是他的运行时间是 $|\mathcal{K}|$ 的线性函数。

这种时间/优势的权衡很容易被推广。事实上，考虑一个随机选择 t 个密钥的对手，它测试每个密钥是否满足式 4.2。该对手的运行时间是 t 的线性函数，并且它能以 $\approx t/|\mathcal{K}|$ 的概率赢得密钥恢复游戏。

我们将在 4.2.2 小节中描述一些现实世界中的穷举搜索攻击。我们将在 4.7.2 小节中对穷举搜索进行详细的处理，特别是，我们届时将证明上面所使用的启发式假设，即最多仅有一个密钥能够以高概率满足式 4.2。

因此，如果想要保证一个分组密码是安全的，就必须赋予它一个大的密钥空间，目的是使其能够抵抗密钥恢复攻击。

4.1.2 随机置换的有效实现

请注意，在攻击游戏 4.1 的实验 1 中，挑战者的协议并不是很高效，因为他需要构造一个*极大的*随机对象。事实上，仅仅写下 $\mathrm{Perms}[\mathcal{X}]$ 中的一个元素就需要大约 $|\mathcal{X}| \log_2 |\mathcal{X}|$ 个比特。对于 AES 来说，$|\mathcal{X}| = 2^{128}$，这意味着大约需要 10^{40} 个比特！

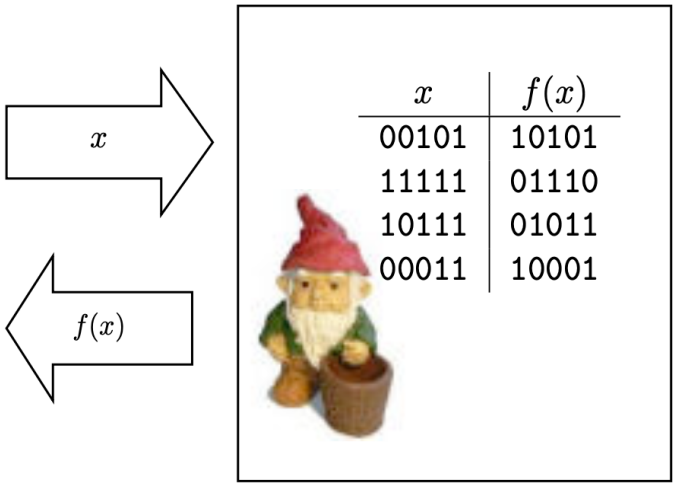


图 4.3: 一个忠实的侏儒实现了随机置换 f

虽然从纯粹的定义角度来看，这并不是一个问题。但出于审美和技术的原因，如果能有一个更高效的实现就更好了。事实上我们可以通过一个 “惰性” 的方式来实现 f。具体来说，挑战者通过跟踪输入/输出对 (x_i, y_i) 来表示随机置换 f。当挑战者收到第 i 个查询 x_i 时，他会测试是否存在某个 $j < i$ 能使得 $x_i = x_j$；如果确实存在这样的 j，他就令 $y_i \leftarrow y_j$（这确保挑战者实现了一个函数）；否则，他就从 $\mathcal{X} \setminus \{y_1, \ldots, y_{i-1}\}$ 中随机选出一个 y_i（这确保该函数是一个置换）；最后，他将 y_i 发送给对手。我们可以把挑战者的逻辑表述如下：

当收到对手 $\mathcal{A}$ 的第 i 个查询 $x_i \in \mathcal{X}$ 时：
  如果存在某个 $j < i$ 使得 $x_i = x_j$ 成立：
    则令 $y_i \leftarrow y_j$
    否则令 $y_i \xleftarrow{\mathrm{R}} \mathcal{X} \setminus \{y_1, \ldots, y_{i-1}\}$
  将 y_i 发送给 $\mathcal{A}$。

为了使这个实现尽可能快，我们可以使用一个恰当的字典数据结构（比如哈希表、搜索前缀树、平衡树等）来实现对 “如果存在某个 $j < i$ 使得 $x_i = x_j$ 成立” 的测试。假设可以有效地生成 $\mathcal{X}$ 中的随机元素，那么实现 “$y_i \xleftarrow{R} \mathcal{X} \setminus \{y_1, \ldots, y_{i-1}\}$” 这一步的方法如下：

重复 $y \xleftarrow{\mathrm{R}} \mathcal{X}$ 直至 $y \notin \{y_1, \ldots, y_{i-1}\}$
令 $y_i \leftarrow y$

同样，我们可以使用适当的字典数据结构来检查 “$y \notin \{y_1, \ldots, y_{i-1}\}$” 是否成立。当 $i < |\mathcal{X}|/2$ 时，期望上只需要运行两次迭代即可找到满足要求的 y_i。

一种理解这个实现的方式是，实验 1 中的挑战者就是一个 “黑盒子”，但盒子里有一个**忠实的侏儒 (faithful gnome)**，它的工作就是维护一张代表随机置换 f 的输入/输出表，如图 4.3 所示。

4.1.3 强安全的分组密码

请注意，在攻击游戏 4.1 中，解密算法 D 从未被使用过。事实上，我们可以定义一个攻击游戏来给出一个更强的安全概念，在这个游戏中，对手被允许向挑战者发起两种类型的查询：

- **前向查询**：对手向挑战者发送一个值 $x_i \in \mathcal{X}$，挑战者以 $y_i := f(x_i)$ 应答对手；
- **反向查询**：对手向挑战者发送一个值 $y_i \in \mathcal{X}$，挑战者以 $x_i := f^{-1}(y_i)$ 应答对手（在攻击游戏的实验 0 中，这是使用算法 D 完成的）。

接下来，我们可以为这个攻击游戏定义一个相应的优势。如果对于所有有效对手，这个优势都是可以忽略不计的，那么我们就称这个分组密码是**强安全**的。我们把这个定义的细节留给读者去解决（见练习 4.9）。除了在后面一章中的应用实例（练习 9.12）之外，我们不会在本文中使用这个概念。

4.1.4 直接使用分组密码进行加密

既然分组密码是一种特殊的密码，我们当然可以考虑直接使用它进行加密。问题是，一个安全的分组密码是否也是语义安全的?

只要消息空间和数据分组空间相等，上面的问题的答案就是 “是的”。下面的定理 4.1 将会指出这一点。然而在实践中，分组密码的数据分组非常短，正如我们之前提到的，AES 的数据分组只有 128 比特。如果我们想加密更长的消息，一个自然的想法是将一个长消息分解成一连串的数据分组，并对每个数据分组单独进行加密。这种使用分组密码来加密长消息的方法被称为**电子密码本模式 (electronic codebook mode, ECB)**。

更确切地说，假设 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{X})$ 上的分组密码。对于任意多项式边界的 $\ell \geq 1$，我们可以定义一个 $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{X}^{\leq \ell})$ 上的密码 $\mathcal{E}' = (E', D')$，方法如下：

- 对于 $k \in \mathcal{K}$ 和 $m \in \mathcal{X}^{\leq \ell}$，如果记 $v := |m|$，我们定义：

$$E'(k, m) = (E(k, m[0]), \ldots, E(k, m[v-1]))$$

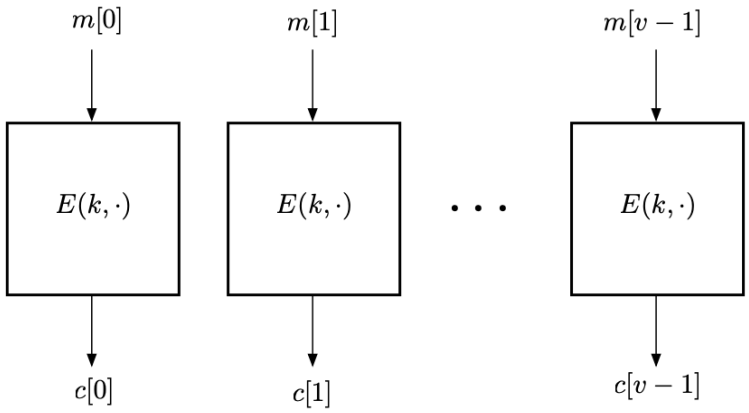
- 对于 $k \in \mathcal{K}$ 和 $c \in \mathcal{X}^{\leq \ell}$，如果记 $v := |c|$，我们定义：

$$D'(k, c) = (D(k, c[0]), \ldots, D(k, c[v-1]))$$

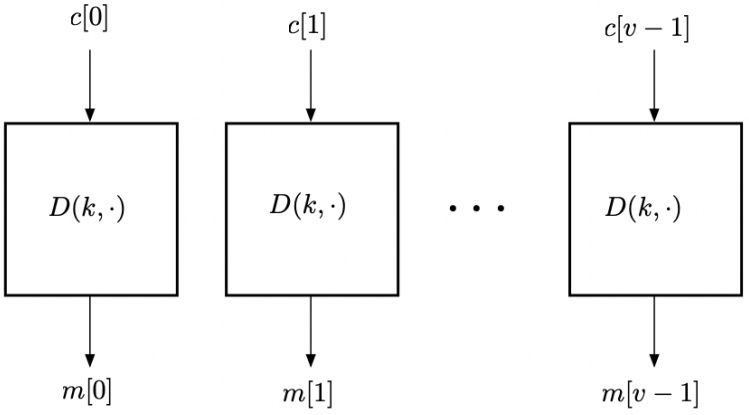
图 4.4 展示了加解密的工作逻辑。我们称 $\mathcal{E}'$ 为**由 $\mathcal{E}$ 派生的 ℓ 次 ECB 密码 (ℓ-wise ECB cipher derived from $\mathcal{E}$)**。

ECB 密码与例 2.3 和例 2.6 中讨论的置换密码有非常密切的关系。主要区别在于，我们现在不是从 $\mathcal{X}$ 上所有可能的置换中完全随机地选择一个，而是在小得多的置换 $\{E(k, \cdot) : k \in \mathcal{K}\}$ 中选择。另一个不太重要的区别是，在例 2.3 中，我们定义的置换密码拥有定长的消息空间（这实际上只是一个任意的选择，因为我们也可以将置换密码定义在变长的消息空间上），而 ECB 密码的消息空间可以是变长的。除此以外，在例 2.3 中，我们举了一个大小为 27 的置换密码例子，但如果我们使用像 AES 这样的 128 比特分组长度的密码，其 “字母表” 要大得多得多，事实上有 2^{128} 那么大。尽管有这么多的差异，例 2.6 中讨论的置换密码的一些缺陷在 ECB 密码中也同样存在。下面我们举个例子。如果对两条消息 $m_0, m_1 \in \mathcal{M}^2$ 进行 ECB 加密，其中 m_0 由两个相同的分组组成（即 $m_0[0] = m_0[1]$），而 m_1 由两个不同的分组组成（即 $m_1[0] \neq m_1[1]$），那么对手很容易就能区分出这两条消息的加密结果。单只因为这个原因，*ECB 密码就不符合我们对语义安全的定义，因此我们强烈反对将 ECB 作为加密方案使用*。

图 4.5 以图像的方式表现了这种能够轻易分辨相同明文分组的能力。这里，图像数据使用 ECB 模式加密，每个数据分组都来自对明文中小像素点的编码。由于相同的像素块会被映射到相同的密文上，所以原始图片中相同的像素在密文中也是相同的，我们可以从密文中依稀看到明文的轮廓。



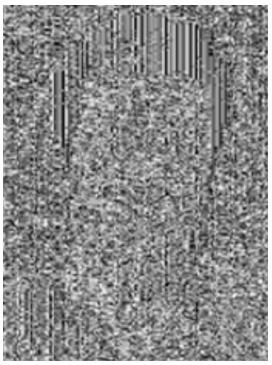
(a) 加密



(b) 解密

图 4.4: ECB 模式的加密与解密

(a) 明文



(b) 使用 AES 在 ECB 模式下加密明文

图 4.5: ECB 模式中的加密

但是请注意，也有一些例 2.6 中讨论的缺陷在这里并不直接适用。假设我们在加密一个 ASCII 编码的文本。如果分组大小是 128 比特，那么每个字符通常会被编码为一个字节，这样一个分组就由 16 个字符组成。这时对手就无法像例 2.6 中那样轻易地找到个别重复字符的位置了。

在本节的最后，我们将要表明，如果消息空间被限制为不同数据分组的序列，那么 ECB 模式实际上是安全的。这对于单个分组加密的特殊情况也是适用的。例如，假设我们使用的是 AES，它有 128 位的数据分组。那么，我们可以从每个数据分组中分配出 32 比特作为计数器，并将剩余的 96 比特作为承载消息的比特。有了这样的策略，我们可以将任何长达 $2^{32} \cdot 96$ 比特的消息编码为一串不同的数据分组。当然，这种策略的缺点是密文会比明文长 33%。

定理 4.1. *令 $\mathcal{E} = (E, D)$ 是一个分组密码，$\ell \geq 1$ 是任意多项式边界的值，令 $\mathcal{E}' = (E', D')$ 是由 $\mathcal{E}$ 派生的 ℓ 次 ECB 密码，但其消息空间被限制为最多 l 个不同数据分组的所有可能序列。如果 $\mathcal{E}$ 是一个安全的分组密码，那么 $\mathcal{E}'$ 是一个语义安全的密码。*

特别地，对于每个对 $\mathcal{E}'$ 进行攻击游戏 2.1 的语义安全对手 $\mathcal{A}$，都存在一个对 $\mathcal{E}$ 进行攻击游戏 4.1 的分组密码对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是围绕 $\mathcal{A}$ 的一个基本包装器，满足：

$$\mathrm{SSadv}[\mathcal{A}, \mathcal{E}'] = 2 \cdot \mathrm{BCadv}[\mathcal{B}, \mathcal{E}] \tag{4.3}$$

证明思路. 基本思想是，如果交给对手一个对不同数据分组的序列加密后的密文，那么对手能看到的实际上也只是一个随机数据分组的序列（无替换采样）。 □

证明. 如果 $\mathcal{E}$ 定义在 $(\mathcal{K}, \mathcal{X})$ 上，令 $\mathcal{X}_*^{\leq \ell}$ 表示 $\mathcal{X}$ 中由最多 ℓ 个不同元素所组成的所有序列的集合。

令 $\mathcal{A}$ 是一个有效对手，他如攻击游戏 2.1 中那样攻击 $\mathcal{E}'$。我们的目标是，假设 $\mathcal{E}$ 是一个安全的分组密码，证明 $\mathrm{SSadv}[\mathcal{A}, \mathcal{E}']$ 是可忽略不计的。使用语义安全攻击游戏的比特猜测版本更加方便。我们试图证明：

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}'] = \mathrm{BCadv}[\mathcal{B}, \mathcal{E}] \tag{4.4}$$

对于某个有效对手 $\mathcal{B}$ 成立。那么根据定理 2.10，我们就能得到式 4.3。

所以，考虑对手 $\mathcal{A}$ 在攻击游戏 2.1 的比特猜测版本中对 $\mathcal{E}'$ 的攻击。在这个游戏中，$\mathcal{A}$ 向挑战者发送两个长度相同的消息 m_0, m_1，然后挑战者选择一个随机密钥 k 和一个随机比特 b，并用 k 加密 m_b，将得到的密文 c 交给 $\mathcal{A}$；最后，$\mathcal{A}$ 输出一个比特 $\hat{b}$。如果 $\hat{b} = b$，则对手 $\mathcal{A}$ 赢得游戏。

挑战者在该游戏中的逻辑可以表示如下：

当从对手 $\mathcal{A}$ 处收到消息 $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$ 时，记 $v := |m_0| = |m_1|$：
选取 $b \xleftarrow{R} \{0,1\}$
选取 $k \xleftarrow{R} \mathcal{K}$
令 $c \leftarrow (E(k, m_b[0]), \ldots, E(k, m_b[v-1]))$
将 c 发送给 $\mathcal{A}$。

我们将该游戏称作**游戏 0**。我们下面还将定义游戏 1 和游戏 2。对于 $j = 0, 1, 2$，我们记 W_j 为 $\mathcal{A}$ 在游戏 j 中输出的 $\hat{b} = b$ 的事件。根据定义，我们有：

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}'] = |\Pr[W_0] - 1/2| \tag{4.5}$$

游戏 1。该游戏与游戏 0 基本相同，只是现在挑战者用一个随机的 $f \in \mathrm{Perms}[\mathcal{X}]$ 代替 $E(k, \cdot)$。我们的挑战者现在的看起来是这样的：

当从对手 $\mathcal{A}$ 处收到消息 $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$ 时，记 $v := |m_0| = |m_1|$：
选取 $b \xleftarrow{R} \{0,1\}$
选取 $f \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$
令 $c \leftarrow (f(m_b[0]), \ldots, f(m_b[v-1]))$
将 c 发送给 $\mathcal{A}$。

直观地说，$\mathcal{E}$ 是一个安全的分组密码的事实意味着对手应该注意不到这个变化。为了严格地证明这一点，我们下面展示了如何构建一个分组密码对手 $\mathcal{B}$，使得它是围绕 $\mathcal{A}$ 的一个基本包装器，且满足：

$$|\Pr[W_0] - \Pr[W_1]| = \mathrm{BCadv}[\mathcal{B}, \mathcal{E}] \tag{4.6}$$

$\mathcal{B}$ 的设计直接来自于游戏 0 和 1 的逻辑。对手 $\mathcal{B}$ 对 $\mathcal{E}$ 进行攻击游戏 4.1 中的攻击，其工作原理如下：

记 f 为 $\mathcal{B}$ 的分组密码挑战者在攻击游戏 4.1 中选择的函数。我们让 $\mathcal{B}$ 扮演 $\mathcal{A}$ 的挑战者的角色，其工作逻辑如下：

当从对手 $\mathcal{A}$ 处收到消息 $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$ 时，记 $v := |m_0| = |m_1|$：
选取 $b \xleftarrow{R} \{0,1\}$
令 $c \leftarrow (f(m_b[0]), \ldots, f(m_b[v-1]))$
将 c 发送给 $\mathcal{A}$。

请注意，$\mathcal{B}$ 通过查询它自己的分组密码挑战者来计算 $f(m_b[0]), \ldots, f(m_b[v-1])$。最后，当 $\mathcal{A}$ 输出一个比特 $\hat{b}$ 时，$\mathcal{B}$ 输出 $\delta(\hat{b}, b)$，其中 δ 的定义见式 3.7。

显然，当 $\mathcal{B}$ 处于其攻击游戏的实验 0 时，它会以 $\Pr[W_0]$ 的概率输出 1。而当 $\mathcal{B}$ 处于其攻击游戏的实验 1 时，它会以 $\Pr[W_1]$ 的概率输出 1。这样我们就能得到式 4.6。

游戏 2。我们现在重写游戏 1 中的挑战者，让其使用我们在 4.1.2 小节中讨论的“忠实的侏儒”来实现随机置换。每个消息 m_0 和 m_1 都需要由不同的数据分组组成（我们的挑战者不需要验证这一点），因

此我们的侏儒的工作很容易：它甚至不需要看输入数据分组，因为这些数据分组被确保是不同的；然而，它仍然需要确保它产生的输出分组是不同的。

我们可以把我们的挑战者的逻辑表述如下：

选取 $y_0 \xleftarrow{R} \mathcal{X}$，$y_1 \xleftarrow{R} \mathcal{X} \setminus \{y_0\}$，…，$y_{\ell-1} \xleftarrow{R} \mathcal{X} \setminus \{y_0, \ldots, y_{\ell-2}\}$
当从对手 $\mathcal{A}$ 处收到消息 $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$ 时，记 $v := |m_0| = |m_1|$：
    选取 $b \xleftarrow{R} \{0,1\}$
    令 $c \leftarrow (y_0, \ldots, y_{v-1})$
    将 c 发送给 $\mathcal{A}$。

由于我们的侏儒是忠实的，我们有：

$$\Pr[W_1] = \Pr[W_2] \tag{4.7}$$

此外，我们声称：

$$\Pr[W_2] = 1/2 \tag{4.8}$$

这源于这样一个事实：在游戏 2 中，对手的输出 $\hat{b}$ 是它自己的随机选择的函数，$y_0, \ldots, y_{\ell-1}$ 也是一样。由于这些值（根据定义）与 b 无关，因此 $\hat{b}$ 和 b 是相互独立的。所以式 4.8 成立。

综合式 4.5，式 4.6，式 4.7 和式 4.8，我们就能得到式 4.4，因此定理 4.1 得证。 □

4.1.5 数学细节

和之前一样，我们下面讨论一些之前被忽略了的数学细节。

由于分组密码只是一种特殊的密码，所以关于分组密码的定义，其实没有什么是 2.3 节中没有交代的。像往常一样，定义 4.1 需要被正确地解释。首先，在攻击游戏 4.1 中，我们要理解，对于安全参数 λ 的每个值，我们都会得到一个不同的概率空间，它由挑战者的随机选择和对抗者的随机选择共同决定。其次，挑战者会产生一个系统参数 Λ，并在游戏一开始就将其发送给对手。第三，优势 $\mathrm{BCadv}[\mathcal{B}, \mathcal{E}]$ 是安全参数 λ 的一个函数，安全性则意味着它是一个可忽略不计函数。

4.2 在实践中构建分组密码

分组密码是密码学中的一个基本原语，许多其他系统都是基于它建立的。几乎所有在实践中使用的分组密码都使用了相同的基本框架，被称为**迭代密码 (iterated cipher)** 范式。为了构建一个迭代分组密码，设计者需要做出以下两个抉择：

- 首先，他需要选择一个简单的分组密码 $\hat{\mathcal{E}} := (\hat{E}, \hat{D})$，它本身显然是不安全的。我们称 $\hat{\mathcal{E}}$ 为**轮密码 (round cipher)**。
- 其次，他还需要选择一个简单（但不一定安全）的 PRG G，用来将密钥 k 扩展为 $\hat{\mathcal{E}}$ 的 d 个密钥 $k_1, \ldots, k_d$。我们称 G 为**密钥扩展函数 (key expansion function)**。

一旦做出这两个选择，迭代分组密码 $\mathcal{E}$ 就完全确定下来了。加密算法 $E(k, x)$ 的工作原理如下（另见图 4.6）：

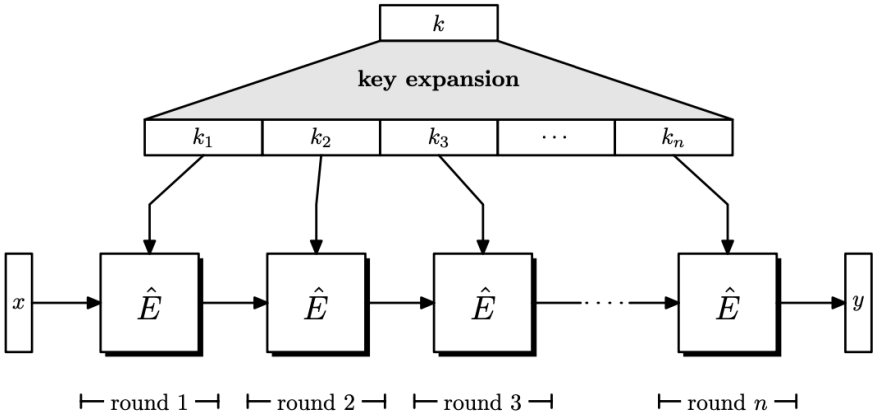


图 4.6: 现实世界中分组密码的加密

算法 $E(k, x)$:

1. **密钥扩展**: 使用密钥扩展函数 G 将 $\mathcal{E}$ 的密钥 k 扩展为 $\hat{\mathcal{E}}$ 的 d 个密钥:
$$(k_1, \ldots, k_d) \leftarrow G(k)$$
2. **迭代**: 对于 $i = 1, 2, \ldots$, 应用 d 次 $\hat{E}(k_i, \cdot)$, 即:
$$y \leftarrow \hat{E}(k_d,\ \hat{E}(k_{d-1}, \ldots, \hat{E}(k_2,\ \hat{E}(k_1,\ x)) \ldots))$$

每次应用 $\hat{\mathcal{E}}$ 称为一**轮 (round)**, 总轮数为 d。$k_1, \ldots, k_d$ 被称为**轮密钥 (round keys)**。解密算法 $D(k, y)$ 与加密算法 $E(k, x)$ 基本相同，除了解密时轮密钥是被反向使用的。$D(k, y)$ 的定义如下:

$$x \leftarrow \hat{D}(k_1, \hat{D}(k_2, \ldots, \hat{D}(k_{d-1}, \hat{D}(k_d, y)) \ldots))$$

表 4.1 列出了一些常见的分组密码和它们的相关参数。我们将在下一节中介绍 DES 和 AES。

迭代是否能提供一个安全的分组密码? 没有人知道。然而，启发式的证据表明，分组密码的安全性来自于对简单密码的多次迭代。但并非所有的轮密码都能发挥作用。例如，迭代一个这样的线性函数:

$$\hat{E}(k, x) = k \cdot x \mod q$$

无论多少次都不会生成一个安全的分组密码，因为 $\hat{E}$ 的迭代只会是另一个线性函数。目前还没有办法辨别哪些轮密码最终能产生安全的分组密码。此外，对于一个候选的轮密码 $\hat{E}$，我们也还没有严格的方法来衡量需要迭代它多少次才能生成一个安全的分组密码。我们知道的是，某些函数，比如线性函数，是完全不可能导出安全分组密码的。但是简单的非线性函数似乎在几次迭代后就能得到一个安全的分组密码。

密码学家面临的挑战是要想出一个快速的轮密码，它在几轮之内就能收敛为一个安全分组密码。看看表 4.1，拿 AES-128 来说，它使用了一个很简单的轮密码，但只需要十轮就能够产生一个安全的分组密码，这种性能就十分理想。

	key size (bits)	block size (bits)	number of rounds	performance[1] (MB/s)
DES	56	64	16	80
3DES	168	64	48	30
AES-128	128	128	10	163
AES-256	256	128	14	115

表 4.1: 分组密码示例

注意事项。 虽然本节解释了几种分组密码的内部工作原理，但它并没有教授如何设计新的分组密码。事实上，本节的主要收获之一是，读者不应自行设计分组密码，而应该始终使用这里描述的标准密码。分组密码的设计非同小可，需要经过多年的分析才能对某一具体方案产生充分信心。此外，读者甚至不应该自己实现分组密码，因为分组密码的实现往往容易受到计时攻击和功耗攻击，就如 4.3.2 小节将要讨论的。使用诸如 OpenSSL 等密码库中免费提供的标准实现要安全得多。这些实现经历了多年来的反复分析，并已被多次加固以抵御各种攻击。

4.2.1 案例研究：DES

上世纪 70 年代，美国国家标准局 (National Bureau of Standards, NBS)，即现在的美国国家标准技术研究所 (National Institute of Standards and Technology, NIST) 向社会公开征集加密方案。因应这一需求，IBM 开发了数据加密标准 (Data Encryption Standard, DES)。它于 1975 年在联邦公报上发表，并于 1977 年被采纳为 “非机密” 应用的加密标准。DES 算法单枪匹马地开启了密码分析领域，因为所有人都想破解它。自诞生以来，DES 经历了相当多的分析，也间接导致了许多分组密码分析工具的出现。

DES 的前身是 IBM 早期设计的一个分组密码，名为 Lucifer。Lucifer 的某些变体使用 128 比特密钥对 128 比特的分组进行操作。然而，国家标准局要求设计使用更短分组（64 比特）和更短密钥（56 比特）的分组密码。作为回应，IBM 团队设计了一个符合这些要求的密码，并在最终成为了 DES。将 DES 的密钥大小设定为 56 比特在当时饱受批评。甚至有人猜测 DES 的这种弱设计是美国情报机构故意要求的。在接下来的章节中，我们还会看到将分组大小减少到 64 比特也带来了许多问题。

由于密钥太短，DES 算法现在被认为是不安全的，不应该再被使用。然而，一个名为 3DES 的加强版 DES 在 1998 年被重新确立为美国国家标准。NIST 已经批准 3DES 供政府使用直至 2030 年。在 2002 年，DES 被一个更高效的分组密码标准所取代，即 AES，它使用 128 比特（或更长）的密钥，并在 128 比特的分组上运行。

[1]性能数字是使用 OpenSSL 1.0.1e 在 Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz (Haswell) 处理器上运行获得的。

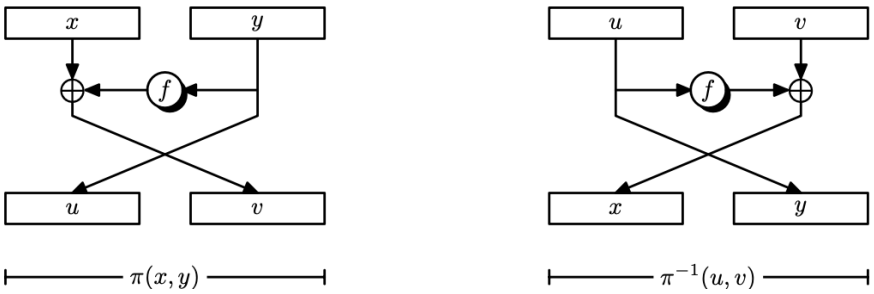


图 4.7: 费斯妥置换法

4.2.1.1 DES 算法

DES 算法由一个简单的轮密码经 16 次迭代组成。为了描述 DES，我们需要先介绍 DES 轮密码和 DES 密钥扩展函数。我们下面依次介绍它们。

费斯妥置换法。 DES 的关键创新之一是由 IBM 的霍斯特 · 费斯妥 (Horst Feistel) 发明的费斯妥置换法 (Feistel Permutation)，它能基于任意函数建立一个置换。令 $f : \mathcal{X} \to \mathcal{X}$ 是一个函数，我们按如下方法构建一个置换 $\pi : \mathcal{X}^2 \to \mathcal{X}^2$（见图 4.7）：

$$\pi(x, y) := (y,\ x \oplus f(y))$$

为了证明 π 是一个双射，我们给出它的逆变换：

$$\pi^{-1}(u, v) = (v \oplus f(u),\ u)$$

映射 π 被称为**费斯妥置换**，它被用于构建 DES 轮密码。n 个费斯妥置换的组合被称为 n **轮费斯妥网络 (n-round Feistel network)**。设计成费斯妥网络的分组密码被称为**费斯妥密码**。对于 DES 来说，函数 f 需要 32 比特输入，产生的映射 π 对 64 比特的分组进行操作。

需要注意的是，费斯妥逆置换 π^{-1} 几乎与 π 相同。因此，只用设计同一套硬件电路就能计算 π^{-1} 和 π。这也意味着加密和解密电路也可以使用相同的硬件实现。

DES 轮函数。 DES 加密算法是一个 16 轮费斯妥网络，每轮使用一个不同的函数 $f : \mathcal{X} \to \mathcal{X}$。在第 i 轮中，函数 f 的定义为：

$$f(x) := F(k_i, x)$$

其中，k_i 是第 i 轮的 48 比特密钥，F 是一个固定函数，称为 **DES 轮函数**。函数 F 是 DES 算法的核心，如图 4.8 所示。F 使用的几个辅助函数 E、P 和 $S_1, \ldots, S_8$ 定义如下：

- 函数 E 通过对输入比特的重新排列和复制，将 32 比特输入扩展为 48 比特输出。例如，E 将输入的第 1 比特映射到输出的第 2 和第 48 比特，将输入的第 2 比特映射到输出的第 3 比特，以此类推。

- 函数 P 称为**混合置换 (mixing permutation)**，它通过重新排列输入的比特，将 32 比特输入映射到 32 比特输出。例如，P 将输入的第 1 比特映射到输出的第 9 比特，将输入的第 2 比特映射到输出的第 15 比特，以此类推。

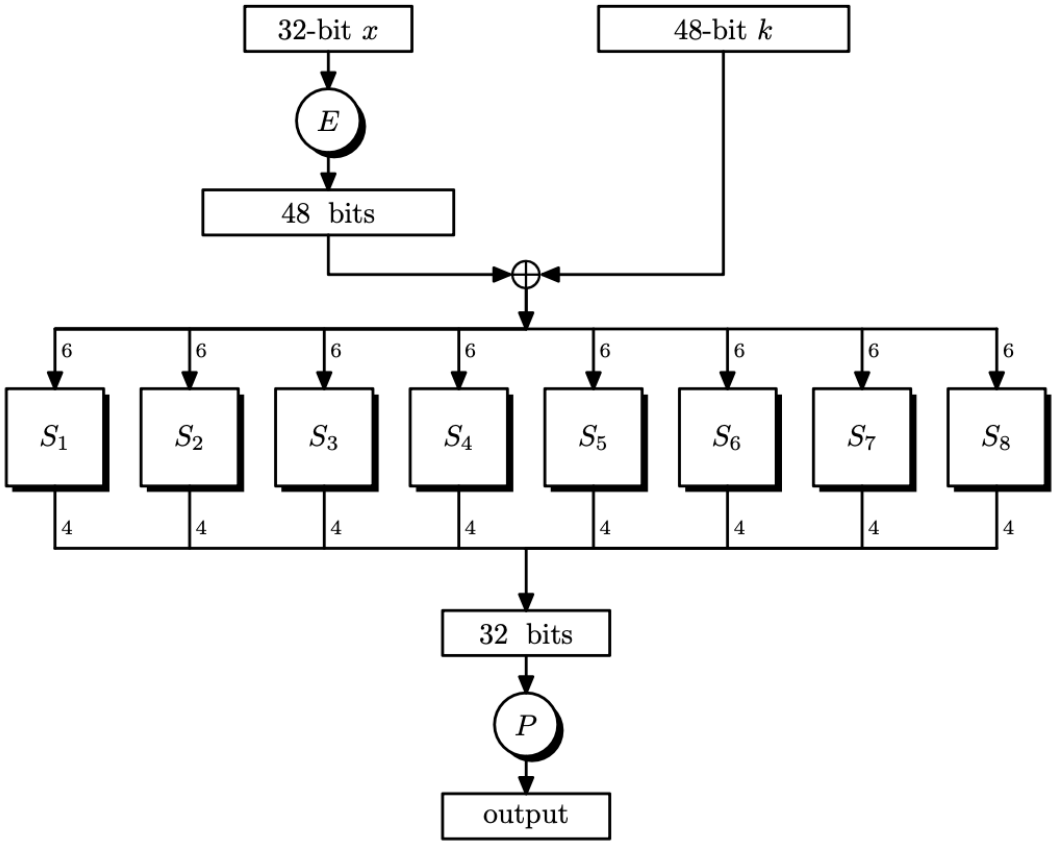


图 4.8: DES 的轮函数 $F(k, x)$

- 函数 $S_1, \ldots, S_8$ 称为 **S 盒 (S-boxes)**。每个 S 盒 S_i 通过一个查找表将一个 6 比特输入映射到一个 4 比特输出。DES 标准列出了这 8 张查找表，每张表中包含 64 个条目。

基于以上函数，DES 的轮函数 $F(k, x)$ 工作原理如下：

输入：$k \in \{0,1\}^{48}$ 和 $x \in \{0,1\}^{32}$
输出：$y \in \{0,1\}^{32}$

$F(k, x)$：
计算 $t \leftarrow E(x) \oplus k \in \{0,1\}^{48}$
将 t 分成 8 组，每组 6 比特：$t := t_1 \| \cdots \| t_8$
对于 $i = 1, 2, \ldots, 8$，令 $s_i \leftarrow S_i(t_i)$
组合 $s \leftarrow s_1 \| \cdots \| s_8 \in \{0,1\}^{32}$
计算 $y \leftarrow P(s) \in \{0,1\}^{32}$
输出 y

除了 S 盒，DES 的轮密码完全由异或操作和比特置换组成。八个 S 盒是设计中唯一引入的非线性组件。在公开文献揭示了一种名为 “差分密码分析” 的强大攻击技术后，IBM 于 1994 年公布了设计 S 盒的标准。IBM 的这份报告清楚地表明，DES 的设计者早在 1973 年就已经了解了多年后才在公开文献中出现的这种攻击技术。他们在设计 DES 时加入了抵抗这种攻击的因素。下面这段话解释了对 S 盒设计标准保密的原因：

> 我们在设计 DES 时利用了某些密码分析技术的知识，其中最突出的是 “差分密码分析” 技术，这些技术在公开的文献中并不为人所知。在与美国国家安全局讨论后，我们认为如果披露设计思路，就会反过来揭示这种差分密码分析技术。这种技术非常强大，可以用来对付很多密码器，因此会削弱美国在密码学领域相对其他国家的竞争优势。

一旦差分密码分析技术被公开，再保密 DES 的设计思路也就没太多意义了。由于 S 盒的重要性，我们下面列举了当初设计它的一些准则：

1. 将 6 比特输入映射到 4 比特输出的查找表，是以 1974 年的技术能在单个芯片上做到的最大尺寸；
2. S 盒的任何输出比特都不应该接近于输入比特的线性函数。也就是说，如果我们选择任何一个输出比特和 6 个输入比特的任何一个子集，这个输出比特等于这些输入比特的异或结果的概率应该接近 $\frac{1}{2}$；
3. 如果我们把输入的最左边和最右边的比特固定在一个 S 盒上，那么产生的 4 比特到 4 比特的函数是一个双射。特别地，这意味着每个 S 盒都是一个 4 到 1 的映射；
4. 改变一个 S 盒的输入比特，至少会改变输出的两个比特；
5. 对于 $x, y \in \{0,1\}^6$ 的 64 种可能情况，令 $\Delta = x \oplus y \in \{0,1\}^6$，$S_i(x) \oplus S_i(y)$ 的结果是同一个值最多不能超过 8 次。

这些标准是为了使 DES 在 56 比特密钥大小的限制下能尽可能地强大。现在我们已经知道，如果 S 盒是简单随机选出的，那么很有可能产生的 DES 密码是不安全的。特别是，可能只需要对挑战者进行几百万次查询，对手就可以恢复出秘钥。

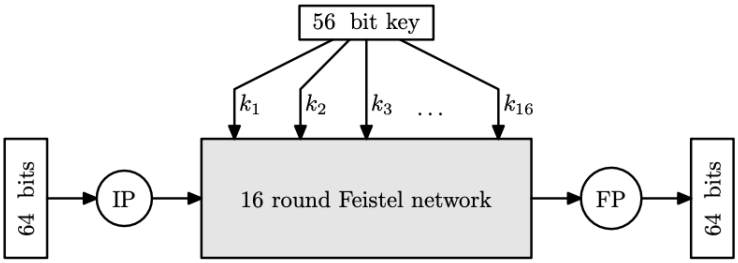


图 4.9: 完整的 DES 电路

除了 S 盒之外，混合置换 P 也起着重要的作用。它确保 S 盒不总是在同一组 6 比特上进行操作。同样，P 也是根据一些标准决定的。如果 P 是简单随机选出的，那么 DES 的安全性也将大打折扣。

密钥扩展函数。 DES 的密钥扩展函数 G 将 56 比特的密钥 k 作为输入，输出 16 个密钥 $k_1, \ldots, k_{16}$，每个 48 比特长。每个密钥 k_i 都由从 56 比特密钥 k 中选出的 48 比特组成，每个 k_i 使用 k 中不同比特的子集。

DES 算法。 完整的 DES 算法架构如图 4.9 所示，它包括 DES 轮密码的 16 次迭代，以及被称为 IP 和 FP 的初始和最后的置换排列。这些置换只是重新排列了 64 比特的输入和输出，其中 FP 变换是 IP 变换的逆变换。

IP 和 FP 没有任何密码学意义，被包括在算法内的原因至今不明。由于比特置换在软件实现中很慢，但在硬件实现中却很快，一种理论认为，IP 和 FP 是为了故意减慢 DES 的软件实现而加入的。

4.2.2 对 DES 的穷举搜索：DES 的挑战

回顾一下，对分组密码 (E, D) 的穷举搜索攻击（4.1.1.2 小节）指的是这样的攻击：给予对手少量的明文分组 $x_1, \ldots, x_Q \in \mathcal{X}$ 和使用 $\mathcal{K}$ 中的密钥 k 加密它们得到的密文分组 $y_1, \ldots, y_Q$。对手尝试所有可能的 $\kappa \in \mathcal{K}$，直至找到一个密钥 k，能够将给定的明文转化为给定的密文。如果给定的密文分组足够多，k 就能够被对手唯一确定。

对于像 DES 和 AES-128 这样的分组密码，仅需三个分组就足以确保有唯一密钥能以较大概率将给定的明文分组映射到给定的密文分组。我们将在 4.7.2 小节讨论理想密码及其特性时看到原因。现在我们只需知道，给定三个明文/密文分组对，攻击者就可以用穷举法找到秘钥 k。

在 1974 年设计 DES 时，对 256 比特的密钥空间进行穷举搜索攻击被认为是不可行的。但随着计算机硬件的持续发展，人们渐渐发现 56 比特的密钥远远不够。

为了证明对 DES 的穷举搜索是可行的，RSA 数据安全部设置了一连串的挑战，称为 **DES 挑战**。规则很简单：在一个预先宣布的日期，RSA 数据安全部公布了 DES 的三个输入/输出对。第一个找到相应密钥的小组就能赢得一万美元。为了使挑战更具娱乐性，挑战包含 n 个 DES 输出 $y_1, y_2, \ldots, y_n$，其中前三个输出 y_1, y_2, y_3 是对下面这 24 字节明文消息应用 DES 后得到的结果：

$$\underbrace{\texttt{The unkn}}_{x_1}\underbrace{\texttt{own mess}}_{x_2}\underbrace{\texttt{age is:}}_{x_3}$$

它由 3 个 DES 分组构成，每个分组为 8 字节，也就是 64 比特。目标是找到一个 DES 密钥，对于

$i = 1, 2, 3$，该密钥能够将 x_i 映射到 y_i。得到密钥后，挑战者就能够使用密钥来解密 $y_4, \ldots, y_n$ 编码的秘密消息。

第一个挑战在 1997 年 1 月发布。DESCHALL 项目组用 96 天的时间解决了它。该团队在 78,000 名志愿者的帮助下使用众包搜索的方式进行破解，这些志愿者贡献了他们自己设备的闲置算力。找到秘钥的志愿者获得了 40% 的奖金。解密后大家得知，$y_4, \ldots, y_n$ 对应的明文是 “强大的密码学使世界更加安全。” (Strong cryptography makes the world a safer place.)

第二个挑战于 1998 年 1 月发布。**distributed.net** 项目组使用了类似的但规模更大的众包搜索，仅用时 41 天就解决了该挑战。

1998 年初，电子前沿基金会 (Electronic Frontiers Foundation, EFF) 与 Paul Kocher 签订合同，建造一台专门的设备来进行针对 DES 的穷举密钥搜索。这台机器被称为 **DeepCrack**，耗资 25 万美元，包含大约 1900 个专用 DES 芯片，装在六个柜子里。这些芯片并行工作，每个芯片在指定的密钥空间中搜索。当 RSA 数据安全部在 1998 年 7 月发布下一个挑战时，DeepCrack 在 56 小时内就解决了这个挑战，并轻松赢得了一万美元的奖金。这还不足以支付机器的成本，但足以说明 DES 存在致命问题。

最后的挑战是在 1999 年 1 月发布的。在 DeepCrack 和 *distributed.net* 的共同努力下，它在 22 小时内就被破解了。这在 DES 的棺材板上钉上了最后一颗钉子，表明 56 比特的密钥可以在短短几个小时内就被破解。

在这个故事的末尾，2007 年，COPACOBANA 团队建立了一个由现成的 120 块 FPGA 板组成的集群，总成本约为 1 万美元。该集群可以在大约 12.8 天内搜索完整个 2^{56} DES 密钥空间。

上述所有工作将我们导向一个共同的结论，那就是 56 比特密钥太短了。目前的最小安全密钥长度是 128 比特。

4.2.2.1 穷举搜索对 AES-128 是否奏效?

让我们把 DES 上的结论推广到 AES。虽然这些估计本身是不精确的，但它们在一定程度上说明了针对 AES 的穷举搜索的复杂性。AES 的密钥空间最小是 2^{128}。如果扫描一个 2^{56} 的空间需要 22 小时，那么扫描一个 2^{128} 大小的空间将需要:

$$(22 \text{ hours}) \times 2^{128-56} \approx 1.18 \times 10^{20} \text{ years}$$

即使计算速度和计算资源能提升十亿倍，即使 AES 计算比 DES 计算更快，所需的时间也远超人类的认知范围。因此，我们可以得出这样的结论: 对 AES 进行暴力穷举式的搜索攻击是不现实的。然而，正如我们将在 18.7 节中讨论的，如果利用时空权衡这一经典思路，我们仍然可以对 AES-128 发起更复杂的暴力攻击。

4.2.3 加强密码以抵抗穷举攻击: $3\mathcal{E}$ 构造

事实证明，DES 密码对复杂的攻击有很强的抵抗力。尽管经过了多年的分析，对 DES 最实用的攻击仍然是对整个密钥空间进行暴力穷举搜索。唯一的问题就是，56 比特的密钥空间太小了。

一个自然而然的想法是，我们能否在不改变其内部结构的情况下加强密码的抗穷举攻击能力? 最简单的解决方案是使用独立的密钥对密码进行多次迭代。

令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{X})$ 上的分组密码。我们定义分组密码 $3\mathcal{E} = (E_3, D_3)$ 为:

$$E_3((k_1, k_2, k_3),\ x) = E(k_3,\ E(k_2,\ E(k_1, x)))$$

$3\mathcal{E}$ 密码的密钥在 $\mathcal{K}^3$ 上。基于 DES 算法的 $3\mathcal{E}$ 密码被称为 **Triple-DES**，它的密钥长度为 $3 \times 56 = 168$ 比特。

安全性。 为了分析 $3\mathcal{E}$ 密码的安全性，我们需要一个称为*理想密码模型*的框架。我们将在本章末尾介绍这个框架，并在那一节中分析 $3\mathcal{E}$ 密码的安全性。

Triple-DES 标准。 NIST 批准 Triple-DES 在 2030 年前可以供政府使用。严格来说，NIST 版本的 Triple-DES 的定义为：

$$E_3((k_1, k_2, k_3),\ x) = E(k_3,\ D(k_2,\ E(k_1, x)))$$

其原因是，如果设置 $k_1 = k_2 = k_3$，NIST 的 Triple-DES 就可以被还原为普通 DES，因此 Triple-DES 的硬件实现也可以直接复用于单一 DES 场景中。这个修改不会影响我们对 Triple-DES 安全性的讨论。练习 4.5 中还讨论了 Triple-DES 的另一个变体。

4.2.3.1 $2\mathcal{E}$ 构造是不安全的

虽然 Triple-DES 不容易被穷举搜索攻击所攻破，但它的性能比单一 DES 慢三倍，如表 4.1 所示。

那么为什么不使用 Double-DES 呢？它的密钥长度是 $2 \times 56 = 112$ 比特，这已经足以抵抗穷举搜索，而且它的性能也要比 Triple-DES 好得多。

问题在于，Double-DES 并不比单一 DES 更安全。更一般地说，令 $\mathcal{E} = (E, D)$ 是一个具有密钥空间 $\mathcal{K}$ 的分组密码。我们能够证明，定义为：

$$E_2((k_1, k_2),\ x) = E(k_2,\ E(k_1, x))$$

的 $2\mathcal{E} = (E_2, D_2)$ 构造不会比 $\mathcal{E}$ 更安全。针对它的攻击策略被称为**中间相遇 (meet in the middle)**。

假设我们得到了 Q 个明文分组 $x_1, \ldots, x_Q$ 和它们的 $2\mathcal{E}$ 加密 $y_i = E_2((k_1, k_2), x_i)$，其中 $i = 1, \ldots, Q$。我们下面展示如何在与 $|\mathcal{K}|$ 成正比的时间内恢复密钥 (k_1, k_2)，即使密钥空间看起来是 $|\mathcal{K}|^2$。和单一 DES 的穷举搜索一样，只需少量的明文/密文对就足以确保以高概率恢复唯一密钥 (k_1, k_2)，事实上 10 个分组就足以发动这种攻击了。

定理 4.2. *令 $\mathcal{E} = (E, D)$ 是一个定义在 $(\mathcal{K}, \mathcal{X})$ 上的分组密码。存在一种算法 $\mathcal{A}_{EX}$，它以 Q 个明文/密文对 $(x_i, y_i) \in \mathcal{X}^2$ 作为输入，其中 $i = 1, \ldots, Q$，输出一个密钥对 $(k_1, k_2) \in \mathcal{K}^2$，满足：*

$$y_i = E_2\big((k_1, k_2),\ x_i\big) \quad \textit{for all } i = 1, \ldots, Q \tag{4.9}$$

其运行时间的主要成分是算法 E 和算法 D 的总计 $2Q \cdot |\mathcal{K}|$ 次评估的耗时。

证明. 令 $\bar{x} := (x_1, \ldots, x_Q)$，$\bar{y} = (y_1, \ldots, y_Q)$。为了简化符号，我们用：

$$\bar{y} = E_2\big((k_1, k_2),\ \bar{x}\big) = E(k_2,\ E(k_1, \bar{x}))$$

来描述式 4.9 中的 Q 个映射。我们也可以把它写成：

$$D(k_2, \bar{y}) = E(k_1, \bar{x}) \tag{4.10}$$

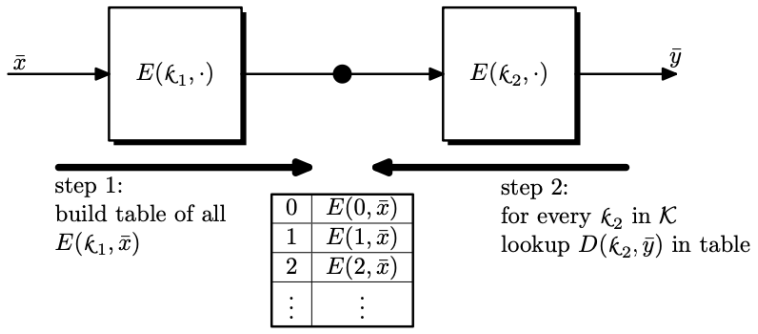


图 4.10: $2\mathcal{E}$ 上的中间相遇攻击

为了找到能满足式 4.10 的一对 (k_1, k_2)，算法 $\mathcal{A}_{EX}$ 进行以下操作：

步骤 1：对于所有的 $k_1 \in \mathcal{K}$，构建一张包含所有数对 $\big(k_1,\ E(k_1,\bar{x})\big)$ 的表 T
步骤 2：对于所有的 $k_2 \in \mathcal{K}$，如下操作：
　　计算 $\bar{x} \leftarrow D(k_2,\bar{y})$
　　查表：如果 T 中包含数对 $(\cdot,\bar{x})$，则
　　　　令 $(k_1,\bar{x})$ 为该数对，输出 (k_1,k_2) 并停机

图 4.10 描述了这种中间相遇攻击。根据构造，算法输出的数对 (k_1, k_2) 必然满足式 4.10，如定理所要求的那样。

算法 $\mathcal{A}_{EX}$ 的步骤 1 需要进行 $Q \cdot |\mathcal{K}|$ 次对 E 的评估，而步骤 2 需要进行 $Q \cdot |\mathcal{K}|$ 次对 D 的评估。因此，该算法的全部运行时间就大约是 $2Q \cdot |\mathcal{K}|$ 次对 E 或 D 进行评估的时间。我们假设向查找表 T 中插入元素和查找元素的时间小于评估算法 E 和 D 的时间。 □

如上所述，对于相对较小的 Q 值，绝大多数情况下只有一个密钥对 (k_1, k_2) 满足式 4.9，它就是定理 4.2 中算法 $\mathcal{A}_{EX}$ 的输出。

定理 4.2 中算法 $\mathcal{A}_{EX}$ 的运行时间与对 $\mathcal{E}$ 进行穷举搜索攻击的时间差不多，这表明 $2\mathcal{E}$ 并没有增强 $\mathcal{E}$ 对穷举搜索的抵抗力。然而，该定理只考虑了 $\mathcal{A}_{EX}$ 的运行时间。需要注意的是，$\mathcal{A}_{EX}$ 必须在内存中维护一张很大的查找表，这可能是很困难的。为了攻击 Double-DES，$\mathcal{A}_{EX}$ 需要存储一张规模为 2^{56} 的表，其中每个表项包含一个 DES 密钥和一条短的密文。总的来说，这相当于至少 2^{60} 字节，也就是大约一百万 TB。虽然不是不可能，但获得如此多的存储可能是相当困难的。但是，攻击者可以通过时空权衡的方式来减少存储上的要求，方法是修改 $\mathcal{A}_{EX}$，使其在运行过程中仅保留查找表中的 $1/\epsilon$，代价是运行时间也要相应增加 $1/\epsilon$。

Triple-DES 上的中间相遇攻击。 类似的中间相遇攻击也适用于上一小节中的 $3\mathcal{E}$ 构造。虽然 $3\mathcal{E}$ 的密钥空间为 $\mathcal{K}^3$，但对 $3\mathcal{E}$ 的中间相遇攻击的时间大约为 $|\mathcal{K}|^2$，所需空间为 $|\mathcal{K}|$。在 Triple-DES 的情况下，该攻击需要对 DES 进行大约 $|\mathcal{K}|^2 = 2^{112}$ 次评估，这在实践中的运行时间实在太长了。因此，Triple-DES 可以抵御这种中间相遇攻击，这也是 Triple-DES 在实践中能够被广泛使用的原因。

4.2.4 案例研究：AES

虽然 Triple-DES 是 NIST 批准的密码，但它有一些明显的缺点。首先，Triple-DES 比 DES 慢三倍，在软件中实现时表现很差。其次，64 比特的分组长度对于一些重要的应用来说还是不够（即第六章中的应用）。到 20 世纪 90 年代中期，所有人都在期待一个新的联邦分组密码标准。

AES 的历程。 1997 年，NIST 发起了一个关于新的分组密码标准的提案征求，该标准被称为**高级加密标准 (Advanced Encryption Standard, AES)**。NIST 要求 AES 密码必须在 128 比特分组上运行，并支持 128、192 和 256 比特三种密钥长度。截止到 1997 年 9 月，NIST 收到了 15 份提案，其中许多来自美国以外的国家。在举行了两次公开会议讨论这些提案后，NIST 在 1999 年将名单缩小到五个候选者。随后又进行了一轮激烈的密码分析，最终在 2000 年 4 月举行的 AES3 会议上，最后五个团队的代表分别作了发言，论证他们的标准为什么应该被选为 AES。2000 年 10 月，NIST 宣布比利时分组密码 **Rijndael** 被选为 AES 密码。2001 年 11 月，当 AES 作为 NIST 的标准在 FIPS 197 中公布后，它正式成为了一个官方标准。这结束了长达五年的替换 DES 的标准化过程。

Rijndael 是由比利时的密码学家尤安 · 达蒙 (Joan Daemen) 和文森特 · 雷曼 (Vincent Rijmen) 设计的。AES 与原始版本的 Rijndael 密码略有不同。例如，Rijndael 支持大小为 128、192 或 256 比特的分组，而 AES 只支持 128 比特的分组。

4.2.4.1 AES 算法

和许多现实中的分组密码一样，AES 也是一个迭代密码，它将一个简单的轮密码迭代数次。迭代的次数取决于秘钥的长度。

![Schematic of the AES-128 block cipher](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/6a0e64af-d6ea-492a-8a9e-fed7d5bb5360/Untitled.png)

Schematic of the AES-128 block cipher

上图所示的是一个包含十轮的 AES-128 密码的结构。里面的 Π_{AES} 是 $\{0,1\}^{128}$ 上的一个固定置换，而且是一个双射。每轮的最后一步是将本轮密钥与 Π_{AES} 的输出进行异或。这样重复 9 次，直至最后一轮使用一个稍加修改的置换排列 $\hat{\Pi}_{\text{AES}}$。AES 的逆运算可以通过反向运行上面的步骤来完成。

遵循上图所示结构的密码被称为**交替密钥密码 (alternating key cipher)** 或**迭代艾文-曼苏尔密码 (iterated Even-Mansour cipher)**。如果每轮的置换排列 Π_{AES} 满足某些理想假设，这种密码就能够保证安全性。我们会在本章的后面部分详细介绍这一性质。

![https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c48ee644-e2f5-422d-9ed7-908b24ea0189/Untitled.](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c48ee644-e2f5-422d-9ed7-908b24ea0189/Untitled.png)
us-west-2.amazonaws.com/secure.notion-static.com/c48ee644-e2f5-422d-9ed7-908b24ea0189/Untitled.png)

AES 轮置换

置换排列 Π_{AES} 由对集合 $\{0,1\}^{128}$ 的三个可逆操作组成。输入的 128 比特会被组织成一个 4×4 的元胞数组，其中每个单元是 8 比特。然后在这个 4×4 的数组上依次进行以下三个可逆运算：

1. `SubBytes`：令 $S:\{0,1\}^8\to\{0,1\}^8$ 是一个固定置换，且是一个双射。这个置换排列会用在所有的 16 个元胞上。在 AES 中，S 被设计成一个包含 256 项的硬编码表。在数学上，S 没有不动点，即对于任意 $x\in\{0,1\}^8$，$S(x)\neq x$ 恒成立。S 也没有反不动点，即对于任意 $x\in\{0,1\}^8$，$S(x)\neq\overline{x}$ 恒成立，其中 $\overline{x}$ 表示 x 的按位补码。我们将在后面介绍这么设计的原因。2. `ShiftRows`：这一步对输入的 4×4 数组进行按行循环移位：第一行不变，第二行向左循环移动 1 字节，第三行向左循环移动 2 字节，

第四行向左循环移动 3 字节：

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \end{bmatrix} \Longrightarrow \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ a_5 & a_6 & a_7 & a_4 \\ a_{10} & a_{11} & a_8 & a_9 \\ a_{15} & a_{12} & a_{13} & a_{14} \end{bmatrix}$$

3. `MixColumns`：在这一步中，4×4 数组被视为一个矩阵，该矩阵会与一个固定的矩阵相乘，其中的算术是有限域 $\mathrm{GF}(2^8)$ 上的运算。有限域 $\mathrm{GF}(2^8)$ 中的元素可以表示成 $\mathrm{GF}(2)$ 中元素的不大于 8 阶的多项式，其中的乘法以不可逆多项式 $x^8 + x^4 + x^3 + x + 1$ 为模。具体来说，此步的变换是这样的：

标量 $01, 02, 03$ 是 $\mathrm{GF}(2^8)$ 中元素的二进制表示，比如 03 代表 $\mathrm{GF}(2^8)$ 上的 $x + 1$。这个固定矩阵在 $\mathrm{GF}(2^8)$ 上是可逆的，所以整个变换也是可逆的。

AES 电路中使用的轮置换 Π_{AES} 就由 `SubBytes`、`ShiftRows` 和 `MixColumns` 这三个运算组成。在最后一轮，AES 会使用一个稍微不同的函数，我们称之为 $\hat{\Pi}_{\mathrm{AES}}$。这个函数只省略了 `MixColumns` 这一步骤，这是为了使 AES 的解密电路看起来与 AES 的加密电路类似。

因为 Π_{AES} 的每一步的逆运算都是容易的，所以 Π_{AES} 全过程的逆运算也是容易的，这也是解密所需要的。

使用预处理表实现 AES

AES 的轮函数是由我们之前介绍的置换排列 Π_{AES} 构建的，它包含 `SubBytes`、`ShiftRows` 和 `MixColumns` 这三个部分。但是 AES 的设计者并不是按照这种方式实现 Π_{AES} 的，他们提出了一种更高效的方式，能够利用四个固定查找表 T_0, T_1, T_2, T_3 一次完成上面的三个步骤。

为了解释其工作原理，回顾一下 Π_{AES} 将一个 4×4 的矩阵 $A = (a_i)_{i=0,\ldots,15}$ 作为输入，并输出相同尺寸的矩阵 $A' = \Pi_{\mathrm{AES}}(A)$。记 $S[a]$ 为对输入的 $a \in \{0,1\}^8$ 进行 `SubBytes` 运算后的结果，记 $M[i]$ 为 M 的第 i 列，记 $A'[i]$ 为 A' 的第 i 列。

现在看一下 [式 4.12](https://www.notion.so/0d98f324b5974f018b0d60a5f9b5adde)，我们可以把 $\Pi_{\mathrm{AES}}(A)$ 输出的四列表示为：

其中的加法和乘法都是在域 $\mathrm{GF}(2^8)$ 上进行的。每一列 $M[i]$ 都是 $\mathrm{GF}(2^8)$ 上的一个 4 字节向量，$S[a_i]$ 是 $\mathrm{GF}(2^8)$ 上的 1 字节标量。

[式 4.13](https://www.notion.so/0d98f324b5974f018b0d60a5f9b5adde) 中的每一项都可以用一张固定预处理表来快速计算。对于 $i = 0, 1, 2, 3$，我们定义一张有 256 项的表 T_i 如下：

- 对于 $a \in \{0,1\}^8$，令 $T_i[a] = M[i] \cdot S[a] \in \{0,1\}^{32}$。

用 T_i 表项替换 [式 4.13](https://www.notion.so/0d98f324b5974f018b0d60a5f9b5adde) 中的相关项，就可以得到一个快速计算 $\Pi_{\mathrm{AES}}(A)$ 的方法：

以这种方式实现的 AES 电路就是一个简单的查表序列。由于每张 T_i 表中包含 256 项，每项 4 字节，所以四张表的总大小也不过 4KB。M 矩阵的循环结构使我们可以进一步将四张表压缩到 2KB 而不对性能产生大的影响。

[式 4.13](https://www.notion.so/0d98f324b5974f018b0d60a5f9b5adde) 的一个例外是 AES 的最后一轮，这一轮中的 `MixColumns` 步骤被省略了。为了计算最后一轮，我们需要第五个 256 字节的表 S，它只包含 `SubBytes` 的操作。

这个 AES 优化并不是必须的。如果你没有足够的空间存储 4KB 的查找表，那么你也可以直接按照基本原理用代码分别实现 Π_{AES} 的三个步骤，这节约了存储空间，但运算速度可能会慢一点。必须要注

意的是，这种查表优化的 AES 实现很可能会受到缓存定时攻击，我们会在后面的章节中介绍这种攻击。

AES-128 的密钥扩展方法

回顾 AES 的结构，我们可以看到 AES-128 的密钥扩展算法需要生成 11 个轮密钥 $k_0, \ldots, k_{10}$，其中每个轮密钥都是 128 比特。为此，我们将 128 比特的 AES 密钥分割成四个 32 比特的字 $w_{0,0}, w_{0,1}, w_{0,2}, w_{0,3}$，这些字构成了第一个轮密钥 k_0。其余 10 个轮回密钥按以下方法产生：

- 对于 $i = 1, \ldots, 10$，轮密钥 $k_i = (w_{i,0}, w_{i,1}, w_{i,2}, w_{i,3})$ 按如下方式生成：- $w_{i,0} \leftarrow w_{i-1,0} \oplus g_i(w_{i-1,3})$；- $w_{i,1} \leftarrow w_{i-1,1} \oplus w_{i,0}$；- $w_{i,2} \leftarrow w_{i-1,2} \oplus w_{i,1}$；- $w_{i,3} \leftarrow w_{i-1,3} \oplus w_{i,2}$。

这里的函数 $g_i : \{0,1\}^{32} \to \{0,1\}^{32}$ 是 AES 标准中规定的一个固定函数。它对 4 字节输入进行以下三步操作：

1. 对 4 字节输入进行 1 字节循环左移；2. 对得到的 4 字节中的每个字节分别计算 `SubBytes`；3. 将最左边的字节与一个固定的轮常数 c_i 进行异或。

轮常数 $c_1, \ldots, c_{10}$ 也是在 AES 标准中规定的，其中 c_i 就是 $\mathrm{GF}(2^8)$ 上的元素 x^{i-1} 的二进制 8 比特序列形式。

AES-192 和 AES-256 的密钥扩展方法与 AES-128 的类似。对于 AES-192 来说，每次迭代都会产生 6 个 32 比特的字，但只有前 4 个字会被用作 AES 的轮密钥。对于 AES-256，每次迭代产生 8 个字，也只有前 4 个字会被用作 AES 的轮密钥。

AES 的密钥扩展方法被有意设计成可反转的：只要给定最后一轮的密钥，我们就可以反过来恢复完整的 AES 秘钥 k。这样做的原因是为了确保每个 AES-128 的轮密钥，就其本身而言，具有与 AES-128 的秘钥 k 相同的熵。不幸的是可逆性也有助于攻击，它可以被用来发起对 AES 的侧信道攻击。

AES 的安全性

AES 算法经受住了针对它的各种复杂的密码分析尝试。截止到成稿时，最有名的攻击有如下这些：

- ** 密钥恢复攻击 **。密钥恢复攻击是指对手在得到若干明文-密文对的情况下，基于这些明密文对恢复出密钥的攻击方式。已知的设计最优化的对 AES-128 的密钥恢复攻击需要进行 $2^{126.1}$ 次 AES 计算。这比穷举搜索要快四倍左右，但需要的时间仍然太长了。因此这种攻击对 AES-128 的安全性影响不大。

对 AES-192 最优化的攻击需要进行 $2^{189.74}$ 次 AES 计算，这也只比穷举搜索快了四倍左右。对 AES-256 的最著名的攻击需要 $2^{254.42}$ 次 AES 计算，这比穷举搜索快三倍。这些攻击都不能影响任何一个 AES 变体的安全性。

- ** 关联密钥攻击 **。在 l 路关联密钥攻击中，对手得到了 l 张明密文对列表。对于 $i = 1, \ldots, l$，第 i 张表是用密钥 k_i 生成的。重点是，所有 l 个密钥 $k_1, \ldots, k_l$ 都必须满足对手选择的一些固定关系。攻击者的目标是恢复其中一把密钥。在实现良好的密码系统中，密钥总是独立随机生成的，不太可能满足所需的关系。因此，关联密钥攻击通常不会威胁到正确的加密实现。

AES-256 容易受到关联密钥攻击，该攻击利用了其相对简单的密钥扩展机制。该攻击需要四个关联密钥 k_1, k_2, k_3, k_4，其中的关系是一个简单的异或关系：它要求 $k_1 \oplus k_2$，$k_1 \oplus k_3$ 和 $k_2 \oplus k_4$ 的某些比特被置为特定值。然后给定由这四个密钥产生的明密文对的列表，攻击者可以在 $2^{99.5}$ 时间内恢复这四个密钥。这比在 AES-256 上进行穷举搜索的时间要快得多。虽然这种攻击相当有趣，但它并不能影响 AES-256 在完善的系统中的安全性。

AES 的硬件实现

在 AES 刚被标准化为美国国家加密标准的时候，大多数的实现都还是基于软件的。软件产品对 AES 的广泛采用，促使所有主要的处理器供应商扩展他们的指令集，以增加对 AES 硬件实现的支持。

例如英特尔在其 Xeon 和 Core 系列处理器中增加了名为 **AES-NI** 的指令，以加快在软件中使用 AES 的计算速度。新指令的工作原理如下：

- `AESKEYGENASSIST`：运行密钥扩展程序，从 AES 密钥中生成 AES 轮密钥；- `AESENC`：运行 AES 轮加密算法。该指令的调用方法是：

```
AESENC xmm15, xmm1
```

其中 `xmm15` 寄存器保存 128 比特数据分组，`xmm1` 寄存器保存该轮的 128 比特轮密钥。得到的 128 比特分组会被写入寄存器 `xmm15`。在 AES 的前九轮中，可以事先将轮密钥加载到寄存器 `xmm1`, . . . , `xmm9` 中，然后调用九次该指令。

- `AESENCLAST`：调用与 `AESENC` 类似的指令来运行 AES 算法的最后一轮。最后一轮的功能与前面几轮稍有不同，它省略了 `MixColumns` 这一步。- `AESDEC` 和 `AESDECLAST`：运行 AES 解密算法，与加密指令类似。

这些 AES-NI 硬件指令的速度比极尽所能去优化 AES 的软件实现还要更快。Emilia Käsper 在 2009 年的实验表明，在英特尔 Core 2 处理器上使用 AES-NI 指令的 AES 需要 1.35 周期/字节，而优化的软件实现需要 7.59 个周期/字节。

在英特尔 2015 年推出的 Skylake 处理器中，`AESENC`、`AESDEC` 和 `AESENCLAST` 指令各需要四个周期来完成。这些指令是完全流水线式的，因此每个周期都可以发射一条新指令。换句话说，英特尔将 `AESENC` 的执行划分为四个阶段的流水线，四个 AES 分组可以由流水线的不同阶段同时处理。虽然处理一个 AES-128 分组需要 40 个周期，但在一条流水线上处理四个分组只需要 44 个周期。因此流水线可以将 AES 的速度提高近 4 倍。

除了速度之外，AES 的硬件实现还提供了更好的安全性，因为它可以抵御下一节中将介绍的侧信道攻击。

4.3 针对分组密码的复杂攻击

4.3.1 算法攻击

4.3.2 边信道攻击

4.3.3 针对 AES 的错误注入攻击

4.3.4 量子穷举搜索攻击

4.4 伪随机函数：基本定义与性质

4.4.1 定义

4.4.2 随机函数的有效实现

4.4.3 什么时候一个安全的分组密码是安全的 PRF?

4.4.4 从 PRF 构建 PRG

4.4.5 数学细节

4.5 使用 PRF 构建分组密码

4.6 树构造：从 PRG 到 PRF

4.7 理想密码模型

4.7.1 正式定义

4.7.2 理想密码模型中的穷举搜索

4.8 一个有趣的应用：比较但不透露信息

4.9 笔记

对文献的引用有待补充。

4.10 练习

第五章　选择明文攻击

5.1　简介

5.2　针对多密钥攻击的安全性

5.3　针对选择明文攻击的语义安全性

5.4　构建 CPA 安全的密码

5.4.1　placeholder

5.5　基于 nonce 的加密

5.6　一个有趣的应用：可撤销的广播加密

5.7　笔记

5.8　练习

第六章　消息完整性

第七章　来自通用哈希的消息完整性

第八章　来自抗碰撞哈希的消息完整性

第九章　认证加密

第二部分

公钥密码学

第十章　公钥基本工具

第十一章　公钥加密

11.1　门限加密

第十二章　选择密文安全的公钥加密

第十三章　数字签名

第十四章　基于哈希的快速签名

第十五章　椭圆曲线密码学与配对

第十六章　后量子密码学

第十七章　对数论假设的分析

第三部分

密码学协议

第十八章　用于身份识别与认证的协议

在这一部分，我们关注身份认证和登录问题。考虑甲方希望向乙方表明自己的身份以获取乙方提供的可用资源。他们使用身份识别协议来实现这一目的。识别协议是密码学提供的基本工具之一。我们下面给出几个具体应用场景作为说明，旨在为读者建立一个直观的理解。

打开一把门锁。Alice 想通过数字门锁的身份识别，以打开门锁并进入房间。Alice 可以使用一个简单的口令系统，她将电子钥匙插入门锁，如果这个钥匙提供了一个有效的口令，门锁就会打开。与这个场景类似的场景是电脑或者手机上的本地登录界面，当 Alice 想要向设备表明身份来获取访问权时，她可以使用口令来解锁电脑或者手机。

解锁汽车。Alice 想要用一个无线硬件钥匙来解锁她的智能汽车，这把钥匙扣能与汽车互动。对手可以窃听无线电信道并记录钥匙扣与汽车之间的一次或若干次通信。尽管如此，对手仍然无法通过这种*窃听攻击*解锁汽车。

登录银行自动取款机。Alice 想要使用 ATM 机从自己的账户中提取现金。问题是，她可能正在与一台假的 ATM 机交互。一份来自某个大型 ATM 设备提供商的报告显示，假 ATM 机正在成为银行业的一个重大威胁：

> 早在 1993 年，就有犯罪团伙在曼彻斯特的一个购物中心安装了一台假的自动柜员机。这种假机器的设计目的并不是直接偷钱，而是在顾客面前假装不能正常工作，同时从试图使用的顾客那里窃取银行卡的信息。

使用这样的一个假的 ATM 机，对手就可以与 Alice 交互并试图窃取她的凭证，并在之后使用该凭证来假冒 Alice 进行认证。我们称这种对手为*主动*对手。我们的目标是设计识别协议，确保即使是这种主动对手也无法攻破安全系统。

登录到一个在线银行账户。Alice 想要访问她的在线银行账户。她的浏览器首先与银行建立了一个安全信道。随后 Alice 通过安全信道运行一个身份识别协议向银行证明自己的身份，比如使用口令，就像之前的例子一样。问题是对手同样可以克隆一个银行的网站来欺骗 Alice，这样 Alice 就会向对手的网站提供自己的信息。这种攻击被称为**网络钓鱼 (phishing)**，它是另一个主动攻击的典型例子，对手在与 Alice 交互的过程中扮演一个积极的角色。

对手试图窃取她的凭证，以便以后可以将凭证卖给任何希望冒充 Alice 到真正银行的人。同样，我们的目标是确保即使是网络钓鱼的对手也无法得知 Alice 的有效凭证。我们将在第 21.11.1 节中更详细地讨论网络钓鱼攻击，并介绍一种潜在的密码学防御手段。

身份识别协议 (Identification protocols)。 身份识别协议在上述所有的场景中都会用到。抽象地讲，身份识别问题涉及到两方，即一个**证明者 (prover)** 和一个**验证者 (verifier)**。

比如说在 ATM 的例子中，Alice 扮演证明者的角色，而 ATM 机则扮演验证者的角色。证明者有

一个**证明私钥** sk，它能用来说服验证者相信自己的身份。而验证者有一个相应的**验证密钥** vk，它能用来确认证明者的声称。我们有时会把证明者称为人类用户，把验证者称为计算机或服务器。

上面的几个例子也揭示了身份识别协议的三种攻击模型，我们将在接下来的章节中详细讨论这些模型：

- **直接攻击**：门锁和本地登录的例子描述了物理距离上接近的证明者和验证者之间的交互。如果对手无法窃听到交互的内容，那么除了公开的信息之外，对手无法得到任何其他的有效信息，因此对手必须以某种方式在验证者面前冒充证明者。一个简单的口令协议就足以抵御这种直接攻击。

- **窃听攻击**：在无线汽车钥匙的例子中，对手可以窃听无线电信道并获得证明者和验证者之间的若干次交互记录。在这种情况下，简单的口令协议也是不安全的。但是稍微复杂一点的一次性口令协议就能够保证安全。

- **主动攻击**：最后两个例子，也就是假 ATM 机和假的网上银行，展示了一个更加积极主动的对手。对手在这种场合中试图主动学习一些信息，让他能够在以后的验证过程中冒充证明者。抵御这种主动攻击的身份识别协议往往需要在证明者和验证者之间进行多次交互，我们通常把这种技术称为挑战-应答。

当 Alice 试图登录一台已受感染的计算机时，发生的攻击也属于主动攻击。感染计算机的恶意软件可以展示一个假的登录界面并欺骗 Alice 与之交互，从而发动主动攻击。以这种方式窃取用户口令的恶意软件被称为**木马 (Trojan horse)**。被盗的口令可以用来冒充 Alice 登入其他设备。

验证密钥的保密与公开. 在一些身份识别协议中，验证者必须将其验证密钥 vk 保密，而另一些协议允许 vk 公开。显然，可以公开 vk 的协议更具优势，因为在这种协议中，即使验证者（比如 ATM 机）被攻破，也不会造成任何损害。

无状态协议与有状态协议. 理想情况下，在协议设置阶段之后，vk 和 sk 的值就不会再被修改。但是在某些协议中，vk 和 sk 的值在每次协议执行时都会更新，其中证明者更新 sk，验证者更新 vk。我们称 vk 和 sk 永远固定的协议为**无状态**协议，而 vk 和 sk 会被更新的协议为**有状态**协议。一些有状态协议能够比无状态协议以更低的成本提供更高的安全性。但是有状态协议往往更难使用，因为证明者和验证者需要始终保持正确同步。

单向证明与相互识别. 在本章中，我们只研究单向的识别问题，即 Bob 希望验证 Alice 的身份。与此相关的另一个话题是**相互识别**，即 Bob 同样也要向 Alice 证明自己的身份。我们将在第 21 章探讨这部分内容，届时我们会构造能够生成一个共享密钥的相互识别协议。

身份识别协议的安全性和局限性. 识别协议是为了防止对手在没有 Alice 协助的情况下仿冒 Alice。在定义身份识别协议的安全性时，我们可以允许对手窃听协议通信，甚至直接与 Alice 进行交互。但是，当对手开始尝试冒充 Alice 时，他必须在不与 Alice 通信的情况下进行。在上面的例子里，比如打开门锁的场景，我们给出了一些设定，主要目标就是在 Alice 不在的时候防止冒充。

然而，身份识别协议并不足以在 Alice 和远程的参与方（比如 Alice 的银行）之间建立安全会话。问题在于，身份识别协议很容易受到中间人 (MiTM) 攻击。假设 Alice 通过一个不安全的信道与她的银行执行一个身份识别协议，那么如果对手控制了信道，他就可以随意阻止或注入信息。对手等待 Alice 与

她的银行运行识别协议，并将所有协议信息从一方转发到另一方。一旦协议成功完成，对手就可以向银行发送请求，这些请求似乎来自 Alice，因此银行会尊重这些请求。实际上，对手利用 Alice 向银行进行身份验证，然后劫持会话以向银行发送自己的信息。

为了抵抗 MiTM 攻击，我们可以把识别协议和会话密钥交换协议结合起来，正如我们将在第 21 章讨论的那样。在 Alice 和她的银行之间共享会话密钥可以防止对手仿冒 Alice 注入消息。

18.1 交互式协议：基本表记

在具体介绍身份识别协议之前，我们首先需要准确定义**交互式协议 (interactive protocol)**。

一个交互式协议可以在任意数量的参与方之间进行，但在本文中我们主要关注两方协议。不管有多少参与方，一个协议都可以运行很多次。我们把每一次协议的运行称为**协议实例 (protocol instance)**。

在任意一个协议实例中，每一方都会从初始化设置开始。随着协议的运行，各参与方会发送和接收消息，并更新其本地设置。尽管具体细节因协议不同而各异，但是我们可以使用一个**交互式协议算法**来对协议实例中各方的计算进行建模。这种算法是一个概率性算法 I，它接收元组 $(config_{\text{old}}, data_{\text{in}})$ 作为输入，并输出一个新元组 $(config_{\text{new}}, data_{\text{out}})$。当某一方开始执行一个协议实例时，他首先需要提供一个**输入值**，这个输入值用于对该参与方的协议实例进行**初始化设置 (initial configuration)**。当该参与方从网络中收到一个新的消息时，它会调用算法 I 并输入 $(config_{\text{old}}, data_{\text{in}})$，其中 $config_{\text{old}}$ 是该方先前的状态，$data_{\text{in}}$ 是接收到的消息；I 将会输出元组 $(config_{\text{new}}, data_{\text{out}})$，其中 $config_{\text{new}}$ 是该方新的状态，$data_{\text{out}}$ 是发出的消息。该方会通过传输信道发送消息给协议的另一参与方。根据协议的要求，上述过程会被反复执行若干次，直到达到某个**终止设置 (terminal configuration)**。这种终止配置可以指定一个**输出值**，该输出值可以被该方使用，比如说用于一些更高层的协议中。

一般来说，一个参与方可能会运行多个协议，或同一协议的多个实例。所有这些不同的协议实例的设置都需要单独维护。

18.1.1 数学细节

像往常一样，我们可以使用第 2.3 节中定义的术语更精确地定义上述概念。除了上面定义的输入，交互式协议算法 I 还会接受一个安全参数 λ 和系统参数 Λ 作为输入。然而有几个细节值得讨论。

简单起见，我们可以认为一个运行中的协议实例的设置规模是多项式边界的。也就是说，我们可以将设置为一个比特序列，且该序列的长度总是以固定的 λ 的多项式为界。这使得我们可以将定义 2.8 应用于算法 I。同样，该定义也要求算法的任何输入也同样应是多项式边界的。因此，对于任何多项式边界的输入，算法 I 的输出也同样是多项式边界的。

我们在这里试图解决的问题如下。假设在每一轮之后，设置的大小会增加一倍。几轮之后，设置的大小将会指数爆炸，尽管在每一轮中，计算的时间与当前配置大小成多项式关系。通过坚持要求状态的大小满足多项式边界，我们就可以避免出现指数爆炸的情况。

简单起见，我们还需要坚持协议的轮复杂度 (round complexity) 也是多项式边界的。我们主要对执行恒定轮数的协议感兴趣。更一般地，我们希望协议的轮复杂度也是由 λ 的固定多项式来约束。这可以通过要求从任何初始配置开始，I 经过多轮迭代后必然达到一个最终状态来实现。

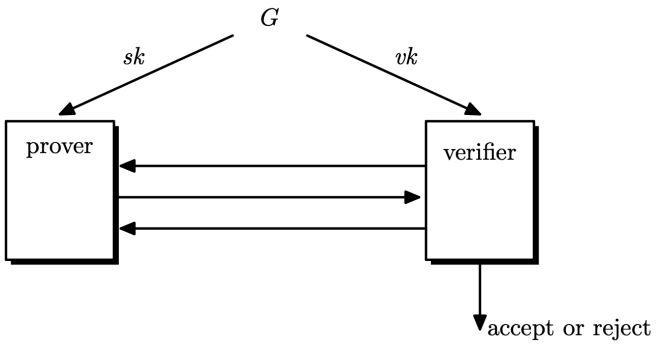


图 18.1: 身份认证协议中的证明者与验证者

18.2 身份认证协议的定义

我们首先定义图 18.1 所示的构成身份识别协议的各个算法。

定义 18.1. 一个**身份识别协议** $\mathcal{I} = (G, P, V)$ 由三部分组成：

- G 是一个概率性的**密钥生成**算法，不接受任何输入，并输出一对密钥 (vk, sk)，其中 vk 被称为**验证密钥** ***(verification key)***，sk 被称为**私钥** ***(secret key)***；
- P 是一个交互式协议算法，被称为**证明者** ***(prover)***，它接受 sk 作为输入；
- V 是一个交互式协议算法，被称为**验证者** ***(verifier)***，它接受 vk 作为输入，输出 accept 或 reject。

我们要求，当 $P(sk)$ 和 $V(vk)$ 能够正确交互时，$V(vk)$ 总是输出 accept。也就是说，对于 G 的所有可能输出 (vk, sk)，如果 P 由 sk 初始化，V 由 vk 初始化，那么当 P 和 V 交互结束时，V 输出 accept 的概率为 1。

18.3 口令协议：针对直接攻击的安全性

在**基本口令协议 (Basic Password protocol)** 中，证明者的密钥是一个**口令** pw。在该协议中，证明者将 pw 发送给验证者，验证者需要检查 pw 是否是合法的口令。因此，该协议的密钥 sk 就是 pw。显然，只有当对手无法窃听证明者和验证者之间的交互时，这个协议才可以被使用。为了完善对基本口令协议的定义，我们还需要指定验证者如何检查给定的口令是否合法。

首先想到的方法是直接将验证者的公钥也定义为 $vk := pw$，这样验证者只需要检查他从证明者那里收到的口令与 vk 是否相等即可。但是这种简陋的协议显然是有问题的，因为如果验证者被攻破，所有验证者存储的口令都会直接泄露。

为了避免这个问题，我们可以让验证者保存口令的哈希值，而不是口令本身。我们称这个修改后的口令协议为**版本 1**。我们下面用一种相当理想化的方式来描述这个协议，在该协议中，我们从某个有限的口令空间中随机均匀地选择口令，但实际情况可能并非如此。

口令协议版本 1. 证明者的私钥 sk 是一个从有限口令空间 $\mathcal{P}$ 中随机均匀选出的口令 pw，验证者的公钥 $vk = H(pw)$，其中的哈希函数为 $H : \mathcal{P} \to \mathcal{Y}$。这样，**口令身份认证协议** $\mathcal{I}_{\text{pwd}} = (G, P, V)$ 的定义如下：

id_1	$H(pw_1)$
id_2	$H(pw_2)$
id_3	$H(pw_3)$
$\vdots$	$\vdots$

图 18.2: 储存在服务器中的口令文件（版本 1）

- G：令 $pw \xleftarrow{R} \mathcal{P}$ 并输出 $sk = pw$ 和 $vk = H(pw)$。
- 以 $sk = pw$ 为输入的算法 P 及以 $vk = H(pw)$ 为输入的算法 V，它们按照以下方式交互：
  1. P 将 pw 发送给 V；
  2. 如果收到的 pw 满足 $H(pw) = vk$，V 输出 accept，否则输出 reject。

在一个多用户系统中，验证者（服务器）通常会在数据库中维护一张口令映射表，就像图 18.2 一样。因此，对服务器的攻击不会泄漏出任何的明文口令。

为了分析上述口令协议的安全性，我们下面首先正式定义针对直接攻击的安全性，然后解释为什么上面的协议能够满足这个安全定义。

攻击游戏 18.1 (针对直接攻击安全的身份识别). 对于一个给定的身份识别协议 $\mathcal{I} = (G, P, V)$ 和一个给定对手 $\mathcal{A}$，攻击游戏按照以下方式运行：

- 密钥生成阶段。挑战者运行 $(vk, sk) \xleftarrow{R} G()$ 并将 vk 发送给 $\mathcal{A}$。
- 仿冒尝试阶段。挑战者与 $\mathcal{A}$ 交互，其中挑战者掌握 vk，并按照验证者的算法 V 运行，而对手 $\mathcal{A}$ 扮演证明者的角色，但不一定会诚实执行证明者的算法 P（事实上 $\mathcal{A}$ 根本没有获取到私钥 sk）。

如果 V 在交互结束时输出 accept，我们就称对手 $\mathcal{A}$ 赢得了本游戏。我们定义 $\mathsf{ID1adv}[\mathcal{A}, \mathcal{I}]$ 为对手 $\mathcal{A}$ 相对 $\mathcal{I}$ 的优势，其值为 $\mathcal{A}$ 赢得本游戏的概率。 □

定义 18.2. *如果对于任意有效对手 $\mathcal{A}$，$\mathsf{ID1adv}[\mathcal{A}, \mathcal{I}]$ 的值都可以忽略不计，我们就称身份识别协议 $\mathcal{I}$ 对于直接攻击是安全的。*

请注意，在游戏 18.1 中，对手 $\mathcal{A}$ 能够获得的是验证密钥 vk。因此，一个将明文口令直接存储的简陋口令协议并不满足定义 18.2。下面的简单定理表明，版本 1 的协议是安全的。

定理 18.1. *如果哈希函数 $H : \mathcal{P} \to \mathcal{Y}$ 是单向的，那么口令协议 $\mathcal{I}_{pwd}$ 针对直接攻击是安全的。*

证明简述. 为了攻击协议 $\mathcal{I}_{pwd}$，对手 $\mathcal{A}$ 必须能够提供一个口令 pw' 使得 $H(pw') = H(pw)$。需要注意的是 pw' 可能与 pw 不同。而能够提供满足上述要求的 pw' 的对手显然已经破坏了 H 的单向性。 □

我们可以注意到，针对直接攻击（见攻击游戏 18.1）的安全性事实上是一个非常弱的安全概念。比如说，尽管口令协议 $\mathcal{I}_{pwd}$ 对于直接攻击是安全的，但是对手哪怕能够窃听到协议的一个实例，它显然就丧失了安全性。

```
123456,  password,    12345,    12345678,  football,
qwerty,  1234567890,  1234567,  princess,  1234,
login,   welcome,     solo,     abc123,    admin
```

图 18.3: 依次列出的 2016 年中最常见的 15 个口令

18.3.1 利用字典攻击破解口令

口令协议 $\mathcal{I}_{pwd}$ 在实践中被广泛使用，因为它使用起来非常方便。任何人只要记住一个口令 pw，就可以参与到协议中来，扮演证明者的角色，并且不需要任何额外的计算设备。问题是，人类生成和记忆随机口令的能力是在太过糟糕。在实践中，被使用的口令通常太短了。更糟糕的是，口令通常根本就不是随机生成的，而是根据一些可预测的信息推导来的。

图 18.3 总结了 2016 年进行的一项研究的结果，该研究检查了 500 万个泄露的口令，这些口令大多由北美和西欧的用户持有。数据显示，这些口令与大集合上的均匀分布相去甚远，尤其是有相当比例的口令属于相对较小的常用口令字典。大约 4% 的人使用 “123456” 这个口令，而大约 10% 的人使用前 25 个最常用口令列表中的某一个。图 18.3 中的口令列表在一段时间内非常稳定。每年的变化都非常小。

从现在开始，我们使用**强口令**来表示从一个很大口令空间 $\mathcal{P}$ 中随机均匀选出的口令。只有口令是强口令时，定理 18.1 的安全性声明才有效。相对地，**弱口令**就是指那些从一些常用口令的小字典中按照某个任意分布选取的口令。我们将弱口令的口令空间表示为 $\mathcal{D}$，则必然有 $\mathcal{D} \subseteq \mathcal{P}$。

18.3.1.1 在线字典攻击

假设现在有一个对手怀疑某个用户的口令很弱，属于某个常用口令的小字典 $\mathcal{D}$。那么对手就可以发动一个**在线字典攻击**：他只需要逐一尝试 $\mathcal{D}$ 中的所有口令，直到找到匹配的口令。为了进一步提升效率，对手可以按照流行程度对 $\mathcal{D}$ 中的元素进行排序，首先尝试更流行的口令。

一个常见的防御在线字典攻击的方法是，在特定用户 ID 或来自特定 IP 地址的登录尝试每两次失败后，就将服务器的响应时间增加一倍。因此在十次登录失败后，下一次尝试的时间就会是正常响应时间的 32 倍。即使一个用户并不能熟记自己的口令，他也不过需要等待一点时间，所以对他来说这种设计的影响并不大。但是对于恶意攻击者来说，通过暴力破解的方式去猜测口令会变得非常困难。

对手为了应对这种反制对策，可以在许多不同的用户名中尝试同一个普通的口令，比如 `123456`。此外，对手可以利用位于不同 IP 地址的肉机来反制对于单个 IP 地址的登录尝试次数限制。用这种方式对随机账户进行攻击对于一般的对手来说往往已经足够了，他们通常会在地下论坛中交易攻击得到的数据。

非密码学的防御手段对于阻吓这些在线攻击已经很有效了。但是仍然有更具破坏性的攻击，这种攻击也更难阻止。我们将在接下来讨论这种攻击。

18.3.1.2 离线词典攻击

攻击者如果破坏了登录服务器，就可以窃取存储在服务器上的口令数据库。这给了攻击者一个大的哈希口里 ing 列表，每个在该系统注册的用户都有一个口令。

侵入登录服务器的对手可以窃取服务器上存储的口令数据库，攻击者可以借此得到大量哈希后的口令。除了直接破坏服务器外，还有许多其他方法可以获得口令文件。例如一项研究表明，在 eBay 上购

买的二手硬盘可以包含很多有趣的、未被删除的数据，其中就包括口令文件。

现在假设一个对手设法获得了某个用户的验证密钥 $vk = H(pw)$。如果口令 pw 是弱的，并且属于某个常见口令的小字典 $\mathcal{D}$，那么对手就能够发动**离线字典攻击**，他可以这么做：

$$\begin{aligned} &\text{对每个 } w \in \mathcal{D}: \\ &\quad \text{如果 } H(w) = vk: \\ &\quad\quad \text{输出 } w \text{ 并停机} \end{aligned} \tag{18.1}$$

如果 pw 在 $\mathcal{D}$ 中，那么依照上述过程，对手迟早能够获取 pw，抑或是另一个 pw' 使得 $H(pw') = H(pw)$。

这种离线字典攻击的时间复杂度是 $O(|\mathcal{D}|)$，其中时间单位为对一个输入计算一次 H 的耗时。这种计算完全可以*脱机*运行，不需要与证明者或服务器有任何交互。

口令统计. 2016 年，一个名为 CrackStation 的口令破解服务发布了一个大小约为 15 亿的的常用口令词典。经验证据表明，人类生成的口令中有接近 50% 都在这个列表中。这意味着，每经过约 15 亿次的离线哈希，每两个口令中就有一个能被破解。如果哈希函数 H 是 SHA256，那么现代 GPU 只需要不到一分钟的时间就能够完成破解。由此我们能够得出一个结论：简单地使用 SHA256 对口令进行哈希处理对保护口令数据来说远远不够。

从另一个角度来说，我们可以发现，只包含可打印字符的 8 位字符口令的总数大约是 $95^8 \approx 2^{52}$ 个，这是因为美式键盘上有 95 个可输入字符。使用现代 GPU 阵列对这组口令中的所有单词运行 SHA256 也不过仅仅几天就可以完成。这说明，*所有* 8 位以下长度口令在服务器被攻破的情况下都是不安全的。

量子离线口令攻击. 更糟糕的是，一旦有了大规模的量子计算机，上面的穷举搜索攻击将会更加容易。我们在 4.3.4 节中介绍过，量子计算机可以在 $\sqrt{n}$ 级别的时间内搜索大小为 n 的空间。因此对于 8 位长度的口令空间来说，量子搜索相当于只需要进行 $\sqrt{2^{52}} = 2^{26}$ 次哈希计算。这在现代经典计算机上也仅仅需要几秒钟而已。换言之，由于 8 位长度口令在经典计算机上是不能抵抗穷举搜索的，所以一旦我们拥有一台在速度和大小上与当前经典计算机相当的量子计算机，16 位长度的口令也将失去安全保障。我们将在 18.4.3 节中讨论一些针对该问题防御措施。

18.3.1.3 含预处理的离线字典攻击

如果对手能够在发动攻击之前对字典 $\mathcal{D}$ 进行预处理，那么上面讨论的离线字典攻击将会更容易发动。一旦获取了哈希后的口令 vk，对手就能够立即从字典中查到原始口令 pw。具体来说，我们可以将字典攻击分为两个阶段：一个**预处理阶段**，在获取任何哈希口令之前发动；一个**攻击阶段**，针对特定口令 vk 发起攻击。我们的目标是尽量减少攻击阶段破解特定 vk 所需的时间。

一个简单的带有预处理阶段的字典攻击原理如下：

$$\begin{aligned} &\text{预处理阶段}: \\ &\quad \text{对于每个 } pw \in \mathcal{D}, \text{ 将 } (pw, H(pw)) \text{ 加入列表 } L \\ &\text{针对输入 } vk \text{ 的攻击阶段}: \\ &\quad \text{如果在表 } L \text{ 中存在一项 } (pw, vk), \text{ 则输出 } pw \\ &\quad \text{否则输出 fail} \end{aligned} \tag{18.2}$$

假设我们把对一个口令计算一次 H 的耗时作为时间单位，那么预处理阶段所需要的时间为 $O(|\mathcal{D}|)$ 级。如果表 L 存储在一个支持常数时间查找的哈希表中，那么攻击阶段将会非常快，耗时仅仅为常数级。

批量离线字典攻击. 一旦完成预处理，攻击者就能利用它快速破解许多哈希口令。具体来说，假设攻击者从一个被攻破的登录服务器中获得了一个大型的哈希口令数据库 F，那么现在使用字典 $\mathcal{D}$ 破解 F 中的哈希后口令仅需要：

$$\begin{aligned} &\text{预处理阶段：} O(|\mathcal{D}|) \\ &\text{攻击阶段：} O(|F|) \end{aligned} \tag{18.3}$$

其中 $|F|$ 指 F 中哈希口令的数量。上述批量离线字典攻击的总工作量是 $O(|\mathcal{D}| + |F|)$，这比针对 F 中每一个元素分别单独发起式 18.1 的离线字典攻击要快得多，因为后者的总工作量是 $O(|\mathcal{D}| \times |F|)$。

回忆一下，18.3.1.2 节中的统计数据表明，对手可以使用 CrackStation 字典找到 F 中近*半数*的口令。因此一旦完成预处理，只需要 $O(|F|)$ 时间就能完成攻击。事实上这种攻击能够使用极少的工作量暴露数百万个被破解的口令。

时空权衡. 式 18.2 所展示的基于预处理的字典攻击要求攻击者建立并存储一个包含大量哈希口令的列表 L。然而当字典 $\mathcal{D}$ 是所有 2^{52} 个 8 字符口令的集合时，这个表 L 可能会相当大，存储成本过高。在第 18.7 节中，我们会介绍一种名为**彩虹表**的方法，它在预处理阶段构建一个小得多的表 L，并用它快速破解口令。比如说，在 $n := |\mathcal{D}|$ 的情况下，该方法能够实现：

$$\begin{aligned} &\text{表大小：} O(n^{2/3}) \\ &\text{预处理时间：} O(n) \\ &\text{攻击时间：} O(n^{2/3}) \end{aligned}$$

表的大小从 $O(n)$ 缩减到 $O(n^{2/3})$ 级别。然而，攻击一个哈希口令的时间从 $O(1)$ 级增长到 $O(n^{2/3})$ 级。换言之，我们用更长的攻击时间换取了表 L 大小的缩减。正因此，这种方法被称为**时空权衡**。我们通常不讨论预处理阶段的耗时，因为预处理是一个一次性过程，它在攻击开始之前就完成了。

这种时空权衡进一步说明了为什么简单地存储哈希口令是错误的做法。我们将在之后讨论针对性的防御措施。

18.4 使字典攻击更难实施

当在服务器上存储弱口令的哈希时，离线字典攻击，尤其是基于预处理的离线字典攻击是一种真正的威胁。本节中，我们将讨论一些针对性的技术，这些技术能够使得字典攻击更加困难。

18.4.1 公共盐

上一节中，我们展示了攻击者如何对字典 $\mathcal{D}$ 进行预处理以构建一个速查表 L，进而可以快速破解一个或多个哈希后的口令。一种叫做**加盐 (salting)** 的简单防御措施就可以反制这种预处理攻击。就算允许攻击者拥有无限时间对 $\mathcal{D}$ 进行预处理，加盐也可以将破解一个哈希口令文件 F 的时间复杂度提升到：

$$\Omega(|\mathcal{D}| \times |F|)$$

id_1	$salt_1$	$H(pw_1, salt_1)$
id_2	$salt_2$	$H(pw_2, salt_2)$
id_3	$salt_3$	$H(pw_3, salt_3)$
⋮	⋮	⋮

图 18.4: 口令文件（版本 2）

换言之，加盐可以确保没有任何其他方法的效率优于基于穷举搜索的暴力破解。

加盐的具体做法是在注册新口令时生成一个称为**盐 (salt)** 的随机字符串。系统中的每个用户都会被分配一个新的盐，我们可以认为盐是从一个集合 $\mathcal{S}$ 中随机选出的。在实践中，往往 $|\mathcal{S}| = 2^{128}$ 就足以满足要求。这个盐会与口令一起被哈希，得到验证密钥 vk。盐也会被存储在服务器的口令表中，如图 18.4 所示。只有服务器需要知道盐值，用户甚至不需要知道有盐的存在。

于是我们就有了新的口令协议，我们称之为**口令协议版本 2**，它的运行方式如下：

- G：令 $pw \xleftarrow{R} \mathcal{P}$，$salt \xleftarrow{R} \mathcal{S}$，计算 $y \leftarrow H(pw, salt)$，输出 $sk = pw$ 和 $vk = (salt, y)$。
- 以 $sk = pw$ 为输入的算法 P 和以 $vk = (salt, y)$ 为输入的算法 V，按如下逻辑交互：
  1. P 将 pw 发送给 V；
  2. 如果收到的 pw 满足 $H(pw, salt) = y$，V 输出 accept，否则输出 reject。

与版本 1 的描述一样，版本 2 的描述也是相当理想化的，因为我们假设口令 pw 是从空间 $\mathcal{P}$ 中随机均匀选出的，但实际情况远非如此。

现在有了盐的存在，对手可以采取两种策略去攻击文件 F 中的哈希口令：

- 第一种策略是使用批量离线字典攻击。问题是，现在预处理阶段需要考虑哈希函数 H 的庞大的输入集合，事实上集合 $\mathcal{D} \times \mathcal{S}$ 中的任何元素都是可能的输入。因此，如果仍然采用式 18.2 中的预处理算法，需要准备的预处理表 L 的规模将达到 $|\mathcal{D}| \times |\mathcal{S}|$。这个大小无论如何都是难以生成的，更不要说出村了。因此，式 18.2 所介绍的预处理方法不再可行。
- 第二种策略是对 F 中的所有口令进行穷举搜索，就像式 18.1 中所做的那样。我们已经说明过，穷举搜索的时间复杂度是

$$O(|\mathcal{D}| \times |F|)$$

为了使得没有任何其他方法的效率优于穷举搜索，即上面的第二种策略，盐值空间 $\mathcal{S}$ 需要尽可能大。即使对手采用时空权衡的方法来对 $\mathcal{D} \times \mathcal{S}$ 进行预处理，也同样应该成立。为了推导对 $\mathcal{S}$ 大小的必要约束，我们首先需要更精确地定义在预处理模型中逆转加盐函数的含义。

带有预处理的加盐单向函数. 为了便于定义，我们将对手 $\mathcal{A}$ 拆分成两个独立的对手 $\mathcal{A}_0$ 和 $\mathcal{A}_1$，其中对手 $\mathcal{A}_0$ 拥有无限运行时间，执行预处理阶段。对手 $\mathcal{A}_1$ 能够高效地进行反转攻击。两个对手之间唯一允许的通信是交换一个 ℓ 比特的字符串 L，其中 L 是预处理阶段的结果。这些概念在下面的定义中得到了体现，该定义将 H 建模为一个随机预言机。

定义 18.3. 假设 H 是一个定义在 $(\mathcal{D} \times \mathcal{S}, \mathcal{Y})$ 上的哈希函数。我们定义对手 $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ 在预处理模型中攻破哈希函数 H 的单向性的优势为 $\mathrm{OWsp}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, H]$，其大小为对手 $\mathcal{A}$ 赢得下面游戏的概率：

id_1	$salt_1$	$H\big(password_1, salt_1, pepper_1\big)$
id_2	$salt_2$	$H\big(password_2, salt_2, pepper_2\big)$
id_3	$salt_3$	$H\big(password_3, salt_3, pepper_3\big)$
$\vdots$	$\vdots$	$\vdots$

图 18.5: 口令文件（版本 3）

- $\mathcal{A}_0$ 向 H 发起查询并输出一个建议字符串 L；
- 挑战者随机选择 $(pw, s) \xleftarrow{R} \mathcal{D} \times \mathcal{S}$，令 $y = H(pw, s)$，并将 (L, y, s) 发送给 $\mathcal{A}_1$；
- $\mathcal{A}_1$ 向 H 发起查询并输出 $pw' \in \mathcal{D}$。当 $H(pw', s) = y$ 时，$\mathcal{A}$ 获胜。

需要注意的是，对手 $\mathcal{A}_1$ 同时获得了 L 和盐 s，而它的目的是找到带有盐 s 的哈希 y 的原象。下面的定理给出了在预处理模型中反转加盐单向函数 H 的时间界限，其中 H 符合随机预言机模型。

定理 18.2. 假设 H 是一个定义在 $(\mathcal{D} \times \mathcal{S}, \mathcal{Y})$ 上的哈希函数，H 符合随机预言机模型，且 $|\mathcal{D}| \leq |\mathcal{Y}|$。令 $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ 是一个定义 *18.3* 中的对手，其中 $\mathcal{A}_0$ 输出一个 ℓ 比特的建议字符串 L，$\mathcal{A}_1$ 最多能向 H 发起 Q_{ro} 次查询，则有：

$$\text{OWsp}^{\text{ro}}\mathsf{adv}[\mathcal{A}, H] \leq O\left(\frac{\ell \cdot Q_{\text{ro}}}{|\mathcal{S}| \cdot |\mathcal{D}|} + \frac{Q_{\text{ro}}}{|\mathcal{D}|}\right) \tag{18.4}$$

该定理表明，如果 $\mathcal{A}$ 在反转 $vk := y \in \mathcal{Y}$ 时有恒定成功概率，比如 0.5，那么他在攻击阶段必须至少花费 $Q_{\text{ro}} \geq \Omega(|\mathcal{D}| \cdot |\mathcal{S}|/\ell)$ 的时长。因此，为了防止预处理带来的破解性能提升，我们应该令 $|\mathcal{S}| \geq \Omega(\ell)$，这会确保不会有其他方法效率优于穷举搜索。比如说，如果我们假设建议字符串 L 的最大存储空间为 2^{80}，那么盐空间应当至少是 $\{0,1\}^{80}$。在实践中，人们通常取 $|\mathcal{S}| := \{0,1\}^{128}$。

从技术上讲，定理 18.2 限定了破解单个口令的时间，但并没有限定批量离线字典攻击的时间。在这种情况下，攻击者可以尝试一次性破解 t 个口令，其中 $t > 1$。然而，我们希望该定理可以被推广到批量攻击中，使得约束 $|\mathcal{S}| \geq \Omega(\ell)$ 即使面对批处理场景仍能够有效防止破解。

加盐的局限性. 虽然加盐可以抵御预处理攻击，但它并不能阻止其他攻击。比如说，即使使用了盐，一个选择了弱口令的用户仍然容易受到式 18.1 中的那种离线字典攻击。在接下来的内容中，我们将展示如何对离线字典攻击提供进一步的保护。

18.4.2 秘密盐

通过向人类口令中添加人工熵，我们可以使对手的破解变得更难。想法是从一个小空间 $\mathcal{S}_{\text{p}}$ 中再随机挑选一个短字符串，称为**秘密盐 (secret salt)** 或者**胡椒 (pepper)**，并将其也纳入到哈希计算中。但是秘密盐不会被记录在口令文件中，因此产生的口令文件如图 18.5 所示。

为了验证一个口令，服务器只需要遍历秘密盐的所有可能值，直至找到一个能与存储的哈希值匹配的值。依此设计的**口令协议（版本 3）**按如下方式运行：

- G: 令 $pw \xleftarrow{R} \mathcal{P}, salt \xleftarrow{R} \mathcal{S}, pepper \xleftarrow{R} \mathcal{S}_p, y \leftarrow H(pw, salt, pepper)$，输出 $sk := pw$ 和 $vk := (salt, y)$。
- 以 $sk = pw$ 为输入的算法 P 及以 $vk = (salt, y)$ 为输入的算法 V，按如下逻辑交互：

1. P 将 pw 发送给 V；

2. 对于收到的 pw，如果存在 $p \in \mathcal{S}_{\mathrm{p}}$ 满足 $H(pw, salt, p) = y$，V 输出 accept，否则输出 reject。

一个典型的设计是将秘密盐空间设置为 $\mathcal{S}_{\mathrm{p}} := \{0,1\}^{12}$。与版本 2 协议相比，这会使得服务器上的口令验证时长变为原来的 4096 倍，但这在现代计算机上仍然只需要不到百分之一秒的时间，因此用户无法察觉到。然而更重要的是，对手现在破解口令文件中的弱口令的难度也增长到了原来的 4096 倍。

秘密盐使离线字典攻击更加困难，因为现在对手必须在 $\mathcal{D}\times\mathcal{S}_{\mathrm{p}}$ 空间中搜索，而不仅仅是 $\mathcal{D}$。这种技术对用户体验的影响很小。秘密盐增加了用户口令的熵，但并没有迫使用户记住更复杂的口令。

18.4.3 慢哈希函数

保护弱口令的另一种方法是使用慢哈希函数。回顾一下，用 SHA256 这样的哈希函数对口令进行哈希是非常快的，因此 SHA256 也使得离线字典攻击可能发动成功。现在假设服务器使用一个更慢的哈希函数对口令进行哈希，比如该函数处理一个输入需要百分之一秒，这比 SHA256 慢一万倍左右。尽管这对用户体验的影响微乎其微，但是对手这时再想要对字典中的所有项进行哈希的工作量就增大了一万倍。

那么我们如何构造一个慢哈希函数呢？一个直观的方法就是把一个快速哈希函数迭代足够多的次数，直至它变得很慢。具体地，对于一个定义在 $(\mathcal{X}, \mathcal{X})$ 上的哈希函数 H，我们现在定义：

$$H^{(d)}(x) = H(H(H(\cdots(x)\cdots))) \tag{18.5}$$

这样就将 H 迭代了 d 次，可参见第 14.3 节。如果 $d = 10000$，那么计算 $H^{(d)}(x)$ 就比计算 $H(x)$ 要多花一万倍的时间。然而这种方法是有问题的，它也不应该被应用到现实的系统中去。原因之一是，函数 $H^{(d)}(x)$ 也比函数 $H(x)$ 容易逆运算 d 倍，参见练习 14.17。我们下面会介绍一个性质更好的函数。首先，我们先对慢哈希函数进行严格的定义。

定义 18.4. *一个基于口令的密钥推导函数 **(password-based key derivation function, PBKDF)** 是这样的一个函数 H，其输入是一个口令 $pw \in \mathcal{P}$，一个空间 $\mathcal{S}$ 中的盐，以及一个难度参数 $d \in \mathbb{Z}^{>0}$；其输出是一个值 $y \in \mathcal{Y}$。我们要求函数 H 能由一个运行时间与 d 成正比的算法有效计算。像往常一样，我们称 PBKDF 定义在 $(\mathcal{P}, \mathcal{S}, \mathcal{Y})$ 上。*

我们会在练习 18.3 中讨论 PBKDF 的安全要求。我们的第一个 PBKDF 的例子，不妨称之为 **PBKDF1**，就基于式 18.5，其定义为：

$$\mathrm{PBKDF1}_H(pw, salt, d) = H^{(d)}(pw, salt)$$

对于定义在 $(\mathcal{X}, \mathcal{X})$ 上的哈希函数 H，PBKDF1 定义在 $(\mathcal{P}, \mathcal{S}, \mathcal{X})$ 上，其中 $\mathcal{X} = \mathcal{P} \times \mathcal{S}$。由于存在练习 14.17 中所介绍的攻击方式，该构造不会被应用在实际的系统中。

18.4.3.1 PBKDF2 函数

一种在实践中被广泛用于构造慢哈希函数的方法被称为 PBKDF2。令 F 是一个定义在 $(\mathcal{P}, \mathcal{X}, \mathcal{X})$ 上的 PRF，其中 $\mathcal{X} := \{0,1\}^n$，那么由之派生出的 PBKDF，不妨用 $\mathrm{PBKDF2}_F$ 表示，它也定义在

$(\mathcal{P}, \mathcal{X}, \mathcal{X})$ 上，并按如下方式工作：

$$\mathrm{PBKDF2}_F(pw, salt, d) := \left\{ \begin{array}{l} x_0 \leftarrow F(pw, salt) \\ \text{for } i = 1, \ldots, d-1: \\ \qquad x_i \leftarrow F(pw, x_{i-1}) \\ \text{output } y \leftarrow x_0 \oplus x_1 \oplus \cdots \oplus x_{d-1} \in \mathcal{X} \end{array} \right\} \tag{18.6}$$

上面的式 18.6 描述了基本版的 PBKDF2，如果需要输出更多比特，对其稍加调整即可。事实上，PBKDF2 能够输出 $\mathcal{X}^b$ 中的一个元素，其中 $1 < b < 2^{32}$，方法是计算：

$$\mathrm{PBKDF2}_F^{(b)}(pw, salt, d) := (\mathrm{PBKDF2}_F(pw, salt_1, d), \ldots, \mathrm{PBKDF2}_F(pw, salt_b, d)) \in \mathcal{X}^b \tag{18.7}$$

其中所有的 b 个盐都是由最初给定的盐派生而来的，方法是令 $salt_i \leftarrow salt \,||\, \mathrm{bin}(i)$。这里的 $\mathrm{bin}(i)$ 是 $i \in \{1, \ldots, b\}$ 的 32 位二进制字符串表示。

在实践中，PBKDF2 通常使用 HMAC-SHA256 作为底层 PRF。难度 d 的设置取决于项目需求和硬件速度。例如，iOS 10 中的备份钥匙包使用 PBKDF2 来保护，其中 d 的值是一千万。在 Windows 10 中，数据保护 API 默认使用 $d = 8000$，但使用 HMAC-SHA512 作为 PRF。

我们会在练习 18.2 和 18.3 中更详细地讨论 PBKDF2 的安全性。

18.4.4 慢内存困难哈希函数

PBKDF2 的一个重要问题是它容易受到并行硬件攻击。我们知道，大部分现代处理器都是基于缓存的，而计算单元只占整个处理器面积的一小部分。因此，一个商用处理器不能对许多输入进行并行 PBKDF2 计算，也不太适合进行离线字典攻击。

更有经验的攻击者通常会在支持高度并行的专用硬件上运行离线字典攻击，比如 GPU，FPGA，甚至是定制芯片。一个定制芯片可以包装超过一百万个 SHA256 计算单元。如果每个单元每秒可以进行一百万次 SHA256 计算，那么一个对手每秒可以在每个芯片上尝试 10^{12} 个口令。即使 PBKDF2 的难度被设置为 $d = 10000$，一个由大约 500 个这样的专用芯片组成的农场也能在不到一天的时间内跑完所有 2^{52} 个 8 位口令。这种攻击之所以能够实现，是因为 SHA256 的硬件实现可以做得相当紧凑，所以大量的 SHA256 计算单元可以打包到一个芯片里。

这表明，我们需要的哈希函数 H 应该要求专用硬件实现也需要庞大的片上面积，这样一个芯片就只能容纳几个计算单元，这就极大程度上降低了定制硬件的性能优势。

那么我们如何建立一个具有较大硬件占用面积的哈希函数 H？一种方法是确保计算 H 的每一步都需要大量的内存，这将迫使攻击者将芯片上的大部分区域分配给单次哈希求值所需的内存，这就确保了每个芯片只能包含少量的计算电路。

需要大量内存的哈希函数被称为**内存困难函数**。已经有一些符合要求的内存难度哈希函数，这些函数能够在随机预言机模型下被严格证明是内存困难的。在我们讨论安全问题之前，让我们先看看一个流行的结构，叫做 **Scrypt**。Scrypt 是由一个内存容易的哈希函数 $h : \mathcal{X} \to \mathcal{X}$，其中 $\mathcal{X} := \{0,1\}^n$。得到的函数记作 Scrypt_h，如图 18.6 所示。

在我们的安全分析中，我们将底层哈希函数 h 当作一个随机预言机。在实践中，函数 h 是由 Salsa 20/8 变换衍生出来的，参见 3.6 节。难度 d 是根据系统的性能需求来设置的。例如我们可以设置一个

input: $x_0 \in \mathcal{X}$, difficulty $d \in \mathbb{Z}^{>0}$

1. for $i = 1, \ldots, d$: $x_i \leftarrow h(x_{i-1})$ // *Then* $x_i = h^{(i)}(x_0)$
2. $y_0 \leftarrow x_d$
3. for $i = 1, \ldots, d$:
4. $j \leftarrow \text{int}(y_{i-1}) \bmod (d+1)$ // $\text{int}(y_{i-1})$ *converts* $y_{i-1} \in \mathcal{X}$ *to an integer*
5. $y_i \leftarrow h(y_{i-1} \oplus x_j)$ // *read random location in the array* $(x_0, \ldots, x_d)$

output $y_d \in \mathcal{X}$

图 18.6: 函数 $\text{Scrypt}_h(x_0, d)$

恰当的 d 使得计算 Scrypt 需要填满整个芯片的存储空间。这会确保计算 Scrypt 不会太慢，但需要大量的内存。

图 18.6 是将 Scrypt 作为一个哈希函数的描述。另一方面，定义在 $(\mathcal{P}, \mathcal{X}, \mathcal{X})$ 上的 Scrypt PBKDF 是基于 Scrypt 哈希构建的，其工作原理如下：

$$\text{ScryptPBKDF}_h(pw, salt, d) := \left\{ \begin{array}{l} x_0 \leftarrow \text{PBKDF2}_F(pw, salt, 1) \\ y \leftarrow \text{Scrypt}_h(x_0, d) \\ \text{output } \text{PBKDF2}_F(pw, y, 1) \end{array} \right\} \tag{18.8}$$

其中 F 是一个定义在 $(\mathcal{P}, \mathcal{X}, \mathcal{X})$ 上的 PRF。在实践中，我们通常使用 HMAC-SHA256 作为 F。如有需要，我们也可多次迭代 Scrypt，以使其速度更慢而不增加所需的内存。同样地，通过调整最后一行的 PBKDF2 的应用，它可以在 $b > 1$ 时输出 $\mathcal{X}^b$ 中的一个元素，就像式 18.7 那样。

Scrypt 是内存困难的吗? 通过存储 $\mathcal{X}$ 中的 $d+1$ 个元素，Scrypt 函数可以在 $O(d)$ 时间内被计算。Scrypt 哈希的构建步骤创建了一个大小为 $d+1$ 的数组 $(x_0, \ldots, x_d)$，随后重复地从该数组中的随机位置读取数据。正因此，一个想要在 $O(d)$ 时间内完成计算的算法必须在内存中保存整个数组 $(x_0, \ldots, x_d)$。但这个结论尚需严格证明。

危险的是，时空权衡可能使攻击者尝试牺牲更多的攻击时间来换取更少的内存消耗。这将是毁灭性的，因为减少的内存将允许攻击者将许多 Scrypt 计算单元打包到一个芯片中，而这正是我们想要极力避免的。

在练习 18.6 中，我们设计了一种针对 Scrypt 的简单的时空权衡。它表明对于任意的 $1 < \alpha < d/2$，只需要存储 $\lceil d/\alpha \rceil$ 个 $\mathcal{X}$ 中的元素，就可以在 $O(\alpha d)$ 的时间内完成对 Scrypt 的计算。特别地，使用恒定空间的 Scrypt 可以在 $O(d^2)$ 时间内完成计算。然而这种类型的时空权衡并不能帮助对手。对手需要将 α 倍的 Scrypt 计算单元装入一个芯片中，但每个计算单元的工作难度也变为原来的 α 倍。因此与图 18.6 中的 Scrypt 实现相比，单个芯片的整体吞吐量并没有变化。尽管如此，我们仍然需要证明没有比 Scrypt 更好的时空权衡方案。

流水线是对内存难度的另一个威胁。假设存在一个算法，它在计算 Scrypt 时仅仅在算法的其中几个步骤中需要使用 $O(d)$ 的内存。如果算法在其余步骤中仅需要使用常数级的内存空间，那么攻击者就可以通过构建流水线来排列多个 Scrypt 计算单元，这些计算单元之间共享一个 $O(d)$ 大小的存储。这样，每个计算单元仅在它需要 $O(d)$ 存储的几个步骤中调用内存，在其他时间可以把存储释放给其他计算单元使用。这样一来，攻击者仍然可以把许多计算单元打包到一个芯片中，只需要共享一个 $O(d)$ 大小的存储阵列即可。为了防止这种形式的流水线，我们还需要证明每一个 Scrypt 实现都需要在许多计

算步骤中使用到 $O(d)$ 级别的内存。

Scrypt 是内存困难的. 为了证明 Scrypt 是内存困难的，我们首先需要定义一个安全模型来分析上面讨论的几种障碍。我们下面定义了一个抽象的并行随机预言机模型，其中的算法 $\mathcal{A}$ 能够并行查询多个输入在随机预言机 $h : \mathcal{Y} \to \mathcal{Z}$ 上的输出。

一个**并行随机预言机算法** $\mathcal{A}$ 将一个 $x \in \mathcal{X}$ 作为输入，并进行一系列的状态转换。在每个状态下，算法 $\mathcal{A}$ 都会向随机预言机 h 发出一组查询，算法在收到查询的所有响应后就会转入下一状态。该过程将不断重复，直至算法终止，此时算法到达最终状态，其中包含算法的输出。所有中间状态都需要被记录，以便跟踪中间状态的大小。

形式上说，算法 $\mathcal{A}$ 实现了一个确定性映射：

$$\mathcal{A} : \mathcal{X} \times \mathcal{S} \times \mathcal{Z}^{\leq p} \to \mathcal{S} \times \mathcal{Y}^{\leq p}$$

其中 p 是正整数，其工作流程如下：

- $\mathcal{A}$ 被作为 $\mathcal{A}(x, \varepsilon, \varepsilon)$ 调用，它输出一个 $\mathcal{S} \times \mathcal{Y}^{\leq p}$ 中的元组 $(s_1, \overline{y}_1)$。这里，s_1 是 $\mathcal{A}$ 的当前状态，$\overline{y}_1 = (y_1, \ldots, y_r)$ 是它对随机预言机 $h : \mathcal{Y} \to \mathcal{Z}$ 所发出的第一组并行查询。
- 对于 $i = 1, \ldots, t$，当 $\mathcal{A}$ 输出 $(s_i, \overline{y}_i)$，$\overline{y}_1 = (y_1, \ldots, y_r) \in \mathcal{Y}^{\leq p}$ 时，我们进行如下操作：
  - 并行地调用随机预言机 h，计算 $\overline{z}_i \leftarrow (h(y_1), \ldots, h(y_r))$，并且
  - 重新调用 $\mathcal{A}$ 并计算 $(s_{i+1}, \overline{y}_{i+1}) \leftarrow \mathcal{A}(x, s_i, \overline{z}_i)$。
- 最终 $\mathcal{A}$ 输出 (s, ε)，这表示算法终止，输出值为 s。

算法 $\mathcal{A}$ 在输入 $x \in \mathcal{X}$ 上的运行时间就是 $\mathcal{A}$ 在终止之前被调用的总次数。以这种方式衡量运行时间，我们可以捕捉到一个事实，即硬件实现可以在多点上并行地计算哈希函数 h。

我们现在记第 i 步输入 $\mathcal{A}$ 的数据为 $st_i := (s_i, \overline{y}_i)$，并称 st_i 为 i 时刻的输入状态。对于 $s \in \mathcal{S}$，我们令 $|s|$ 表示 s 的比特位数，类似地，我们令 $|z|$ 表示 $z \in \mathcal{Z}$ 的长度。对于 $\overline{z} = (z_1, \ldots, z_r) \in \mathcal{Z}^{\leq p}$，我们令 $|\overline{z}| := \sum_{j=1}^{r} |z_i|$。那么当 $\mathcal{Z} = \{0,1\}^n$ 时，我们有 $|\overline{z}| = rn$。最后，我们将输入状态 $st = (s, \overline{z})$ 的长度定义为 $|st| := |s| + |\overline{z}|$。

定义 18.5. *令 $\mathcal{A}$ 是一个以 $\mathcal{X}$ 中元素作为输入的并行随机预言机算法。记 $\mathcal{A}$ 对于随机预言机 $h : \mathcal{Y} \to \mathcal{Z}$ 和 $x \in \mathcal{X}$ 的累计存储复杂度为 $\mathrm{mem}[\mathcal{A}, h, x]$，其定义是：*

$$\mathrm{mem}[\mathcal{A}, h, x] := \sum_{i=1}^{t} |st_i|$$

图 18.6 中对于预言机 $h : \mathcal{X} \to \mathcal{X}$，$\mathcal{X} = \{0,1\}^n$ 计算 $\mathrm{Scrypt}_h(x, d)$ 的算法的累计存储复杂度是 $O(nd^2)$。下面的定理将表示，这种设计是最优的。

定理 18.3. *令 $\mathcal{X} = \{0,1\}^n$ 是一个满足 $|\mathcal{X}|$ 为超多项式的空间，d 为一个使得 2^{-d} 可以忽略不计的值。对于所有的并行随机预言机算法 $\mathcal{A}$ 和所有的 $x \in \mathcal{X}$，都有：*

$$\Pr\left[\mathcal{A}(x, d) = \mathrm{Scrypt}_h(x, d)\right] \leq \Pr\left[\mathrm{mem}[\mathcal{A}, h, (x, d)] \geq \Omega(d^2 n)\right] + \delta$$

其中 δ 是一个可以忽略的值。上面的两个概率都基于随机预言机 $h : \mathcal{X} \to \mathcal{X}$ 的选择。

该定理表明，如果 $\mathcal{A}(x,d)$ 以接近 1 的概率输出 $\text{Scrypt}_h(x,d)$，那么对于几乎所有的 h 的选择，$\mathcal{A}$ 的累计存储复杂度一定是 $\Omega(d^2n)$ 级的。这表明不可能存在一个针对 Scrypt 的时空权衡策略使得其表现比练习 18.6 更好。如果一个算法以最大 dn/α 的存储空间计算 Scrypt，其中 $\alpha > 1$，那么它的运行时间必然是 $\Omega(d\alpha)$。否则其累计存储复杂度将突破下限。

同样地，不可能存在对 Scrypt 的流水线式攻击。任何在 $O(d)$ 时间内运行的计算 Scrypt 的可行算法必须在整个算法中使用 $\Omega(d\alpha)$ 的内存。否则其累计存储复杂度也将突破下限。

从技术上讲，定理 18.3 限定了在单一输入下计算 Scrypt 所需的时间和空间。它并没有限定批量离线字典攻击的时间，此时攻击者试图一次对 $p > 1$ 个口令计算 Scrypt。然而，我们期望该定理可以被推广到批量的设置中：如果一个算法 $\mathcal{A}$ 对 p 个输入正确地计算了 Scrypt 的概率接近 1，那么 $\mathcal{A}$ 的累计存储复杂度一定是 $\Omega(d^2np)$。这将表明，在 p 个点上计算 Scrypt 时，不存在针对 Scrypt 的时空权衡或流水线攻击。

18.4.4.1 模糊口令内存困难函数

虽然 Scrypt 是一个健全的内存困难的口令哈希函数，但它很容易受到第 4.3.2 节中讨论的那种侧信道道攻击。

考虑一个登录服务器，其中一个运行中的进程 P 通过 Scrypt 哈希并验证用户的口令。假设对手获得了对该服务器的低权限访问，对手可以在服务器上运行用户级程序但不能破坏进程 P，也不能观察到用户的口令明文。然而，利用这种权限，它仍然可以发起一个名为**缓存计时攻击 (cache timing attack)** 的巧妙攻击方法，这种攻击能够让攻击者了解到 P 访问内存页的顺序。注意，攻击者并不能知道内存页中存储的内容，他只能获取 P 读取页的顺序。

现在，假设对手捕获了一个哈希值 y，它是将式 18.8 中介绍的 Scrypt PBKDF 应用于某个带有公共盐的口令 pw 的结果。通常情况下，对手需要发起一个字典攻击，每次尝试都需要大量的时间和内存。然而如果对手掌握了进程 P 在计算 pw 的 Scrypt 哈希时的的内存访问模式，他就可以用很少的内存对 pw 进行字典攻击。

为了说明如何实现攻击，让我们回顾一下图 18.6 中所介绍的 Scrypt 的实现。算法在第一次执行第 5 步时，需要从数组 $(x_0,\ldots,x_d)$ 读取 j 号元素，其中 $j = \text{int}(y_0) \mod (d+1)$。

通过观察 P 对内存的访问，对手可以看到算法第一次执行第 5 步时读取了哪一页的内存，这就给了对手一个 j 的近似值 j_a。由于一个内存页中可能存储了多个数组单元，所以对手并没有办法获取 j 的精确值。尽管如此，这个 j_a 也足以使用有限的内存来测试一个可能的口令 pw'，方法是：

1. 按式 18.8 计算 $x'_0 \leftarrow \text{PBKDF2}_F(pw', salt, 1)$，
2. 如图 18.6 那样计算 y'_0，但不存储任何中间值，然后
3. 测试 $j' \leftarrow \text{int}(y'_0) \mod (d+1)$ 是否与 j_a 接近。

如果测试失败，那么 pw' 就不是正确的口令。这个过程让对手用很少的内存就能过滤掉字典中的大部分候选口令。这样，用户的口令又变得容易受到硬件攻击。

一个解决方案. 这种攻击之所以有效，是因为 Scrypt 的内存访问模式取决于用户的口令。如果我们有一个可证明安全的内存困难哈希函数，其内存访问模式与用户的口令无关，那就更好了。它仍然可以依赖于用户的盐，因为盐不是秘密。这样的函数被称为**数据模糊内存难度函数**。这种函数的一个例子是

Argon2i B，它与 Scrypt 密切相关，但其第一部分的内存访问模式与口令无关，这就消解了上面描述的侧信道攻击。

慢哈希与秘密盐. 总结本小节，我们观察到秘密盐方法和慢哈希方法都会增加对手的工作负荷。人们可以使用其中的一种方法，但不能两者兼用。慢速内存难度哈希方法的主要好处是，它使定制的硬件攻击难以实施。与快速哈希函数一起使用的秘密盐并不能防止并行硬件攻击，因此慢内存困难哈希比秘密盐更可取。

18.4.5 其他口令管理问题

共同口令问题 (common password problem). 用户经常在多台机器和多个网站上拥有账户。理想情况下，所有这些服务器都会采取适当的预防措施来防止对手获得口令文件，并对口令进行适当的加盐和哈希，以降低对手获得该文件时的损失。不幸的是，低安全性服务器的设计者可能不会采取与高安全性服务器一样的安全预防措施。这种低安全性的服务器可能更容易被入侵。此外，这种低安全性的服务器可能会存储没有盐的口令哈希，这使批量字典攻击成为可能，并将检索出所有的弱口令。更糟糕的是，有些服务器甚至会在明处存储哈希值，这样对手就会检索出所有的口令，甚至是强口令。因此对手可以侵入一个低安全级别的服务器，并在该服务器上检索到一些，甚至所有的用户 ID/口令，而且这些口令中的一些也很有可能被用在高安全级别的服务中。因此，尽管在高安全性的服务器上采取了所有的预防措施，但该服务器的安全性可能会被一些完全不相关的、低安全性的服务器的不良安全性所破坏。这个问题被称为**共同口令问题**。

解决共同口令问题的一个标准方案是安装客户端软件，它能将通用口令转换为独有的网站口令，也就是个" 客户端盐"。令 H 是一个哈希函数，如果用户名为 id 的用户的口令是 pw，这个口令将在登录时发送给服务器 id_{server}。用户的客户端或浏览器在发送前会将口令 pw 转换为 $\widehat{pw} := H(pw, id, id_{\text{server}})$，然后将 $\widehat{pw}$ 发送给服务器。这样，从服务器的角度来看，用户的口令是 $\widehat{pw}$，而从用户的角度来看，口令仍然是 pw。这种技术可以保证，即使该用户在许多服务器上使用同一个口令，也不会受到那些没有正确加盐和哈希的服务器的影响。

生物识别技术 (biometrics). 基于口令的认证的最大困难是用户往往会忘记他们的口令。所有支持电话中的很大一部分都与口令相关的问题有关。因此，一些已部署的系统试图用人类生物识别技术来取代口令，如指纹、视网膜扫描、面容识别和许多其他技术。人们甚至可以使用击键动态，即击键之间的时间长度和按下一个键的时长作为一种生物识别特征。这个想法是使用生物特征作为用户的口令。

虽然生物识别技术相比口令而言有明显的好处，例如用户不会忘记他的指纹，但它有两个明显的缺点:

- 生物识别技术通常不是机密的，人们会在他们接触的几乎任何东西上留下他们的指纹;
- 生物识别技术是不可逆的，一旦生物识别特征被盗，用户就无法追索。

因此，生物识别技术不应该被用作识别用户的唯一手段，只适合作为额外的识别手段（有时被称为第二因素认证）来提高安全性。

18.5 一次性口令：针对窃听攻击的安全性

18.5.1 挑战-应答协议

18.5.1.1 用公钥 vk 应答挑战

18.6 挑战-应答：针对主动攻击的安全性

18.7 一个有趣的应用：彩虹表

18.8 一个有趣的应用：强化口令存储

18.9 笔记

18.10 练习

第十九章　基于 Sigma 协议的身份识别与签名

在上一章中，我们研究了身份识别协议。特别是在 18.5.1.1 小节中，我们展示了如何使用安全的签名方案来建立一个挑战-应答身份识别方案，该方案提供了最高级别的安全，即针对主动攻击的安全（见定义 18.8）。在本章中，我们考虑相反的方向。

首先，我们使用一种完全不同的技术开发了一种新的身份识别协议，它实现了对窃听攻击的安全性（见定义 18.6）。这个协议本身就很有意义，因为它相当优雅，而且可以在 DL 假设下被证明是安全的。

其次，我们展示了如何将该协议转化为一个非常有效的签名方案，即 **Schnorr 签名方案**。在 DL 假设下，该方案在随机预言机模型下是安全的。

第三，我们归纳这些技术并引入了 **Sigma 协议**的概念。我们利用这些更普适的技术开发了几个新的身份识别协议和签名方案。

在下一章中，我们将这些技术用于更高级的用途，设计出允许一方向另一方证明某些事实是真实的协议，同时不透露任何不必要的信息。例如，我们将展示如何证明加密值 m 位于某个范围内而不透露关于 m 的任何其他信息。

19.1　Schnorr 身份识别协议

我们首先描述一种称为 **Schnorr 身份识别**的安全的身份识别协议。如果假定离散对数问题是困难的，那么该协议对窃听攻击是安全的。

令 $\mathbb{G}$ 是一个 q 阶循环群，其中 q 是素数，$g \in \mathbb{G}$ 是一个生成元。假设证明者 P 有一个私钥 $\alpha \in \mathbb{Z}_q$，且与之对应的公钥为 $u = g^\alpha \in \mathbb{G}$。为了向验证者 V 证明自己的身份，P 想让 V 相信自己知道 α。最简单的方法是 P 直接将 α 发送给 V。这个方法其实就是我们在 18.3 中讨论过的口令协议版本 1，此时函数 $H(\alpha) := g^\alpha$ 扮演单向函数的角色。尽管这个协议提供了针对直接攻击的安全性，但它对窃听攻击是完全不安全的。相对，Schnorr 协议是一个经过巧妙设计的交互式协议，它允许 P 说服 V 相信自己知道以 g 为基数的离散对数 u，但又并不需要真的将这个值发送给 V。

下面是它的工作原理。令 $\mathcal{C} \subset \mathbb{Z}_q$，则 Schnorr 身份识别协议 $\mathcal{I}_{\text{sch}} = (G, P, V)$ 的原理如下：

- 密钥生成算法 G 按如下方式运行：

$$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g^\alpha$$

  验证密钥为 $vk := u$，私钥为 $sk := \alpha$。

- P 与 V 之间的协议按照下面的方式运行。其中证明者 P 由 $sk = \alpha$ 初始化，验证者 V 由 $vk = u$ 初始化：

  1. P 计算 $\alpha_{\text{t}} \xleftarrow{\text{R}} \mathbb{Z}_q$，$u_{\text{t}} \leftarrow g^{\alpha_{\text{t}}}$，并将 u_{t} 发送给 V；

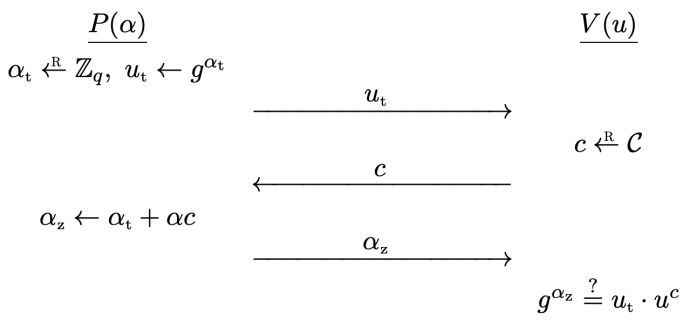


图 19.1: Schnorr 身份识别协议

2. V 计算 $c \xleftarrow{R} \mathcal{C}$，并将 c 发送给 P；
3. P 计算 $\alpha_z \leftarrow \alpha_t + \alpha c \in \mathbb{Z}_q$，并将 α_z 发送给 V；
4. V 检查 $g^{\alpha_z} = u_t \cdot u^c$ 是否成立，如果成立就输出 accept，否则输出 reject。

图 19.1 展示了该协议的工作流程。

$P(\alpha)$ 和 $V(u)$ 之间的一次交互产生一个**对话 (conversation)** $(u_t, c, \alpha_z) \in \mathbb{G} \times \mathcal{C} \times \mathbb{Z}_q$。如果 V 的检查通过，即 $g^{\alpha_z} = u_t \cdot u^c$ 成立，我们就称该对话为 u **的接受对话 (accepting conversation for** u**)**。不难看出，P 和 V 之间的交互总是能够产生一个接受对话，因为如果有 $u_t = g^{\alpha_t}$ 且 $\alpha_z = \alpha_t + \alpha c$，则有：

$$g^{\alpha_z} = g^{\alpha_t + \alpha c} = g^{\alpha_t} \cdot (g^{\alpha})^c = u_t \cdot u^c$$

因此，Schnorr 协议能够满足身份识别协议所必需的基本正确性要求。

我们称集合 $\mathcal{C}$ 为**挑战空间 (challenge space)**。为了满足安全需求，我们要求 $|\mathcal{C}|$ 是超多项式的。事实上我们可以简单地取 $\mathcal{C}$ 为 $\mathbb{Z}_q$，但是技术上也可以允许更小的挑战空间。尽管我们最终会证明 Schnorr 协议在 DL 假设下对窃听攻击是安全的，但我们现在先从一个更简单的定理开始，它只证明了 Schnorr 协议对直接攻击的安全性（攻击游戏 18.1）。

在证明这一点时，我们将表明，任何能够以不可忽略的概率成功执行直接仿冒攻击的对手，都能被转换成一个从验证密钥 u 中高效地恢复私钥 α 的算法。由于这个原因，Schnorr 的协议有时也被称为离散对数的知识证明。

定理 19.1. *基于 $\mathbb{G}$ 上的离散对数假设，假设 $N := |\mathcal{C}|$ 是超多项式的，Schnorr 身份认证协议对直接攻击是安全的。*

特别地，假设 $\mathcal{A}$ 是一个有效冒充对手，他通过攻击游戏 18.1 中的直接攻击方式攻击 $\mathcal{I}_{\text{sch}}$ 的优势为 $\epsilon := \text{ID1adv}[\mathcal{A}, \mathcal{I}_{\text{sch}}]$。那么必然存在一个有效的离散对数对手 $\mathcal{B}$，其运行时间大致是 $\mathcal{A}$ 的两倍，其优势为 $\epsilon' := \text{DLadv}[\mathcal{B}, \mathbb{G}]$，并且有：

$$\epsilon' \geq \epsilon^2 - \frac{\epsilon}{N} \tag{19.1}$$

也即：

$$\epsilon \leq \frac{1}{N} + \sqrt{\epsilon'} \tag{19.2}$$

证明思路. 假设攻击游戏 18.1 中的对手 $\mathcal{A}$ 在攻击 $\mathcal{I}_{\text{sch}}$ 时有优势 ϵ。在该游戏中，挑战者生成验证密钥 $u = g^{\alpha}$。在对手 $\mathcal{A}$ 的冒充尝试中，他基于某个任意的对手策略生成了协议的第一条交互信息 u_{t}。为了赢得游戏，对手 $\mathcal{A}$ 必须用一个有效应答 α_{z} 来响应随机挑战 $c \in \mathcal{C}$，使得 $g^{\alpha_{\text{z}}} = u_{\text{t}} \cdot u^{c}$。直观地讲，如果 $\mathcal{A}$ 能以概率 ϵ 对一个这样的随机挑战产生一个有效应答，那么他就应该能以概率 ϵ^2 对两个随机挑战产生一个有效的应答。要使这一直觉变得严谨，需要一个有点技术性的论证，我们将在随后的引理中给出。

下面，我们先介绍如何使用 $\mathcal{A}$ 计算随机数 $u \in \mathbb{G}$ 的离散对数。我们使用 u 作为 $\mathcal{I}_{\text{sch}}$ 的验证密钥，并让 $\mathcal{A}$ 生成协议的第一条交互 u_{t}。然后我们向 $\mathcal{A}$ 提供一个随机挑战 c，并希望 $\mathcal{A}$ 产生一个有效应答 α_{z}。如果这确实发生了，我们就将 $\mathcal{A}$ 的内部状态 “回溯” 到它刚刚生成 u_{t} 后的那一刻，然后给 $\mathcal{A}$ 提供另一个随机挑战 c'，并希望 $\mathcal{A}$ 产生另一个有效应答 α'_{z}。

如果上面这些都发生了，那么对于一个给定的验证密钥 u，我们就得到了两个接受对话 $(u_{\text{t}}, c, \alpha_{\text{z}})$ 和 $(u_{\text{t}}, c', \alpha'_{\text{z}})$，且这两个对话的第一条交互 u_{t} 相同。此外，由于 $\mathcal{C}$ 是超多项式的，因此我们可以以压倒性的概率得到 $c' \neq c$ 。基于以上信息，我们可以很容易地计算出 $\mathsf{Dlog}_g u$。事实上，由于上面的两个对话都是接受对话，我们有下面两个等式：

$$g^{\alpha_{\text{z}}} = u_{\text{t}} \cdot u^{c}, \quad g^{\alpha'_{\text{z}}} = u_{\text{t}} \cdot u^{c'}$$

用第一式除以第二式，u_{t} 就被抵消了，于是我们有：

$$g^{\Delta\alpha} = u^{\Delta c} \tag{19.3}$$

其中 $\Delta\alpha := \alpha_{\text{z}} - \alpha'_{\text{z}}$，$\Delta c := c - c'$。由于 $\Delta c \neq 0$，且循环群 $\mathbb{G}$ 的阶 q 是素数，所以倒数 $1/\Delta c$ 存在于 $\mathbb{Z}_q$ 中。我们可以将式 19.3 中等式的左右两边都升阶 $1/\Delta c$，于是有：

$$g^{\Delta\alpha/\Delta c} = u$$

因此，我们可以有效计算离散对数 $\mathsf{Dlog}_g u$，其值为 $\Delta\alpha/\Delta c$。

读者应该注意到，这里提出的从两个接受对话中提取离散对数的技术，与事实 10.3 中使用的想法基本相同。事实上，使用 10.6.1 节中的术语，我们可以看到 $(\alpha_{\text{z}}, -c)$ 和 $(\alpha'_{\text{z}}, -c')$ 是 u_{t} 相对于 g 和 u 的不同表示。事实 10.3 告诉了我们如何从这两个表示中计算出 $\mathsf{Dlog}_g u$。 □

这个定理与我们在本文中迄今为止所介绍的所有其他安全定理有着质的不同。事实上，在这个定理的证明中，虽然我们表明每个攻击 $\mathcal{I}_{\text{sch}}$ 的对手 $\mathcal{A}$ 都可以转化为破解离散对数问题的对手 $\mathcal{B}$，但我们构造的对手 $\mathcal{B}$ 并不是 $\mathcal{A}$ 的基本包装器，因为对手 $\mathcal{B}$ 要运行 $\mathcal{A}$ 两次。此外，这个定理在量上也是不同的，因为安全规约根本就不是很严密：如果 $\mathcal{A}$ 以概率 ϵ 获胜，那么 $\mathcal{B}$ 只能保证以 $\approx \epsilon^2$ 的概率获胜。

因此，为了使上述想法更严谨，我们需要先引入下面的引理：

引理 19.2 (回溯引理). 令 S 和 T 是两个非空有限集，$f : S \times T \to \{0,1\}$ 是一个函数。令 X、Y 和 Y' 是相互独立的随机变量，其中 X 在集合 S 中取值，Y 和 Y' 都在 T 上均匀分布。令 $\epsilon := \Pr[f(\mathsf{X}, \mathsf{Y}) = 1]$，$N := |T|$，则有：

$$\Pr[f(\mathsf{X}, \mathsf{Y}) = 1 \wedge f(\mathsf{X}, \mathsf{Y}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'] \geq \epsilon^2 - \frac{\epsilon}{N}$$

证明. 对于每个 $s \in S$，令 $g(s) := \Pr[f(s,\mathsf{Y}) = 1]$。首先，我们可以观察到 $E[g(\mathsf{X})] = \epsilon$，这是因为：

$$\begin{aligned}
E[g(\mathsf{X})] &= \sum_{s\in S} g(s)\Pr[\mathsf{X}=s] = \sum_{s\in S}\Pr[f(s,\mathsf{Y})=1]\Pr[\mathsf{X}=s] \\
&= \sum_{s\in S}\Pr[f(s,\mathsf{Y})=1 \wedge \mathsf{X}=s] \quad (\text{独立性}) \\
&= \sum_{s\in S}\Pr[f(\mathsf{X},\mathsf{Y})=1 \wedge \mathsf{X}=s] \\
&= \Pr[f(\mathsf{X},\mathsf{Y})=1] \quad (\text{全概率}) \\
&= \epsilon
\end{aligned}$$

其次，考虑一个固定的 $s\ S$，令 $\mathcal{U}_s$ 为 $f(s,\mathsf{Y}) = 1 \wedge f(s,\mathsf{Y}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'$ 成立的事件，我们称：

$$\Pr[\mathcal{U}_s] = g(s)^2 - \frac{g(s)}{N}$$

为了证明这一点，令 N_s 是满足 $f(s,t) = 1$ 的 $t \in T$ 的数量，那么有 N_s 种方法可以选择满足 $f(s,\mathsf{Y}) = 1$ 的 Y。而对于每个 Y 的选择，有 $N_s - 1$ 种方法可以选择满足 $f(s,\mathsf{Y}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'$ 的 Y'。由于 $g(s) = N_s/N$，因此我们有：

$$\Pr[\mathcal{U}_s] = \frac{N_s(N_s-1)}{N^2} = \frac{N_s^2}{N^2} - \frac{N_s}{N^2} = g(s)^2 - \frac{g(s)}{N}$$

最后，令 $\mathcal{U}$ 为 $f(\mathsf{X},\mathsf{Y}) = 1 \wedge f(\mathsf{X},\mathsf{Y}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'$ 成立的事件，我们有：

$$\begin{aligned}
\Pr[\mathcal{U}] &= \sum_{s\in S}\Pr[\mathcal{U} \wedge \mathsf{X}=s] \quad (\text{全概率}) \\
&= \sum_{s\in S}\Pr[f(s,\mathsf{Y})=1 \wedge f(s,\mathsf{Y}')=1 \wedge \mathsf{Y}\neq\mathsf{Y}' \wedge \mathsf{X}=s] \\
&= \sum_{s\in S}\Pr[f(s,\mathsf{Y})=1 \wedge f(s,\mathsf{Y}')=1 \wedge \mathsf{Y}\neq\mathsf{Y}'] \cdot \Pr[\mathsf{X}=s] \quad (\text{独立性}) \\
&= \sum_{s\in S}\Pr[\mathcal{U}_s]\cdot\Pr[\mathsf{X}=s] = \sum_{s\in S}[g(s)^2 - \frac{g(s)}{N}]\cdot\Pr[\mathsf{X}=s] \\
&= E[g(\mathsf{X})^2] - \frac{g(\mathsf{X})}{N} \\
&\geq E[g(\mathsf{X})]^2 - \frac{E[g(s)]}{N} = \epsilon^2 - \frac{\epsilon}{N}
\end{aligned}$$

上面我们使用了这样一个一般事实：对于任意随机变量 Z 都有 $E[\mathsf{Z}^2] \geq E[\mathsf{Z}]^2$，在这里我们的这个 $\mathsf{Z} = g(\mathsf{X})$。这是詹森不等式的一个特例。 □

定理 *19.1* 的证明. 利用具有优势 ϵ 的冒充对手 $\mathcal{A}$，我们能够建立一个具有优势 ϵ' 的 DL 对手 $\mathcal{B}$，如下所示。对手 $\mathcal{B}$ 从其挑战者那里得到一个 DL 问题实例 $u = g^\alpha$，而我们的目标是让 $\mathcal{B}$ 在 $\mathcal{A}$ 的帮助下计算出 α。$\mathcal{B}$ 的计算过程可以分为两个阶段。

在第一阶段，$\mathcal{B}$ 扮演 $\mathcal{A}$ 的挑战者的角色，将 u 的值交给 $\mathcal{A}$ 作为验证公钥。在这一步中，$\mathcal{B}$ 的目标是计算出两个具有不同挑战的 u 的接受对话，即：

$$(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}}),\ \ (u_{\mathrm{t}}, c', \alpha'_{\mathrm{z}})$$

其中：

$$g^{\alpha_z} = u_t \cdot u^c, \quad g^{\alpha'_z} = u_t \cdot u^{c'}, \quad c \neq c'$$

以下是 $\mathcal{B}$ 做到这一点的方法：

1. $\mathcal{A}$ 扮演证明者，并将 u_t 发送给扮演验证者的 $\mathcal{B}$；
2. $\mathcal{B}$ 向 $\mathcal{A}$ 发送一个随机数 $c \in \mathcal{C}$；
3. $\mathcal{A}$ 向 $\mathcal{B}$ 发送 α_z；
4. $\mathcal{B}$ 对 $\mathcal{A}$ 进行 “回溯” 操作，使 $\mathcal{A}$ 的内部状态与步骤 1 结束时完全相同，然后 $\mathcal{B}$ 向 $\mathcal{A}$ 发送一个新的随机数 $c' \in \mathcal{C}$；
5. $\mathcal{A}$ 向 $\mathcal{B}$ 发送 α'_z。

下面我们应用回溯引理 19.2。在该引理中，随机变量 Y 与挑战 c 对应，Y' 与挑战 c' 对应，X 与 $\mathcal{A}$、$\mathcal{B}$ 和 $\mathcal{B}$ 的挑战者的所有其他随机选择对应，具体包括群 $\mathbb{G}$ 和群元素 $g, u, u_t \in \mathbb{G}$。当产生的对话是 u 的一个接受对话时，引理中的函数 f 为 1，否则就为 0。因此，如果 (u_t, c, α_z) 是 u 的接受对话，则 $f(\mathsf{X}, \mathsf{Y}) = 1$，如果 (u_t, c', α'_z) 是 u 的接受性对话，则 $f(\mathsf{X}, \mathsf{Y}') = 1$。应用该引理，我们发现 $\mathcal{B}$ 得到两个不同挑战的接受对话的概率至少为 $\epsilon^2 - \epsilon/N$。

所以现在假设 $\mathcal{B}$ 已经成功计算出两个这样的接受对话 (u_t, c, α_z) 和 (u_t, c', α'_z)。在第二阶段，$\mathcal{B}$ 用这两个对话来计算 α。事实上，正如在上面的证明思路中已经讨论过的，我们可以计算 $\alpha = \Delta\alpha/\Delta c$，其中 $\Delta\alpha := \alpha_z - \alpha'_z$，$\Delta c := c - c'$。

这就得到了式 19.1。下面我们论证式 19.2 可由式 19.1 推出。为此，我们不妨假设 $\epsilon \geq 1/N$，否则式 19.2 显然是成立的。因此我们有：

$$\begin{aligned}(\epsilon - \frac{1}{N})^2 &= \epsilon^2 - \frac{2\epsilon}{N} + \frac{1}{N^2} \leq \epsilon^2 - \frac{2\epsilon}{N} + \frac{\epsilon}{N} \quad (\text{因为 } \epsilon \geq \frac{1}{N}) \\ &= \epsilon^2 - \frac{\epsilon}{N} \leq \epsilon' \quad (\text{式 19.1})\end{aligned}$$

因此式 19.2 得证。

回顾一下，我们展示了如何从恶意验证者 $\mathcal{A}$ 中有效地*提取*私钥 α，进而证明了针对直接攻击的安全性。这使我们能够利用恶意验证者来解决 $\mathbb{G}$ 中的离散对数问题。我们的*提取器*通过回溯证明者的状态来获得两个接受对话 (u_t, c, α_z) 和 (u_t, c', α'_z)，其中 $c \neq c'$。在安全性证明中回溯证明者 $\mathcal{A}$ 是可能的，因为我们完全控制了 $\mathcal{A}$ 的执行环境。在现实世界中，由于不能对诚实证明 P 进行回溯，攻击者就不能使用这种策略从 P 那里提取私钥。 □

19.1.1 诚实验证者零知识和针对窃听的安全性

我们已经证明了在离散对数假设下，Schnorr 协议对*直接*攻击是安全的。事实上在相同的假设下，Schnorr 协议对于*窃听*攻击也是安全的。现在，在某次窃听攻击中，对手获得了 vk 和一张记录列表，该表的内容是 P（在输入 sk 上）与 V（在输入 vk 上）之间的若干对话。我们的想法是证明这些对话对于对手来说没有任何帮助，因为在给定 vk 而没有给定 sk 的情况下，对手也可以自己有效地生成这些对话。如果我们能证明这一点，我们也就大功告成了。事实上，假设 $\mathcal{A}$ 是一个对手，他通过窃听成功发动仿冒攻击的优势是不可忽略不计的。然后，我们用另一个对手 $\mathcal{B}$ 来替代 $\mathcal{A}$，它的工作方式与 $\mathcal{A}$ 是一

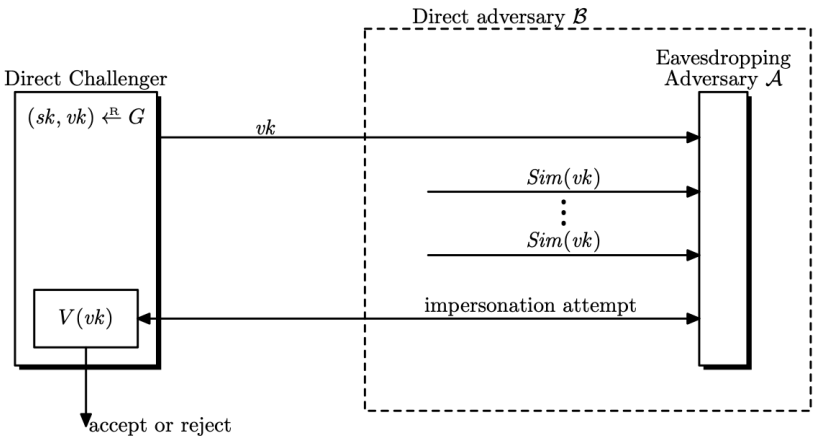


图 19.2: 定理 19.3 的证明中的对手 $\mathcal{B}$

样的，只是现在 $\mathcal{B}$ 自己生成了对话列表，而不是从挑战者那里获取记录。这样，$\mathcal{B}$ 就发动了一个直接攻击，但是它与发动仿冒攻击的 $\mathcal{A}$ 具有相同的优势。

下面，我们将上面的想法扩展到更一般的情况，引入**诚实验证者零知识**的概念。

定义 19.1. 令 $\mathcal{I} = (G, P, V)$ 是一个身份识别协议。如果存在一个有效的概率性算法 Sim（称为**模拟器** ***simulator***）使得对于 G 的任意可能输出 (vk, sk)，Sim 在输入 vk 上的输出分布与 P（在输入 sk 上）与 V（在输入 vk 上）之间对话记录的分布相同，我们就称 $\mathcal{I}$ 是**诚实验证者零知识** ***(honest verifier zero knowledge, HVZK)*** 的。

下面，我们对其中的一些术语进行说明。所谓“零知识”的意思是，对手从 P 那里什么也学不到，因为对手可以在不知道 sk 的情况下自己使用算法 Sim 模拟对话。而“诚实验证者”表达了这样一个事实，即这种模拟只对 P 和诚实的验证者 V 之间的对话有效，而对于非诚实的验证者，比如主动攻击中可能出现的验证者，都是无效的。零知识的概念，包括诚实验证者零知识以及许多其他的变体，还会出现在身份识别协议之外的许多其他类型的协议中。

定理 19.3. 如果一个身份识别协议 $\mathcal{I} = (G, P, V)$ 对于直接攻击是安全的，且该协议是 $HVZK$ 的，那么该协议对于窃听攻击也是安全的。

> 特别地，如果 $\mathcal{I}$ 是带有模拟器 Sim 的 $HVZK$，那么对于每一个通过攻击游戏 *18.2* 中的窃听攻击来攻击 $\mathcal{I}$ 的冒充对手 $\mathcal{A}$，如果 $\mathcal{A}$ 最多能获得 Q 次交互记录，必然存在一个通过攻击游戏 *18.1* 中的直接攻击来攻击 $\mathcal{I}$ 的对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，并且 $\mathcal{B}$ 最多运行 Q 次 Sim，使得：
>
> $$\mathrm{ID2adv}[\mathcal{A}, \mathcal{I}] = \mathrm{ID1adv}[\mathcal{B}, \mathcal{I}]$$

证明. $\mathcal{B}$ 的工作方式与 $\mathcal{A}$ 相同，只是它不是从挑战者那里获得交互记录，而是用 Sim 自己生成交互记录。对手 $\mathcal{B}$ 的工作方式如图 19.2 所示。 □

下面让我们回到 Schnorr 身份识别协议。

定理 19.4. *Schnorr* 身份识别协议是 $HVZK$ 的。

证明. 证明的思路是在生成模拟对话 $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ 时，我们不需要像 P 和 V 之间的真实对话那样，按照给定的顺序生成对话中的消息。事实上，模拟器 Sim 是以**倒序**生成信息的。在输入 $vk = u$ 时，模拟器 Sim 计算：

$$\alpha_{\mathrm{z}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad c \xleftarrow{\mathrm{R}} \mathcal{C}, \quad u_{\mathrm{t}} \leftarrow \frac{g^{\alpha_{\mathrm{z}}}}{u^c}$$

并输出对话 $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$。

现在我们论证 Sim 在输入 $vk = u$ 上的输出具有正确的分布。一个关键性的观察是，在真实交互中，c 和 α_{z} 是相互独立的，c 在 $\mathcal{C}$ 上均匀分布，而 α_z 在 $\mathbb{Z}_q$ 上均匀分布。此外，给定 c 和 α_{z}，u_{t} 的值可由等式 $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$ 唯一确定。而这显然与模拟器的输出分布相同。 □

作为一个推论，我们立即可以得到：

定理 19.5. *如果 Schnorr 身份识别协议对直接攻击是安全的，那么它对窃听攻击也是安全的。*

> *特别地，对于每个通过攻击游戏 18.2 中的窃听攻击来攻击 $\mathcal{I}_{\mathrm{sch}}$ 的冒充对手 $\mathcal{A}$，都有一个通过攻击游戏 18.1 中的直接攻击来攻击 $\mathcal{I}_{\mathrm{sch}}$ 的对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，满足：*
>
> $$\mathrm{ID2adv}[\mathcal{A}, \mathcal{I}_{\mathrm{sch}}] = \mathrm{ID1adv}[\mathcal{B}, \mathcal{I}_{\mathrm{sch}}]$$

乍看之下，我们关于 Schnorr 协议的结论似乎是反直觉的，甚至可能是矛盾的。在只知道 vk 的情况下，怎么可能既很难进行冒充攻击，又容易产生对话呢？答案是，在进行冒充攻击时，验证者 V 积极参与对话，而消息的时间和顺序至关重要：扮演证明者的对手必须在看到 V 生成的挑战 c **之前**生成第一条消息 u_{t}。然而，模拟器可以自由地以任何方便的顺序生成信息：在定理 19.4 的证明中，模拟器先生成了 c 和 α_{z}，然后才计算 u_{t}。这些结果表明，如果挑战空间很小，Schnorr 协议将是完全**不安全**的：在冒充尝试中，对手可以使用模拟器准备一个接受对话 $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$，然后将 u_{t} 发送给 V，然后希望 V 选择的挑战等于其准备的挑战 c。如果真的是这样，对手就可以用 α_{z} 应答从而使 V 接受对话。因此，以 $1/|\mathcal{C}|$ 的优势破解 Schnorr 识别协议是很容易的。所以为了确保安全性，挑战空间的大小 $|\mathcal{C}|$ 必须是超多项式的。

Schnorr 识别协议对攻击游戏 18.3 中的**主动**攻击是否安全是一个开放的问题：目前还没有已知的有效主动攻击，但也没有证据可以排除离散对数假设下的这种攻击。在本章的稍后部分，我们将介绍一个对 Schnorr 身份识别的细小变化，它被证明在 DL 假设下对主动攻击是安全的。

19.2 从身份识别协议到签名

在本节中，我们将展示如何将 Schnorr 协议转换为一个签名方案。在离散对数假设下，该签名方案在随机预言机模型下是安全的。在本章的稍后部分，我们将看到这个构造实际上是一个更一般的结构的一个具体实例。

我们从 Schnorr 识别协议 $\mathcal{I}_{\mathrm{sch}}$ 开始，它定义在阶为素数 q 的循环群 $\mathbb{G}$ 上，该群有生成元 $g \in \mathbb{G}$。此外，其挑战空间 $\mathcal{C} \subseteq \mathbb{Z}_q$。除此之外，我们还需要一个哈希函数 $H : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$，该函数在安全证明中被建模为一个随机预言机。这里的 $\mathcal{M}$ 将会是签名方案的消息空间。

签名方案的构造思想是，对于消息 $m \in \mathcal{M}$ 的签名是一个元组 $(u_{\mathrm{t}}, \alpha_{\mathrm{z}})$，其中 $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ 是在 Schnorr 识别协议 $\mathcal{I}_{\mathrm{sch}}$ 中对验证密钥 u 的接受对话，挑战则来自 $c \leftarrow H(m, u_{\mathrm{t}})$。直观地讲，哈希函数 H 在这里扮演 Schnorr 身份识别协议中验证者的角色。

具体地，Schnorr 签名方案为 $\mathcal{S}_{\text{sch}} = (G, S, V)$，其中：

- 密钥生成算法 G 按如下方式运行：
$$\alpha \xleftarrow{R} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha}$$
其中公钥为 $pk := u$，私钥为 $sk := \alpha$。

- 为了使用私钥 $sk = \alpha$ 对消息 $m \in \mathcal{M}$ 进行签名，签名算法 S 按如下方式运行：
$$\alpha_{\text{t}} \xleftarrow{R} \mathbb{Z}_q, \quad u_{\text{t}} \leftarrow g^{\alpha_{\text{t}}}, \quad c \leftarrow H(m, u_{\text{t}}), \quad \alpha_{\text{z}} \leftarrow \alpha_{\text{t}} + \alpha c$$
输出签名 $\sigma := (u_{\text{t}}, \alpha_{\text{z}})$。

- 为了使用公钥 $pk = u$ 验证消息 $m \in \mathcal{M}$ 的签名 $\sigma = (u_{\text{t}}, \alpha_{\text{z}})$，签名验证算法 V 计算 $c \leftarrow H(m, u_{\text{t}})$，当 $g^{\alpha_{\text{z}}} = u_{\text{t}} \cdot u^{c}$ 成立时输出 accept，否则输出 reject。

尽管我们把签名算法描述为一种随机化算法，但这并不是必须的。练习 13.6 展示了如何对签名算法进行去随机化。这种去随机化在实践中很重要，因为它可以避免不良随机性攻击，就像练习 19.1 所展示的那样。

我们下面将要说明，如果把 H 建模为一个随机预言机，那么如果 Schnorr 识别协议对窃听攻击是安全的，那么 Schnorr 签名方案也一定是安全的。而我们已经在定理 19.5 中证明了前者。然而，首先考虑窃听攻击游戏的一个稍微增强的版本是有好处的。

19.2.1 一个有用的抽象：重复冒充攻击

我们考虑一种增强版本的针对识别协议的冒充攻击，在这种攻击中，我们允许对手反复进行冒充尝试，这些冒充尝试针对的是验证者的不同实例，它们同时运行，并使用相同的验证公钥。我们可以为直接攻击、窃听攻击和主动攻击定义这个概念，但我们目前先考虑窃听攻击，因为这就是我们的应用所需要的。另外，我们只考虑无状态和有验证公钥的识别协议。

下面是攻击游戏的更多细节。

攻击游戏 19.1 (r 次重复冒充窃听攻击). 给定识别协议 $\mathcal{I} = (G, P, V)$，正整数 r 和对手 $\mathcal{A}$，攻击游戏按以下流程运行。其密钥生成阶段和窃听阶段与攻击游戏 18.2 完全相同。

唯一不同的是冒充阶段。现在，我们允许对手 $\mathcal{A}$ 与最多 r 个验证者同时互动。挑战者扮演这些验证者的角色，他们都使用在密钥生成阶段产生的相同的验证公钥。如果对手使这些验证者中的任何一个输出 accept，我们就称他赢得了该游戏。

我们将 $\mathcal{A}$ 对于 $\mathcal{I}$ 和 r 的优势记为 $\text{rID2adv}[\mathcal{A}, \mathcal{I}, r]$，其值等于 $\mathcal{A}$ 赢得该游戏的概率。

下面的引理将表明，r 次重复冒充窃听攻击等价于普通的窃听攻击。也就是说，赢得攻击游戏 19.1 并不会比赢得攻击游戏 18.2 更容易。

引理 19.6. *令 $\mathcal{I}$ 是一个身份识别协议。对于每个 r 次重复冒充窃听攻击对手 $\mathcal{A}$，都存在一个标准窃听对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，满足：*

$$\text{rID2adv}[\mathcal{A}, \mathcal{I}, r] \leq r \cdot \text{ID2adv}[\mathcal{B}, \mathcal{I}] \tag{19.4}$$

证明简述. 这是一个简单的"猜测论证"。对手 $\mathcal{B}$ 随机选择 $\omega \in \{1,\ldots,r\}$，然后扮演 $\mathcal{A}$ 的挑战者。它首先从自己的挑战者那里获得验证公钥和几条对话记录，并将这些传递给 $\mathcal{A}$。在冒充阶段，对于验证者的第 j 个实例，如果 $j \neq \omega$，对手 $\mathcal{B}$ 自己扮演验证者；否则，如果 $j = \omega$，对手 $\mathcal{B}$ 就作为 $\mathcal{A}$ 和攻击游戏 18.2 中他自己的挑战者之间的简单管道。显然，$\mathcal{A}$ 在与 $\mathcal{B}$ 游戏时让其中一个验证者输出 accept 的概率与攻击游戏 19.1 中相同。此外，如果 $\mathcal{B}$ 猜到了其中一个输出 accept 的验证者的索引，他就赢得了攻击游戏，这发生的概率至少是 $1/r$。 □

19.2.2 Schnorr 签名的安全性分析

我们下面将表明，只要 Schnorr 身份识别协议对窃听攻击是安全的，Schnorr 签名方案在随机预言机模型下就是安全的。

定理 19.7. *如果 H 是一个随机预言机，且 Schnorr 身份识别协议对窃听攻击是安全的，那么 Schnorr 签名方案也是安全的。*

> *特别地，令 $\mathcal{A}$ 是如攻击游戏 19.1 的随机预言机模型那样攻击 $\mathcal{S}_{\text{sch}}$ 的一个对手，假设 $\mathcal{A}$ 最多发起 Q_{s} 次签名查询和 Q_{ro} 次随机预言机查询。那么必然存在一个 $(Q_{\text{ro}}+1)$ 次重复冒充攻击对手 $\mathcal{B}$ 能够通过攻击游戏 19.1 中的窃听攻击来攻击 $\mathcal{I}_{\text{sch}}$，且 $\mathcal{B}$ 是一个 $\mathcal{A}$ 的基本包装器，并有：*

$$\text{SIG}^{\text{ro}}\mathsf{adv}[\mathcal{A},\mathcal{S}_{\text{sch}}] \leq \frac{Q_{\text{s}}(Q_{\text{s}}+Q_{\text{ro}}+1)}{q} + \text{rID2}\mathsf{adv}[\mathcal{B},\mathcal{I}_{\text{sch}},Q_{\text{ro}}+1] \tag{19.5}$$

证明思路. 我们的目标是将伪造签名的对手 $\mathcal{A}$ 转化为在 r 次重复冒充窃听攻击中破坏 Schnorr 身份识别协议安全性的对手 $\mathcal{B}$，其中 $r := Q_{\text{ro}}+1$。

第一个想法是，我们必须以某种方式在不使用私钥的情况下响应 $\mathcal{A}$ 的签名查询。想要实现这一点，可以通过利用窃听攻击中获取的对话记录来建立所需的签名，并 "修正" 随机预言机 H，使其与签名一致。只有当随机预言机在已经被查询过的位置重新被查询时，这种修复才会失败。但由于随机预言机的输入包括一个随机群元素，因此发生的概率极低。这也就是公式 19.5 中出现 $Q_s(Q_s+Q_{\text{ro}}+1)/q$ 修正项的原因。

一旦摆脱了签名查询，我们认为，如果对手成功地伪造了签名，他就可以有效地被用于对 $\mathcal{I}_{\text{sch}}$ 的 r 次重复冒充窃听攻击。我们再次利用了 H 被建模为一个随机预言机的事实。由于签名伪造必须被包含在没有被作为签名查询而提交的信息中，因此相应的随机预言机查询必须与所有签名查询的位置相异。由于伪造签名与签名查询不能同时作为同一条消息提交，因此相应的随机预言机查询必须与所有签名查询的位置相异。所以，该位置上随机预言机的输出值在身份识别协议的运行中基本上可以视作一个随机挑战。我们事先并不知道哪一个随机预言机查询对应于一个伪造签名，这就是我们必须使用 r 次重复冒充窃听攻击游戏的原因。 □

证明. 为了简化分析，我们将假设当 $\mathcal{A}$ 输出一个伪造签名 (m,σ)，$\sigma = (u_{\text{t}},\alpha_{\text{z}})$ 时，$\mathcal{A}$ 一定已经在 (m,u_{t}) 这一点上显式地查询了随机预言机。如有必要，我们会修改 $\mathcal{A}$ 以确保这种情况，以使得修改后的 $\mathcal{A}$ 所进行的随机预言机查询次数最多为 $Q_{\text{ro}}+1$。

我们下面将定义两个攻击游戏。游戏 0 基本上就是原始的签名攻击游戏，其中 H 被建模为一个随机预言机。游戏 1 是一个少量修改后的版本。对于 $j=0,1$，我们记 W_j 是 $\mathcal{A}$ 在游戏 j 中获胜的事件。

初始化：

令 $\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q$，计算 $u \leftarrow g^{\alpha}$

初始化两个空的关联数组 $Map : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$ 和 $Explicit : \mathcal{M} \times \mathbb{G} \to \mathbb{Z}$

将公钥 u 发送给 $\mathcal{A}$；

当收到第 i 个签名查询 $m_i \in \mathcal{M}$ 时：

令 $\alpha_{\mathrm{t}i} \xleftarrow{\mathrm{R}} \mathbb{Z}_q$，$c_i \xleftarrow{\mathrm{R}} \mathcal{C}$，计算 $u_{\mathrm{t}i} \leftarrow g^{\alpha_{\mathrm{t}i}}$

(1) 如果 $(m_i, u_{\mathrm{t}i}) \in \mathrm{Domain}(Map)$，则令 $c_i \leftarrow Map[m_i, u_{\mathrm{t}i}]$

令 $Map[m_i, u_{\mathrm{t}i}] \leftarrow c_i$

计算 $\alpha_{\mathrm{z}i} \leftarrow \alpha_{\mathrm{t}i} + \alpha c_i$

将 $(u_{\mathrm{t}i}, \alpha_{\mathrm{z}i})$ 发送给 $\mathcal{A}$；

当收到第 j 个随机预言机查询 $(\widehat{m}_j, \widehat{u}_j) \in \mathcal{M} \times \mathbb{G}$ 时：

如果 $(\widehat{m}_j, \widehat{u}_j) \notin \mathrm{Domain}(Map)$：

令 $Map[\widehat{m}_j, \widehat{u}_j] \xleftarrow{\mathrm{R}} \mathcal{C}$，$Explicit[\widehat{m}_j, \widehat{u}_j] \leftarrow j$

将 $Map[\widehat{m}_j, \widehat{u}_j]$ 发送给 $\mathcal{A}$；

当收到一个伪造尝试 $(m, u_{\mathrm{t}}, \alpha_{\mathrm{z}})$ 时：

// 根据假设，有 $(m, u_{\mathrm{t}}) \in \mathrm{Domain}(Explicit)$

如果 $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^{c}$，其中 $c = Map[m, u_{\mathrm{t}}]$：

则输出 “win”

否则输出 “lose”

图 19.3: 游戏 0 的挑战者

游戏 0。挑战者的工作方式与攻击游戏 13.1 的随机预言机版本相同。与往常一样，我们用一个关联数组 $Map : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$ 来实现随机预言机。我们还维护一个关联数组 $Explicit : \mathcal{M} \times \mathbb{G} \to \mathbb{Z}$ 以记录那些随机预言机第一次被对手显式查询的位置，而不包含被签名算法隐式地查询的位置。挑战者的工作逻辑如图 19.3。

为了处理一个签名查询 m_i，挑战者像往常一样运行签名算法：他首先生成一个随机的 $\alpha_{\mathrm{t}i} \in \mathbb{Z}_q$ 并计算 $u_{\mathrm{t}i} \leftarrow g^{\alpha_{\mathrm{t}i}}$。然后它为 $Map[m_i, u_{\mathrm{t}i}]$ 的值生成一个随机 “默认” 值 $c_i \in \mathcal{C}$，如果通过行 (1) 的测试发现 $Map[m_i, u_{\mathrm{t}i}]$ 已经被定义，那么就使用那个先前定义的值，而不是默认值。

为了处理一个随机预言机查询 $(\widehat{m}_j, \widehat{u}_j)$，如果 $Map[\widehat{m}_j, \widehat{u}_j]$ 的值尚未被之前的签名查询或随机预言机查询所定义，就在此时定义它，并设定 $Explicit[\widehat{m}_j, \widehat{u}_j] \leftarrow j$。

假设对手提交一个伪造尝试 $(m, u_t, \alpha_{\mathrm{z}})$，且 m 与作为签名查询提交的所有 m_i 都不同，根据我们的简化假设，对手必须在之前曾将 (m, u_{t}) 作为随机预言机查询提交，因此此时 (m, u_{t}) 一定在 $\mathrm{Domain}(Explicit)$ 中。由此可见，如果 $(u_{\mathrm{t}}, \alpha_{\mathrm{z}})$ 是一个有效签名，那么挑战者将输出 “win”，因此:

$$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{sch}}] \leq \Pr[W_0]$$

游戏 1。游戏 1 与游戏 0 基本相同，只是删去了图 19.3 中行 (1) 的测试。直接应用差分引理，我们就可以得到:

$$|\Pr[W_1] - \Pr[W_0]| \leq \frac{Q_{\mathrm{s}}(Q_{\mathrm{s}} + Q_{\mathrm{ro}} + 1)}{q}$$

初始化：

从挑战者处收到验证公钥 u

从挑战者处获取窃听对话 $(u_{\mathrm{t}i}, c_i, \alpha_{\mathrm{z}i})$, $i = 1, \ldots, Q_s$

初始化两个空的关联数组 $Map : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$ 和 $Explicit : \mathcal{M} \times \mathbb{G} \to \mathbb{Z}$

将 u 发送给 $\mathcal{A}$；

当从 $\mathcal{A}$ 处收到第 i 个签名查询 $m_i \in \mathcal{M}$ 时：

令 $Map[m_i, u_{\mathrm{t}i}] \leftarrow c_i$

将 $(u_{\mathrm{t}i}, \alpha_{\mathrm{z}i})$ 发送给对手 $\mathcal{A}$；

当收到第 j 个随机预言机查询 $(\widehat{m}_j, \widehat{u}_j) \in \mathcal{M} \times \mathbb{G}$ 时：

如果 $(\widehat{m}_j, \widehat{u}_j) \notin \mathrm{Domain}(Map)$：

发起对验证者 V_j 的冒充尝试：

将 $\widehat{u}_j$ 发送给 V_j，后者将回复一个挑战 $\widehat{c}_j$

令 $Map[\widehat{m}_j, \widehat{u}_j] \leftarrow \widehat{c}_j$，$Explicit[\widehat{m}_j, \widehat{u}_j] \leftarrow j$

将 $Map[\widehat{m}_j, \widehat{u}_j]$ 发送给 $\mathcal{A}$；

当收到一个伪造尝试 $(m, u_{\mathrm{t}}, \alpha_{\mathrm{z}})$ 时：

// 根据假设，有 $(m, u_{\mathrm{t}}) \in \mathrm{Domain}(Explicit)$

向 V_j 发送最终消息 α_z，其中 $j = Explicit[m, u_{\mathrm{t}}]$

图 19.4: 对手 $\mathcal{B}$

事实上，对于第 i 个签名查询，$u_{\mathrm{t}i}$ 在 $\mathbb{G}$ 上均匀分布，联合约束意味着随机预言机已经在 $(m, u_{\mathrm{t}i})$ 位置上被查询过的概率最多是 $Q_s(Q_s + Q_{\mathrm{ro}} + 1)/q$，查询既可能是对手发出的，也可能是之前的签名查询间接导致的。联合约束还意味着对于任何签名来说，发生该情况的概率的总上界为 $Q_s(Q_s + Q_{\mathrm{ro}} + 1)/q$。

做出这一改变的意义在于，现在在游戏 1 中，一个新的随机挑战被用来处理每个签名查询，就像 Schnorr 身份识别协议中的诚实验证者一样。

基于此，很容易构建一个对手 $\mathcal{B}$，它对挑战者进行 $r = Q_{\mathrm{ro}} + 1$ 的 r 次重复冒充窃听攻击游戏，并且自己在游戏 1 中扮演 $\mathcal{A}$ 的挑战者的角色，因而：

$$\Pr[W_1] = \mathrm{rID2adv}[\mathcal{B}, \mathcal{I}_{\mathrm{sch}}, r]$$

$\mathcal{B}$ 的工作逻辑如图 19.4 所示。对于 $j = 1, \ldots, r$，我们用 V_j 表示在 r 次重复冒充窃听攻击游戏中的第 j 个验证者。这样定理 19.7 得证。 □

将已有的结论结合起来。如果我们把定理 19.7、引理 19.6、定理 19.5 和定理 19.1 的结论结合起来，我们就可以得到下面的从对 Schnorr 签名方案的攻击到计算离散对数的问题规约。

令 $\mathcal{A}$ 是一个攻击 $\mathcal{S}_{\mathrm{sch}}$ 的有效对手，如同攻击游戏 *13.1* 中的随机预言机版本。此外，假设 $\mathcal{A}$ 最多能发出 Q_s 次签名查询和 Q_{ro} 次随机预言机查询。那么必然存在一个有效离散对数对手 $\mathcal{B}$，其运行时间大约是 $\mathcal{A}$ 的两倍，并使得：

$$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{sch}}] \leq \frac{Q_s(Q_s + Q_{\mathrm{ro}} + 1)}{q} + \frac{Q_{\mathrm{ro}} + 1}{N} + (Q_{\mathrm{ro}} + 1)\sqrt{\mathrm{DLadv}[\mathcal{B}, \mathbb{G}]} \tag{19.6}$$

其中 N 是挑战空间的大小。

这种规约并不是非常严格。将标量 $(Q_{\mathrm{ro}}+1)$ 直接与 $\sqrt{\mathrm{DLadv}[\mathcal{B},\mathbb{G}]}$ 相乘是很有问题的。事实上可以得到一个更严格的规约，方法是用 $\sqrt{Q_{\mathrm{ro}}+1}$ 代替 $(Q_{\mathrm{ro}}+1)$，这显然要好得多。诀窍是把引理 19.6 中的“猜测阶段”和定理 19.1 中的“回溯阶段”结合起来，变成一个单一且直接的规约。

引理 19.8. *考虑定义在群 $\mathbb{G}$ 上的 Schnorr 身份识别协议 $\mathcal{I}_{\mathrm{sch}}$，其中 $\mathbb{G}$ 的阶为素数 q，有生成元 $g\in\mathbb{G}$，其挑战空间 $\mathcal{C}$ 的大小为 N。对于每个优势为 $\epsilon:=\mathrm{rID2adv}[\mathcal{A},\mathcal{I}_{\mathrm{sch}},r]$ 的攻击 $\mathcal{I}_{\mathrm{sch}}$ 的有效 r 次重复冒充窃听攻击对手 $\mathcal{A}$，必然存在一个有效离散对数对手 $\mathcal{B}$，其运行时间约为 $\mathcal{A}$ 的两倍，其优势为 $\epsilon':=\mathrm{DLadv}[\mathcal{B},\mathbb{G}]$，使得：*

$$\epsilon'\geq\frac{\epsilon^2}{r}-\frac{\epsilon}{N} \tag{19.7}$$

这意味着：

$$\epsilon\leq\frac{r}{N}+\sqrt{r\epsilon'} \tag{19.8}$$

证明. 让我们首先回顾一下 $\mathcal{A}$ 的攻击游戏是如何进行的。首先，攻击游戏 19.1 中的挑战者向 $\mathcal{A}$ 发送一个用于 Schnorr 身份识别协议的验证公钥 $u\in\mathbb{G}$。然后，挑战者向 $\mathcal{A}$ 发送了几份交互的对话记录。最后，$\mathcal{A}$ 进入冒充阶段，它试图让 r 个验证者中的至少一个输出 accept。更详细的过程是这样的。对于第 j 次运行，其中 j 不大于 r，对手 $\mathcal{A}$ 向挑战者发送 $u_{\mathrm{t}j}$，挑战者用随机挑战 $c_j\in\mathcal{C}$ 来应答。在收到所有挑战后，$\mathcal{A}$ 要么输出 fail，要么输出一个数对 (i,α_{z})，使得 $(u_{\mathrm{t}i},c_i,\alpha_{\mathrm{z}})$ 是验证公钥 u 的接受对话。在后一种情况下，我们称 *$\mathcal{A}$ 在验证者 i 处成功*。注意到 $\mathcal{A}$ 的优势是 $\epsilon=\sum_{j=1}^{r}\epsilon_j$，其中 ϵ_j 是 $\mathcal{A}$ 在验证者 j 处成功的概率。

请注意，我们上面的描述其实稍稍简化了对手 $\mathcal{A}$ 在冒充阶段的行为。然而，由于对手可以自己观察到对话是否被接受，所以这并不是一个真正的限制：任何对手都可以被置于上面所描述的形式中，而完全不改变其优势，也不会显著增加其运行时间。另外，在定理 19.7 的证明中构建的 r 次重复冒充窃听攻击对手本质上已经是这种形式了。

我们下面描述我们的离散对数对手 $\mathcal{B}$，他被给予一个 $u\in\mathbb{G}$，任务是计算出 $\mathsf{Dlog}_g u$。像往常一样，$\mathcal{B}$ 扮演 $\mathcal{A}$ 的挑战者的角色。首先，$\mathcal{B}$ 将 u 作为验证公钥发送给 $\mathcal{A}$。然后，$\mathcal{B}$ 使用定理 19.4 中的模拟器生成对话记录，并将这些记录交给 $\mathcal{A}$。最后，$\mathcal{B}$ 让 $\mathcal{A}$ 完成冒充阶段，并向 $\mathcal{A}$ 提供随机挑战 $c_1,\ldots,c_r$。如果 $\mathcal{A}$ 输出一个数对 (i,α_{z})，使得 $(u_{\mathrm{t}i},c_i,\alpha_{\mathrm{z}})$ 是验证公钥 u 的接受对话，那么 $\mathcal{B}$ 将 $\mathcal{A}$ 回溯到它向第 i 个验证者提交 $u_{\mathrm{t}i}$ 的时刻。然后对手 $\mathcal{B}$ 用一个新的随机挑战 $c'\in\mathcal{C}$ 来替换挑战 c_i，然后让 $\mathcal{A}$ 继续使用与之前相同的挑战 c_j，$j=i+1,\ldots,r$ 完成冒充阶段的剩余部分。如果 $\mathcal{A}$ 输出一个数对 $(i',\alpha'_{\mathrm{z}})$ 使得 $i'=i$ 且 $(u_{\mathrm{t}i},c',\alpha'_{\mathrm{z}})$ 是验证公钥 u 的接受对话，并且 $c'\neq c$，那么 $\mathcal{B}$ 就可以使用这两个接受对话来计算 $\mathsf{Dlog}_g u$，就像定理 19.1 的证明中那样。在这种情况下，我们称 *$\mathcal{B}$ 在验证者 i 处成功*。可以注意到 $\mathcal{B}$ 的优势是 $\epsilon'=\sum_{j=1}^{r}\epsilon'_j$，其中 ϵ'_j 是 $\mathcal{B}$ 在验证者 j 处成功的概率。

剩下的工作就是证明式 19.7，注意式 19.8 可以用与定理 19.1 的证明中几乎相同的方法从式 19.7 推出。

我们声称，对于 $j=1,\ldots,r$，必有：

$$\epsilon'_j\geq\epsilon_j^2-\frac{\epsilon_j}{N} \tag{19.9}$$

事实上，对于给定索引 j，上述不等式必然成立，这是回溯引理 19.2 已经证明了的，其中的 Y 对应于挑战 c_j，Y' 对应于挑战 c'，而 X 对应于 $\mathcal{A}$、$\mathcal{B}$ 和 $\mathcal{B}$ 的挑战者的所有其他随机选择。如果 $\mathcal{A}$ 在验证者 j

处成功，则引理 19.2 中的函数 f 就被定义为 1。因此，如果 $i = j$ 且 $(u_{\mathrm{t}j}, c_j, \alpha_{\mathrm{z}})$ 是一个接受对话，就有 $f(\mathsf{X}, \mathsf{Y}) = 1$。同样地，如果 $i' = j$ 且 $(u_{\mathrm{t}j}, c', \alpha'_{\mathrm{z}})$ 是一个接受对话，就有 $f(\mathsf{X}, \mathsf{Y}') = 1$。

由式 19.9，我们可以得到：

$$\epsilon' = \sum_{j=1}^{r} \epsilon'_j \geq \sum_{j=1}^{r} \epsilon_j^2 - \sum_{j=1}^{r} \frac{\epsilon_j}{N} \geq \frac{\epsilon^2}{r} - \frac{\epsilon}{N}$$

最后一个不等式基于这样一个事实：对于任意函数 $g : \{1, \ldots, r\} \to \mathbb{R}$，必然有：

$$r \sum_{j=1}^{r} g(j)^2 \geq (\sum_{j=1}^{r} g(j))^2$$

这个结论也是显然的。比如在概率论中，对于任意随机变量 X，必有 $E[\mathsf{X}^2] \geq E[\mathsf{X}]^2$。特别地，如果令 $\mathsf{X} := g(\mathsf{R})$，其中 R 是在 $\{1, \ldots, r\}$ 上的均匀分布，就转化成了我们上面的场景。 □

基于这个结论，我们可以把式 19.6 中的约束替换成：

$$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{sch}}] \leq \frac{Q_{\mathrm{s}}(Q_{\mathrm{s}} + Q_{\mathrm{ro}} + 1)}{q} + \frac{Q_{\mathrm{ro}} + 1}{N} + \sqrt{(Q_{\mathrm{ro}} + 1) \cdot \mathrm{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}]} \tag{19.10}$$

19.2.3 一个具体实现与一种优化方法

我们可以将 $\mathbb{G}$ 看作是定义在有限域 $\mathbb{F}_p$ 上的椭圆曲线群 P256，其中 p 是一个 256 位素数，具体信息可以参见 15.3 节。使用 128 位的挑战就足够了。在这种情况下，Schnorr 签名 $(u_{\mathrm{t}}, \alpha_{\mathrm{z}})$ 中的每个组成部分都是 256 位的，一个 Schnorr 签名总共约为 512 位。

由于挑战的长度比群元素的编码长度要短得多，下面介绍的 Schnorr 签名优化方案可以获得更短的签名。回顾一下，我们之前将 m 上的签名定义为 $(u_{\mathrm{t}}, \alpha_{\mathrm{z}})$，满足：

$$g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$$

其中 $c := H(m, u_{\mathrm{t}})$。现在，与之前不同，我们可以把签名定义为数对 (c, α_{z})，它满足：

$$c = H(m, u_{\mathrm{t}})$$

其中 $u_{\mathrm{t}} = g^{\alpha_{\mathrm{z}}}/u^c$。变换 $(u_{\mathrm{t}}, \alpha_{\mathrm{z}}) \mapsto (H(m, u_{\mathrm{t}}), \alpha_{\mathrm{z}})$ 可以将一个常规的对 m 的 Schnorr 签名映射到一个优化 Schnorr 签名上，而变换 $(c, \alpha_{\mathrm{z}}) \mapsto (g^{\alpha_{\mathrm{z}}}/u^c, \alpha_{\mathrm{z}})$ 可以将一个优化 Schnorr 签名映射到一个常规 Schnorr 签名上。由此可见，伪造一个优化 Schnorr 签名等价于伪造一个常规 Schnorr 签名。进一步优化的话，我们还可以在公钥中存储 u^{-1} 而不是 u，这将加快签名验证的速度。

通过上述参数的选择，我们可以将签名的长度从 512 位减少到大约 $128 + 256 = 384$ 位，这大约减少 25% 的长度。

19.3 案例研究：ECDSA 签名

1991 年，当需要制定数字签名的联邦标准时，美国国家标准研究所 (NIST) 考虑了一些可行的候选方案。由于 Schnorr 系统受专利保护，NIST 选择了一种基于 $\mathbb{Z}_p^*$ 的素阶子群的临时签名方案，该方案后来被称为**数字签名算法 (Digital Signature Algorithm, DSA)**。该标准后来被更新以支持定义

在有限域上的椭圆曲线群。由此产生的签名方案被称为 **ECDSA**，后来被用在许多现实世界的系统中。我们下面会简要介绍 ECDSA 的工作原理，并讨论几个影响它安全性的问题。

ECDSA 签名方案 (G, S, V) 使用有限域 $\mathbb{F}_p$ 上的椭圆曲线点群 $\mathbb{G}$。令 g 是 $\mathbb{G}$ 的一个生成元，素数 q 是群 $\mathbb{G}$ 的阶。我们还需要一个定义在 $(\mathcal{M}, \mathbb{Z}_q^*)$ 上的哈希函数 H。该方案的工作原理如下：

- $G()$：选择 $\alpha \xleftarrow{R} \mathbb{Z}_q^*$，令 $u \leftarrow g^\alpha \in \mathbb{G}$。输出 $sk := \alpha$，$pk := u$。

- $S(sk, m)$：使用私钥 $sk = \alpha$ 签署消息 $m \in \mathcal{M}$：

  重复：

  选取 $\alpha_t \xleftarrow{R} \mathbb{Z}_q^*$，计算 $u_t \leftarrow g^{\alpha_t}$

  令 $u_t = (x, y) \in \mathbb{G}$，其中 $x, y \in \mathbb{F}_p$

  将 x 当作 $[0, p)$ 区间中的一个整数，令 $r \leftarrow [x]_q \in \mathbb{Z}_q$，即计算 x 模 q

  计算 $s \leftarrow (H(m) + r\alpha)/\alpha_t \in \mathbb{Z}_q$

  直至 $r \neq 0$ 且 $s \neq 0$

  输出 (r, s)

- $V(pk, m, \sigma)$：使用公钥 $pk = u \in \mathbb{G}$ 验证对消息 $m \in \mathcal{M}$ 的签名 $\sigma = (r, s) \in (\mathbb{Z}_q^*)^2$：

  计算 $a \leftarrow H(m)/s \in \mathbb{Z}_q$，$b \leftarrow r/s \in \mathbb{Z}_q$

  计算 $\hat{u}_t \leftarrow g^a u^b \in \mathbb{G}$

  令 $\hat{u}_t = (\hat{x}, \hat{y}) \in \mathbb{G}$，其中 $\hat{x}, \hat{y} \in \mathbb{F}_p$

  将 $\hat{x}$ 当作 $[0, p)$ 区间中的一个整数，令 $\hat{r} \leftarrow [\hat{x}]_q \in \mathbb{Z}_q$，即计算 $\hat{x}$ 模 q

  如果 $r = \hat{r}$ 则输出 accept，否则输出 reject

当使用椭圆曲线 P256 时，p 和 q 都是 256 位素数，因此一个 ECDSA 签名 $\sigma = (r, s)$ 的长度为 512 位。

很容易说明该签名方案是正确的。对于 G 输出的任意密钥对 (pk, sk) 和任意消息 $m \in \mathbb{Z}_q$，如果 $\sigma \xleftarrow{R} S(sk, m)$，那么 $V(pk, m, \sigma)$ 必然输出 accept。原因是 V 计算出的 $\hat{u}_t$ 与 S 计算出的 u_t 必然是相同的。

在特定强假设和理想的群 $\mathbb{G}$ 下 ECDSA 是安全的。

为了保证安全性，签名期间产生的随机值 α_t 必须是新从 $\mathbb{Z}_q^*$ 中均匀挑选出的值。否则该方案会在强意义上变得不安全，因为攻击者可以得到签名私钥 α。针对索尼 PlayStation 3 的攻击就曾经利用了这个漏洞，因为在该设计中，α_t 对于所有发出的签名都是相同的。除此之外，一些针对比特币钱包的攻击也利用了这个问题，因为在一些硬件平台上生成质量较高的随机数是很困难的。一个常用的解决方案是修改签名算法，使得 α 是使用安全性高的 PRF 确定性地生成的，这种修改后的算法被称为**确定性 ECDSA**。Schnorr 签名协议也有同样的问题，因此这种优化也同样适用于它。

ECDSA 不是强安全的. 尽管 Schnorr 签名协议是强安全的（见练习 19.15），但 ECDSA 方案却不是。给定一个对消息 m 的 ECDSA 签名 $\sigma = (r, s)$，任何人都可以对 m 生成其他合法签名。比如说 $\sigma' := (r, -s) \in (\mathbb{Z}_q^*)^2$ 也是一个对消息 m 的合法签名。这个 σ' 之所以合法，是因为椭圆曲线上的点 $u_t \in \mathbb{G}$ 的横坐标与另一点 $1/u_t \in \mathbb{G}$ 的横坐标是相同的。

19.4 Sigma 协议：基本定义

Schnorr 协议是一类被称为 **Sigma 协议**的协议族的其中一个特例。在本节中，我们将介绍与 Sigma 协议相关的基本概念。随后我们将考察 Sigma 协议的一些实例及其应用：

- 我们将考察如何使用 Sigma 协议来构建新的安全识别方案和签名方案；
- 我们将考察如何建立对于**主动**攻击也能保证安全性的身份识别方案，即使这种方案不依赖于随机预言机启发法。回顾一下，我们之前介绍的 Schnorr 协议只能保证对窃听攻击的安全性；
- 在下一章，我们还将看到如何将 Sigma 协议应用于其他与身份识别和签名无关的应用。例如，我们将看到如何加密一条消息 m，然后向持怀疑态度的验证者 "证明" m 满足某些属性，而不向验证者透露关于 m 的任何其他信息。

再回顾一下 Schnorr 识别协议。直观地说，该协议允许证明者 P 说服持怀疑态度的验证者 V 他知道一个满足某些关系的秘密，但不向 V 透露任何关于该秘密的有用信息。对于 Schnorr 的协议，证明者的秘密是私钥 $\alpha \in \mathbb{Z}_q$，它满足关系 $g^\alpha = u$。

下面，我们尝试将其推广到更普遍也更有趣的关系类型。

定义 19.2 (有效关系). *一个有效关系 **(effective relation)** 是一个二元关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$，其中 $\mathcal{X}$、$\mathcal{Y}$ 和 $\mathcal{R}$ 是可有效识别的有限集。$\mathcal{Y}$ 中的元素称为**陈述 (statements)**。如果 $(x, y) \in \mathcal{R}$，我们就称 x 是 y 的**见证 (witness for y)**。*

下面，我们定义 Sigma 协议的语法。

定义 19.3 (Sigma 协议). *令 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 是一个有效关系。一个 $\mathcal{R}$ 上的 **Sigma 协议**是一个对算法 (P, V)。*

- *P 是一个交互式协议算法，称为**证明者 (prover)**，它接受一个见证-陈述对 $(x, y) \in \mathcal{R}$ 作为输入。*
- *V 是一个交互式协议算法，称为**验证者 (verifier)**，它接受一个陈述 $y \in \mathcal{Y}$ 作为输入，输出 accept 或 reject。*
- *P 和 V 之间的交互按如下流程进行：*
  - *为了启动协议，P 计算出一条消息 t，称为**承诺 (commitment)**，并将 t 发送给 V；*
  - *收到 P 的承诺 t 后，V 从一个有限的**挑战空间 (challenge space)** $\mathcal{C}$ 中随机选取一个**挑战 (challenge)** c，并将 c 发送给 P；*
  - *收到 V 的挑战 c 后，P 计算出一个**应答 (response)** z，并将 z 发送给 V；*
  - *收到 P 的应答 z 后，V 输出 accept 或 reject。V 的输出必须严格作为陈述 y 和**对话 (conversation)** (t, c, z) 的函数。特别地，除了挑战 c 的随机选择之外，V 不做任何其他随机选择，即所有其他计算都是完全确定性的。*

我们要求，对于所有的 $(x, y) \in \mathcal{R}$，当 $P(x, y)$ 与 $V(y)$ 完成交互时，V 总是会输出 accept。

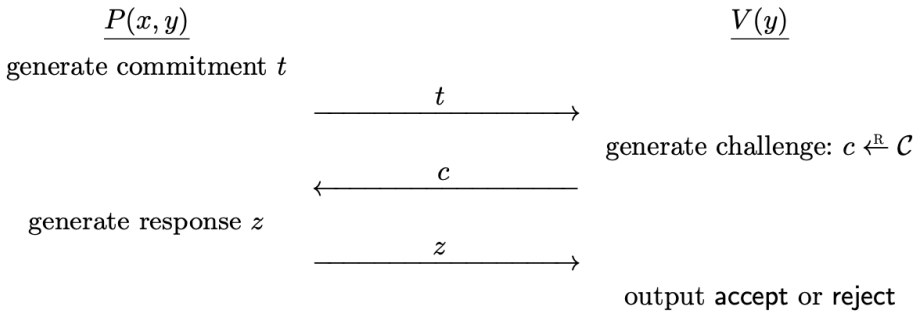


图 19.5: Sigma 协议的执行

图 19.5 展示了 Sigma 协议的执行流程。Sigma 协议的得名是因为这种协议中信息流的形状依稀让人想起希腊字母 Σ。

如定义所述，我们要求验证者的输出是陈述 y 和对话 (t, c, z) 的函数。当 V 输出 accept 时，我们称对话 (t, c, z) 是一个 y **的接受对话 (accepting conversation for** y**)**。显然验证者与诚实的证明者之间只会产生接受对话；但当证明者不诚实或不遵循协议时，也有可能产生非接受对话。

在大多数 Sigma 协议的应用中，我们都会要求挑战空间的大小是超多项式的，为了简洁地表达该要求，我们有时也说协议需要**大的挑战空间**。

例 19.1. 显然 Schnorr 识别协议 (G, P, V) 中的算法 (P, V) 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议的一个特例，其中：

$$\mathcal{X} = \mathbb{Z}_q, \quad \mathcal{Y} = \mathbb{G}, \quad \mathcal{R} = \{(\alpha, u) \in \mathbb{Z}_q \times \mathbb{G} : g^\alpha = u\}$$

挑战空间 $\mathcal{C}$ 是 $\mathbb{Z}_q$ 的一个子集。我们称 (P, V) 为 **Schnorr 的 Sigma 协议 (Schnorr's Sigma protocol)**。

读者应该注意到，与识别协议不同的是，Sigma 协议本身并没有指定生成 $\mathcal{R}$ 中元素的算法。

还需要注意的是，关系 $\mathcal{R}$ 以群 $\mathbb{G}$ 作为参数，具体包括群 $\mathbb{G}$ 的阶 q 和生成元 $g \in \mathbb{G}$。一般来说，我们允许用这种系统参数的方式来定义有效关系，这些参数在系统设置时产生，并公开给所有协议参与方。

Schnorr 的 Sigma 协议的陈述是一个群元素 $u \in \mathbb{G}$，而 u 的见证是使得 $g^\alpha = u$ 成立的 $\alpha \in \mathbb{Z}_q$。因此每个陈述都对应着唯一的一个见证。一个 u 的接受对话是一个形如 $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ 的三元组，其中 $u_{\mathrm{t}} \in \mathbb{G}$，$c \in \mathcal{C}$，$\alpha_{\mathrm{z}} \in \mathbb{Z}_q$，满足：

$$g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$$

读者可能已经注意到，在 Schnorr 身份识别协议中，验证者 P 只需要接受见证 α 作为输入，而不是像 Sigma 协议中所要求的见证-陈述对 (α, u)。事实上，还有很多其他的 Sigma 协议的实例也同样不要求证明者在计算中显式地使用陈述。

19.4.1 知识健全性

接下来我们为 Sigma 协议定义一个关键的安全属性，称为**知识健全性 (knowledge soundness)**。

定义 19.4 (知识健全性). *令 (P, V) 是一个关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议，如果存在一个有效确定性算法 Ext（称为*见证提取器*）具备下述属性：给定一个陈述 $y \in \mathcal{Y}$ 和 y 的两个接受对话 (t, c, z) 和*

(t, c', z') 作为输入，其中 $c \neq c'$，算法 Ext 总是能输出 $x \in \mathcal{X}$ 使得 $(x, y) \in \mathcal{R}$，即 x 是 y 的一个见证；我们就称 *Sigma* 协议 (P, V) 能够提供**知识健全性**。

例 19.2. 回顾例 19.1，我们很容易验证 Schnorr 的 Sigma 协议具备知识健全性。见证提取器 Ext 以 $u \in \mathbb{G}$ 为输入，并接受 u 的两个接受对话 $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ 和 $(u_{\mathrm{t}}, c', \alpha'_{\mathrm{z}})$，其中 $c \neq c'$。正如定理 19.1 的证明中所进行的操作，我们也可以从这两个接受对话中计算出相应的见证 $\alpha = \mathsf{Dlog}_g u$，其值为 $\Delta\alpha/\Delta c \in \mathbb{Z}_q$，其中 $\Delta\alpha := \alpha_{\mathrm{z}} - \alpha'_{\mathrm{z}}$，$\Delta c := c - c'$。

假设 (P, V) 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议。此外，假设 (P, V) 提供知识健全性并且具有大的挑战空间。那么在某种意义上，(P, V) 充当了一个**知识证明 (proof of knowledge)**。考虑任意一个证明者 P^*（甚至可能是一个潜在的“作弊”的验证者），使 V 以不可忽略的概率接受一个陈述 y。那么 P^* 一定“知道” y 的一个见证，方法如下：如定理 19.1 的证明中那样，我们可以回溯 P^* 来得到 y 的两个接受对话 (t, c, z) 和 (t, c', z')，其中 $c \neq c'$，然后使用见证提取器计算出见证 x。

更一般地说，当一个密码学家说 P^* 一定“知道”一个陈述 y 的见证时，她的意思是可以通过回溯从 P^* 中提取出见证 x。虽然我们不会正式定义“知识证明”的概念，但我们将在一些应用中应用知识健全性。

19.4.2 特殊诚实验证者零知识

我们之前在 19.1.1小节中介绍了用于身份认证协议的诚实验证者零知识 (HVZK) 的概念。我们现在可以很容易地将这个概念应用到 Sigma 协议的场景中。

令 (P, V) 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议。直观上我们想表达的是，对于 $(x, y) \in \mathcal{R}$，$P(x, y)$ 和 $V(y)$ 之间的对话不应该透露任何关于见证 x 的信息。下面我们将会严格定义这个直观上的概念，即我们可以在不了解见证 x 的前提下有效模拟 $P(x, y)$ 和 $V(y)$ 之间的对话。然而，我们将增加一些额外的要求，这将简化一些构造和应用。

定义 19.5 (特殊诚实验证者零知识). 令 (P, V) 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 *Sigma* 协议，其挑战空间为 $\mathcal{C}$。如果存在一个有效概率算法 Sim（称为**模拟器**）以 $(y, c) \in \mathcal{Y} \times \mathcal{C}$ 为输入，并满足以下性质：

1. 对于所有输入 $(y, c) \in \mathcal{Y} \times \mathcal{C}$，算法 Sim 总是输出一个数对 (t, z)，使得 (t, c, z) 是 y 的一个接受对话；

2. 对于任意 $(x, y) \in \mathcal{R}$，如果我们计算：

$$c \xleftarrow{\mathrm{R}} \mathcal{C}, \quad (t, z) \xleftarrow{\mathrm{R}} Sim(y, c)$$

则 (t, c, z) 的分布与 $P(x, y)$ 与 $V(y)$ 之间对话记录的分布相同。

则称 *Sigma* 协议 (P, V) 是**特殊诚实验证者零知识** ***(special honest verifier zero knowledge)*** 的，简称为**特殊** ***HVZK (special HVZK)*** 的。

读者有必要认识到这个定义的几个特点。首先，Sim 将挑战 c 作为一个额外的输入。其次，要求即使陈述 y 没有对应的见证，模拟器 Sim 仍然能够产生一个接受对话。这两个特性是“特殊 HVZK”中“特殊”一词的原因。

例 19.3. 回到例 19.2，我们很容易验证 Schnorr 的 Sigma 协议是特殊 HVZK 的。事实上，我们可以将定理 19.4 的证明中的模拟器应用到这个场景中。对于输入 $u \in \mathbb{G}$ 和 $c \in \mathcal{C}$，模拟器计算：

$$\alpha_z \xleftarrow{R} \mathbb{Z}_q, \quad u_t \leftarrow g^{\alpha_z}/u^c$$

并输出 (u_t, α_z)。读者可以自行验证该模拟器是否满足定义 19.5 的所有要求。

19.5 Sigma 协议：范例

到目前为止，我们所见过的唯一的 Sigma 协议是 Schnorr 协议，它允许一个证明者说服一个持怀疑态度的验证者，它 "知道" 一个给定群元素的离散对数，却不向验证者透露任何关于离散对数的信息。在本节中，我们将介绍另外几个 Sigma 协议的例子。这些例子不仅有助于充实 Sigma 协议的普遍理论，也有许多实际应用，其中的一些我们将在下面讨论。

19.5.1 用于表示的 Okamoto 协议

令 $\mathbb{G}$ 是一个由 $g \in \mathbb{G}$ 生成的素阶 q 的循环群，$h \in \mathbb{G}$ 是任意群元素。我们现在将 h 看作是一个系统参数。所谓系统参数会在系统初始化时一次性生成，并对所有参与方公开。对于一个 $u \in \mathbb{G}$，如果给定 g 和 h，可以使用一个数对 $(\alpha, \beta) \in \mathbb{Z}_q^2$ 来表示 u，方法是 $g^\alpha h^\beta = u$。

Okamoto 协议允许证明者说服持怀疑态度的验证者，他 "知道" 一个给定的 $u \in \mathbb{G}$ 的表示，但不需要向验证者透露任何有关该表示的具体信息。更确切地说，它是一个对于关系：

$$\mathcal{R} = \left\{ \big((\alpha, \beta), u\big) \in \mathbb{Z}_q^2 \times \mathbb{G} : g^\alpha h^\beta = u \right\} \tag{19.11}$$

的 Sigma 协议。陈述 $u \in \mathbb{G}$ 的一个见证 u 是一个使得 $g^\alpha h^\beta = u$ 成立的数对 $(\alpha, \beta) \in \mathbb{Z}_q^2$，即 u 的一个表示。因此，在这个例子中，每个陈述都对应着多个见证，准确的说是 q 个见证。

通常假定 Okamoto 协议的挑战空间 $\mathcal{C}$ 是 $\mathbb{Z}_q$ 的一个子集。协议 (P, V) 按如下方式运行，其中证明者 P 由 $((\alpha, \beta), u) \in \mathcal{R}$ 初始化，验证者 V 由 $u \in \mathbb{G}$ 初始化：

1. P 计算：
$$\alpha_t \xleftarrow{R} \mathbb{Z}_q, \quad \beta_t \xleftarrow{R} \mathbb{Z}_q, \quad u_t \leftarrow g^{\alpha_t} h^{\beta_t}$$
并将承诺 u_t 发送给 V；
2. V 选取 $c \xleftarrow{R} \mathcal{C}$，然后将挑战 c 发送给 P；
3. P 计算：
$$\alpha_z \leftarrow \alpha_t + \alpha c \in \mathbb{Z}_q, \quad \beta_z \leftarrow \beta_t + \beta c \in \mathbb{Z}_q$$
并将应答 (α_z, β_z) 发送给 V；
4. V 检查 $g^{\alpha_z} h^{\beta_z} \stackrel{?}{=} u_t \cdot u^c$ 是否成立。如果是，V 输出 accept，否则 V 输出 reject。

参见图 19.6。

定理 19.9. *Okamoto 协议是针对式 19.11 中定义的关系 $\mathcal{R}$ 的一个 Sigma 协议。此外，Okamoto 协议提供知识健全性，且是特殊 HVZK 的。*

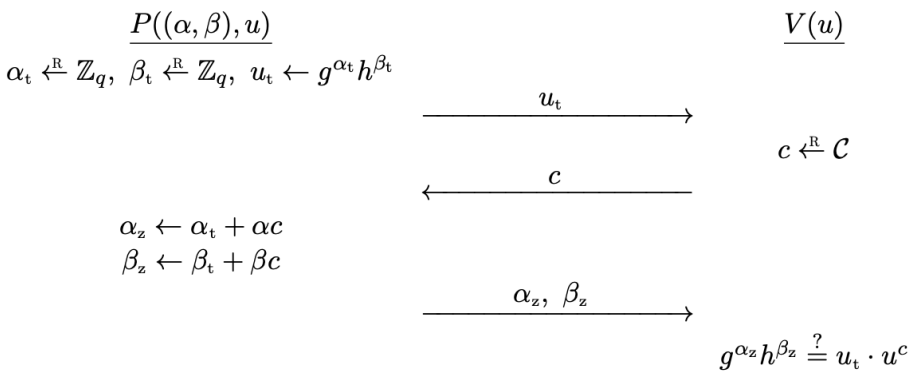


图 19.6: Okamoto 协议

证明. 显然，Okamoto 协议具有 Sigma 协议所要求的语法结构。陈述 $u \in \mathbb{G}$ 的一个接受对话形如：

$$(u_t, c, (\alpha_z, \beta_z)) \ \text{ s.t. } \ g^{\alpha_z} h^{\beta_z} = u_t \cdot u^c$$

正确性. 我们首先必须验证 Okamoto 协议是否满足基本的正确性要求，即诚实证明者和诚实验证者之间的交互总是能够产生一个接受对话。这很容易验证，因为如果：

$$u_t = g^{\alpha_t} h^{\beta_t}, \ \ \alpha_z = \alpha_t + \alpha c, \ \ \beta_z = \beta_t + \beta c$$

则我们有：

$$g^{\alpha_z} h^{\beta_z} = g^{\alpha_t + \alpha c} h^{\beta_t + \beta c} = g^{\alpha_t} h^{\beta_t} \cdot (g^{\alpha} h^{\beta})^c = u_t \cdot u^c$$

知识健全性. 我们下面证明 Okamoto 协议提供知识健全性。假设我们现在有两个陈述 u 的接受对话：

$$(u_t, c, (\alpha_z, \beta_z)), \ \ (u_t, c', (\alpha'_z, \beta'_z))$$

其中 $c \neq c'$。我们需要说明如何从这两个对话中有效提取 u 的表示。这里的计算与 Schnorr 协议中的计算非常类似。注意到：

$$g^{\alpha_z} h^{\beta_z} = u_t \cdot u^c, \ \ g^{\alpha'_z} h^{\beta'_z} = u_t \cdot u^{c'}$$

将第一个等式除以第二个等式，u_t 就被抵消了，于是我们有：

$$g^{\Delta\alpha} h^{\Delta\beta} = u^{\Delta c}, \quad \Delta\alpha := \alpha_z - \alpha'_z, \ \ \Delta\beta := \beta_z - \beta'_z, \ \ \Delta c := c - c'$$

因此，见证提取器可以用：

$$\alpha \leftarrow \Delta\alpha / c, \ \ \beta \leftarrow \Delta\beta / c$$

有效地计算出 u 的一个表示 $(\alpha, \beta) \in \mathbb{Z}_q^2$。需要注意的是，由于 $c \neq c'$，Δc 在 $\mathbb{Z}_q$ 中是可逆的。我们在这里利用了 q 是一个素数这个事实。

特殊 *HVZK*. 最后，我们通过给出一个模拟器来证明 Okamoto 协议是特殊 HVZK 的。这同样与 Schnorr 协议非常相似。对于输入 $u \in \mathbb{G}$ 和 $c \in \mathcal{C}$，模拟器计算：

$$\alpha_z \xleftarrow{R} \mathbb{Z}_q, \ \ \beta_z \xleftarrow{R} \mathbb{Z}_q, \ \ u_t \leftarrow g^{\alpha_z} h^{\beta_z} / u^c$$

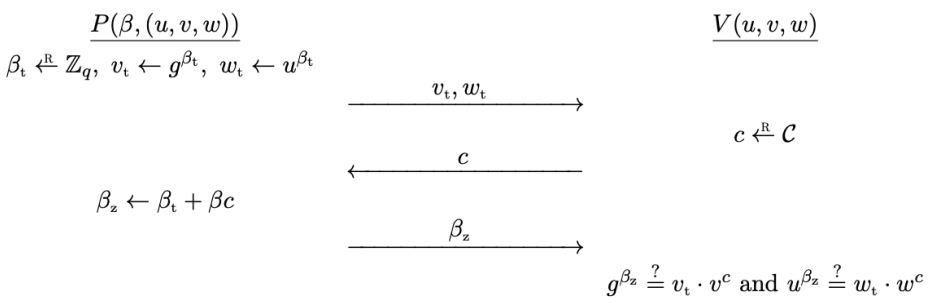


图 19.7: Chaum-Pedersen 协议

并输出 $(u_{\mathrm{t}},(\alpha_{\mathrm{z}},\beta_{\mathrm{z}}))$。可以发现，输出总是会产生一个符合要求的接受对话。

下面我们论证，当 $c \in \mathcal{C}$ 是随机选出的时候，模拟器在输入 u, c 上的输出具有正确的分布。注意到在真实的对话中，c，α_{z} 与 β_{z} 相互独立，其中 c 在 $\mathcal{C}$ 上均匀分布，α_{z} 和 β_{z} 在 $\mathbb{Z}_q$ 上均匀分布。此外，如果给定 c、α_{z} 和 β_{z}，u_{t} 的值由方程：

$$g^{\alpha_{\mathrm{z}}} h^{\beta_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$$

唯一决定。而这显然与模拟器的输出分布是一致的。 □

19.5.2 用于 DH 三元组的 Chaum-Pedersen 协议

Chaum-Pedersen 协议允许证明者在不向验证者透露任何其他信息的情况下，说服一个持怀疑态度的验证者相信一个给定的三元组是 DH 三元组。

令 $\mathbb{G}$ 是一个 q 阶循环群，其中 q 是素数，$g \in \mathbb{G}$ 是生成元。回忆一下 10.5 节，对于 $\alpha, \beta, \gamma \in \mathbb{Z}_q$，如果 $\gamma = \alpha\beta$，我们就称元组 $(g^\alpha, g^\beta, g^\gamma)$ 为一个 DH 三元组 (DH-triple)。等价地，当且仅当存在 $\beta \in \mathbb{Z}_q$ 使得 $v = g^\beta$ 且 $w = u^\beta$ 时，元组 (u, v, w) 才是一个 DH 三元组。

Chaum-Pedersen 协议是关系：

$$\mathcal{R} := \left\{ \big(\beta, (u,v,w)\big) \in \mathbb{Z}_q \times \mathbb{G}^3 : v = g^\beta, w = u^\beta \right\} \tag{19.12}$$

上的 Sigma 协议。陈述 $(u, v, w) \in \mathbb{G}^3$ 的见证是使得 $v = g^\beta$ 且 $w = u^\beta$ 的一个 $\beta \in \mathbb{Z}_q$。因此，当且仅当陈述是一个 DH 三元组时，它才会存在一个见证。与我们之前介绍的其它例子不同，在 Chaum-Pedersen 协议中，并非所有的陈述都必然存在见证。

图 19.7 是 Chaum-Pedersen 协议 (P, V) 的流程图，挑战空间 $\mathcal{C}$ 是 $\mathbb{Z}_q$ 的一个子集。

定理 19.10. *Chaum-Pedersen 协议是式 19.12 中定义的关系 $\mathcal{R}$ 上的一个 Sigma 协议。此外，Chaum-Pedersen 协议提供知识健全性，且是特殊 HVZK 的。*

证明. 显然，Chaum-Pedersen 协议具有 Sigma 协议所要求的语法结构。陈述 $(u, v, w) \in \mathbb{G}^3$ 的一个接受对话形如：

$$((v_{\mathrm{t}}, w_{\mathrm{t}}), c, \beta_{\mathrm{z}}) \ \text{ s.t. } \ g^{\beta_{\mathrm{z}}} = v_{\mathrm{t}} \cdot v^c, \ \ y^{\beta_{\mathrm{z}}} = w_{\mathrm{t}} \cdot w^c$$

不难证明，一个诚实证明者与一个诚实验证者之间总是能够产生接受对话。

知识健全性. 假设我们有两个陈述 (u, v, w) 的接受对话

$$((v_{\mathrm{t}}, w_{\mathrm{t}}), c, \beta_{\mathrm{z}}), \quad ((v_{\mathrm{t}}, w_{\mathrm{t}}), c', \beta'_{\mathrm{z}})$$

其中 $c \neq c'$。不难发现

$$\beta := \Delta\beta / \Delta c$$

是陈述 $(u, v, w) \in \mathbb{G}^3$ 对应的见证，其中 $\Delta\beta := \beta_{\mathrm{z}} - \beta'_{\mathrm{z}}$，$\Delta c := c - c'$。

特殊 *HVZK*. 对于输入 $(u, v, w) \in \mathbb{G}^3$ 和 $c \in \mathcal{C}$ 时，模拟器计算：

$$\beta_{\mathrm{z}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{z}}} / v^c, \quad w_{\mathrm{t}} \leftarrow u^{\beta_{\mathrm{z}}} / w^c$$

并输出 $((v_{\mathrm{t}}, w_{\mathrm{t}}), c, \beta_{\mathrm{z}})$。可以发现，输出总是会产生一个符合要求的接受对话。

下面我们论证，当 $c \in \mathcal{C}$ 是随机选出的时候，模拟器在输入 $((u, v, w), c)$ 上的输出具有正确的分布。注意到在真实的对话中，c 和 β_{z} 相互独立，其中 c 在 $\mathcal{C}$ 上均匀分布，β_{z} 在 $\mathbb{Z}_q$ 上均匀分布。此外，如果给定 c 和 β_{z}，v_{t} 和 w_{t} 的值由

$$g^{\beta_{\mathrm{z}}} = v_{\mathrm{t}} \cdot v^c, \quad u^{\beta_{\mathrm{z}}} = w_{\mathrm{t}} \cdot w^c$$

唯一决定。而这显然与模拟器的输出分布是一致的。 □

19.5.3 用于任意线性关系的 Sigma 协议

读者可能已经注意到 Schnorr、Okamoto 和 Chaum-Pedersen 协议之间的某些相似之处。事实上，它们都是用于证明一组元素之间线性关系的通用 Sigma 协议的特例。

与之前一样，令 $\mathbb{G}$ 是一个 q 阶循环群，其中 q 是素数，$g \in \mathbb{G}$ 是生成元。考察形如：

$$\phi(x_1, \ldots, x_n) := \left\{ u_1 = \prod_{j=1}^{n} g_{1j}^{x_j} \quad \wedge \quad \cdots \quad \wedge \quad u_m = \prod_{j=1}^{n} g_{mj}^{x_j} \right\} \tag{19.13}$$

的布尔表达式 ϕ。其中所有的 g_{ij} 和 u_i 都是 $\mathbb{G}$ 中的元素，这些元素中的有些可以是系统参数甚至是常数，有些是表达式特有的。ϕ 中的 x_i 是公式的变量。当我们给 $x_1, \ldots, x_n$ 赋 $\mathbb{Z}_q$ 中的值时，如果式 19.13 中的所有等式都成立，ϕ 就会输出 true。

对于这类公式的集合 $\mathcal{F}$，我们可以定义关系：

$$\mathcal{R} := \left\{ \Big((\alpha_1, \ldots, \alpha_n), \phi \Big) \in \mathbb{Z}_q^n \times \mathcal{F} : \ \phi(x_1, \ldots, x_n) = \text{true} \right\} \tag{19.14}$$

因此，陈述是一个表达式 $\phi \in \mathcal{F}$，ϕ 的见证是能使得表达式 ϕ 输出 true 的 $x_1, \ldots, x_n$ 的一组赋值 $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$。我们之所以称其为 “线性” 关系集，是因为如果我们取离散对数，式 19.13 就可以改写为如下的线性方程组：

$$\mathsf{Dlog}_g(u_j) = \sum_{j=1}^{n} x_i \cdot \mathsf{Dlog}_g(g_{ij}) \quad (i = 1, 2, \ldots, m)$$

而见证就是上述线性方程组的一组解。

图 19.8 展示了关系 $\mathcal{R}$ 上的**通用线性协议 (generic linear protocol)** (P, V)。证明者拥有 ϕ 和见证 $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$，挑战空间 $\mathcal{C}$ 是 $\mathbb{Z}_q$ 的一个子集。这样，到目前为止我们所介绍的所有 Sigma 协议都是这个通用线性协议的特例：

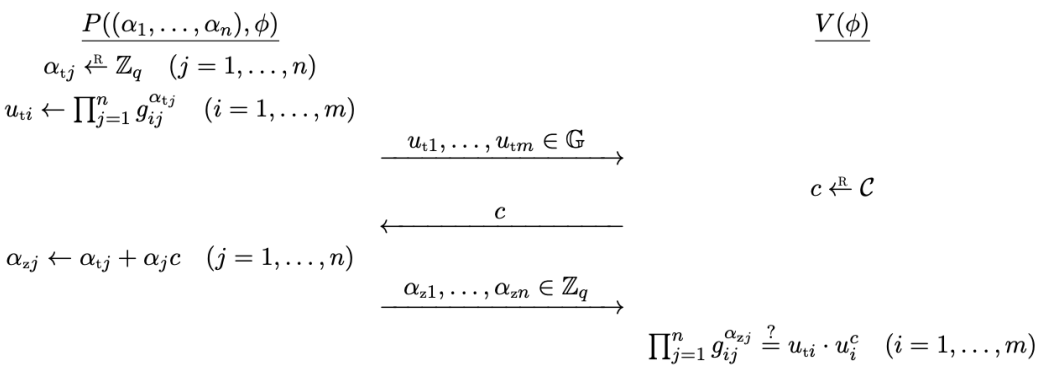


图 19.8: 通用线性协议

- Schnorr 协议对应着 $\phi_1(x) := \{u = g^x\}$ 的情况。
- Okamoto 协议对应着 $\phi_2(x,y) := \{u = g^x h^y\}$ 的情况。
- Chaum-Pedersen 协议对应着 $\phi_3(x) := \{v = g^x \wedge w = u^x\}$ 的情况。

我们可以通过模仿 Schnorr、Okamoto 和 Chaum-Pedersen 的相应定理的证明来证明以下定理。我们把它作为一个练习留给读者。

定理 19.11. *图 19.8 所描述的通用线性协议是式 19.14 中定义的关系 $\mathcal{R}$ 上的一个 Sigma 协议。此外，该协议提供知识健全性，且是特殊 HVZK 的。*

我们还可以进一步泛化这个通用线性协议，方法是允许式 19.13 中的各个方程定义在不同的群上。唯一的要求是所有群都有相同的素阶 q。这种更通用的线性协议的典型出现场景是有两类方程，其中一类定义在密码学所感兴趣的群 $\mathbb{G}$ 上，群的阶为素数 q；另一类定义在 $\mathbb{Z}_q$ 上，其形式为 $\kappa_i = \sum_{j=1}^{n} \lambda_{ij} x_j$，其中 κ_i 和 λ_{ij} 是 $\mathbb{Z}_q$ 中的元素。

19.5.4 一种用于同态原像的 Sigma 协议

到目前为止我们介绍的所有 Sigma 协议，包括通用线性协议，都可以用群同态的语言来更清楚、更简洁地描述。令 $\mathbb{H}_1$ 和 $\mathbb{H}_2$ 是两个阶已知的有限阿贝尔群，$\psi : \mathbb{H}_1 \to \mathbb{H}_2$ 是一个群同态。我们将群 $\mathbb{H}_1$ 中的运算表示为加法形式，将群 $\mathbb{H}_2$ 中的运算表示为乘法形式。

令 $u \in \mathbb{H}_2$。图 19.9 展示了一个 Sigma 协议，它允许证明者说服验证者它 “知道” u 在 ψ 下的原像。具体地，该协议是关系：

$$\mathcal{R} := \left\{ \Big(\alpha, (u,\psi)\Big) \in \mathbb{H}_1 \times (\mathbb{H}_1 \times \mathcal{F}) : \psi(\alpha) = u \right\} \tag{19.15}$$

上的一个 Sigma 协议。其中 $\alpha \in \mathbb{H}_1$ 是 $u \in \mathbb{H}_2$ 在 ψ 下的原像。图 19.9 中的证明者持有见证 $\alpha \in \mathbb{H}_1$，验证者持有像 $u \in \mathbb{H}_2$，双方共有 $(\mathbb{H}_1, \mathbb{H}_2, \psi)$。挑战空间 $\mathcal{C}$ 为 $\{0,1,\ldots,N-1\} \subseteq \mathbb{Z}$，其中 N 为特定整数。

下面让我们看看这个协议是如何包含到目前为止所有的 Sigma 协议实例的。令 $\mathbb{G}$ 是一个素阶 q 的群，且有生成元 $g, h, u \in \mathbb{G}$：

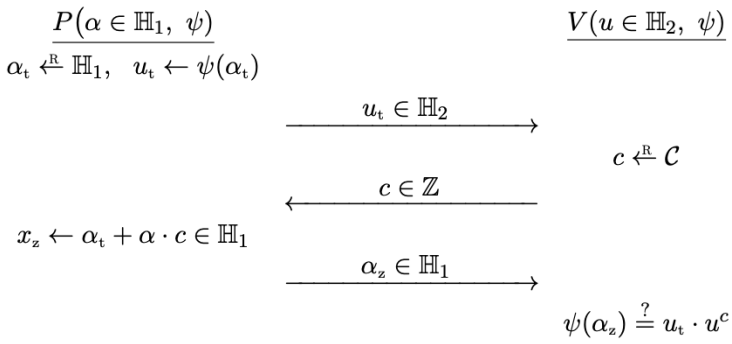


图 19.9: 用于同态原像的 Sigma 协议

- Okamoto 协议对应着 $\mathbb{H}_1 := \mathbb{Z}_q^2$，$\mathbb{H}_2 := \mathbb{G}$ 且 $\psi_1(x, y) := g^x h^y$ 的特殊情况。
- Chaum-Pedersen 协议对应着

$$\mathbb{H}_1 := \mathbb{Z}_q,\ \ \mathbb{H}_2 := \mathbb{G}^2,\ \ \psi_2(x) := (g^x, u^x)$$

  的特殊情况。我们甚至可以令 $\mathbb{H}_2 = \mathbb{G}_1 \times \mathbb{G}_2$，其中 $g \in \mathbb{G}_1$，$u \in \mathbb{G}_2$，并且 $|\mathbb{G}_1| = |\mathbb{G}_2|$。那么对于一个给定的 $(v, w) \in \mathbb{G}_1 \times \mathbb{G}_2$，提供 (v, w) 在 ψ_2 上的一个原像与计算群 $\mathbb{G}_1$ 和 $\mathbb{G}_2$ 上的离散对数问题 $\mathsf{Dlog}_g(v) = \mathsf{Dlog}_u(w)$ 等价。
- 图 19.8 中的通用线性协议对应着

$$\mathbb{H}_1 := (\mathbb{Z}_q)^n,\ \ \mathbb{H}_2 := \mathbb{G}^m,\ \ \psi_3(x_1, \ldots, x_n) := \left(\prod_{j=1}^{n} g_{1j}^{x_j},\ \ \ldots,\ \ \prod_{j=1}^{n} g_{mj}^{x_j} \right)$$

  的特殊情况。其中，对于所有的 $i = 1, \ldots, m$ 和 $j = 1, \ldots, n$，都有 $g_{ij} \in \mathbb{G}$。

我们可以很容易地验证 ψ_1, ψ_2, ψ_3 的映射都是群同态的。通过在图 19.9 中的协议使用这些同态映射，我们可以得到本节中介绍的所有 Sigma 协议实例。

定理 19.12. *图 19.9 中的协议是式 19.15 中定义的关系 $\mathcal{R}$ 上的一个 Sigma 协议。此外，该协议是特殊 HVZK 的，而且只要 $|\mathbb{H}_1| \times |\mathbb{H}_2|$ 的最小素因子至少是 $|\mathcal{C}|$，该协议就提供知识健全性。*

定理 19.12 的证明完全模仿了 Schnorr 协议相应定理的证明。对 $|\mathbb{H}_1| \times |\mathbb{H}_2|$ 的最小素因子的下限要求，是为了确保见证提取器可以从两个接受对话 (u_t, c, α_z) 和 (u_t, c', α'_z) 中获得一个映射 ψ 的原像。就如同在 Schnorr 协议中的见证提取器中那样，我们可以得到 $\psi(\Delta\alpha) = u^{\Delta c}$，其中 $\Delta\alpha := (\alpha_z - \alpha'_z) \in \mathbb{H}_1$，$\Delta c = (c - c') \in \mathbb{Z}$。限制 $|\mathbb{H}_1|$ 和 $|\mathbb{H}_2|$ 素因子的下界是为了确保 (1) 我们可以在 $\mathbb{H}_1$ 中用 Δc 除以 $\Delta\alpha$，并且 (2) 我们可以在等式两边同时升幂 $((\Delta c)^{-1} \mod |\mathbb{H}_2|) \in \mathbb{Z}$，以求得等式右项的 Δc 次方根。然后，我们就可以得到 $\psi(\Delta\alpha/\Delta c) = u$，因此 $\Delta\alpha/\Delta c \in \mathbb{H}_1$ 就是 $u \in \mathbb{H}_2$ 在映射 ψ 下的原像。

19.5.5 一个用于 RSA 的 Sigma 协议

为了避免读者认为 Sigma 协议只适用于与离散对数有关的问题，我们下面介绍一个与 RSA 有关的协议。

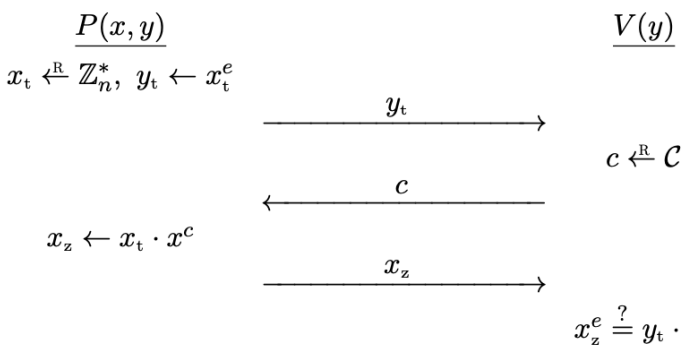


图 19.10: Guillou-Quisquater 协议

令 (n,e) 是一个 RSA 公钥，e 是一个素数。我们将 (n,e) 视为一个系统参数。Guillou-Quisquater (GQ) 协议允许证明者说服持怀疑态度的验证者，他 "知道" $y \in \mathbb{Z}_n^*$ 的一个 e 次方根，而不透露任何其他信息。更确切地说，GQ 协议是关系：

$$\mathcal{R} = \left\{ (x,y) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^* : x^e = y \right\} \tag{19.16}$$

上的一个 Sigma 协议。陈述 $y \in \mathbb{Z}_n^*$ 的见证是 $x \in \mathbb{Z}_n^*$，满足 $x^e = y$。由于 (n,e) 是一个 RSA 公钥，因此由 $x \in \mathbb{Z}_n^*$ 到 $y = x^e \in \mathbb{Z}_n^*$ 的映射是一个双射。因此每个陈述都对应着唯一的见证。

图 19.10 展示了 GQ 协议 (P,V) 的工作流程。其挑战空间 $\mathcal{C}$ 是 $\{0,\ldots,e-1\}$ 的一个子集。需要注意的是，当 e 很小时，挑战空间也很小。如果需要，我们可以用练习 19.5 中介绍的方法将其扩大。然而在使用该协议时，我们通常会将 e 设定为一个大素数来确保挑战空间是足够大的。

图 19.10 中的 GQ 协议也是图 19.9 中展示的用于同态原像的 Sigma 协议的一个特例。此时群同态是 $\psi : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$，定义为 $\psi(x) = x^e$。然而，我们无法像定理 19.12 那样证明 GO 协议提供知识健全性，因为群 $\mathbb{Z}_n^*$ 的阶是未知的。相反地，我们必须对这些属性进行单独证明。我们有下面的定理。

定理 19.13. *GQ 协议是式 19.16 中定义的关系 $\mathcal{R}$ 上的一个 Sigma 协议。此外，它提供知识健全性，并且是特殊 HVZK 的。*

证明. y 的一个接受对话形如 (x_t, c, x_z)，其中 $x_z^e = y_t \cdot y^c$。读者很容易验证基本的正确性要求，即一个诚实证明者和一个诚实验证者之间的交互总是会产生一个接受对话。

知识健全性. 下面我们将证明 GQ 协议具备知识健全性。假设我们有两个陈述 y 的接受对话 (x_t, c, x_z) 和 (x_t, c', x_z')，其中 $c \neq c'$。我们必须证明能够有效计算出 y 的 e 次方根。观察到：

$$x_z^e = y_t \cdot y^c, \quad (x_z')^e = y_t \cdot y^{c'}$$

将第一个方程除以第二个方程，可以得到：

$$(\Delta x)^e = y^{\Delta c}$$

其中：$\Delta x := x_z / x_z'$，$\Delta c := c - c'$。由于 $c \neq c'$，且 c 和 c' 都属于区间 $\{0,\ldots,e-1\}$，因此有 $0 < |\Delta c| < e$，所以 $e \nmid \Delta c$。此外由于 e 是素数，所以 $\gcd(e, \Delta c) = 1$。因此当给定 e，$f := \Delta c$，$w := \Delta x$ 时，我们可以利用定理 10.6 求得 y 的 e 次方根。

读者应该注意到，这里给出的从两个接受对话中计算 RSA 逆变换的技术与定理 10.7 的证明中使用的方法基本相同。事实上，这两个接受对话制造了哈希函数 $H_{\text{rsa}}(a,b) = a^e y^b$ 上的一次碰撞 $((x_{\text{z}}, -c \bmod e), (x'_{\text{z}}, -c' \bmod e))$。

特殊 HVZK. 最后，我们通过给出一个模拟器来证明 GQ 协议是特殊 HVZK 的。对于输入 $y \in \mathbb{Z}_n^*$ 和 $c \in \mathcal{C}$，模拟器计算：

$$x_{\text{z}} \xleftarrow{\text{R}} \mathbb{Z}_n^*, \quad y_{\text{t}} \leftarrow x_{\text{z}}^e / y^c$$

并输出 $(y_{\text{t}}, x_{\text{z}})$。注意到在真实的对话中，$c$ 和 x_{z} 是相互独立的，c 在 $\mathcal{C}$ 上均匀分布，x_z 在 $\mathbb{Z}_n^*$ 上均匀分布。此外，如果给定 c 和 x_z，y_t 的值由方程 $x_{\text{z}}^e = y_{\text{t}} \cdot y^c$ 唯一决定。而这显然与模拟器的输出分布是一致的。 □

19.6 基于 Sigma 协议的身份识别和签名

通过模仿 Schnorr 构造，我们很容易将任何 Sigma 协议转换成相应的识别方案和签名方案。

假设我们有一个关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议 (P, V)。除了算法 P 和 V 之外，我们还需要一个 $\mathcal{R}$ **上的密钥生成算法**。它是一个概率算法 G，生成一个公私钥对 (pk, sk)，其中 $pk = y$，$sk = (x, y)$，$(x, y) \in \mathcal{R}$。

为了得到安全的识别和签名方案，我们需要以下的“单向性”特征：给定 G 输出的公钥 $pk = y \in \mathcal{Y}$，应该很难计算出一个 $\hat{x} \in \mathcal{X}$ 满足 $(\hat{x}, y) \in \mathcal{R}$。我们用下面的攻击游戏来更加精准地描述这个概念。

攻击游戏 19.2 (单向密钥生成). 令 G 是 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个密钥生成算法。对于一个给定对手 $\mathcal{A}$，攻击游戏按如下方式运行：

- 挑战者运行 $(pk, sk) \xleftarrow{\text{R}} G()$，并将 $pk = y$ 发送给 $\mathcal{A}$；
- $\mathcal{A}$ 输出 $\hat{x} \in \mathcal{X}$。

如果 $(\hat{x}, y) \in \mathcal{R}$，我们就称对手赢得了攻击游戏。我们将 $\mathcal{A}$ 相对于 $\mathbb{G}$ 的优势定义为 $\mathsf{OWadv}[\mathcal{A}, \mathbb{G}]$，其值为 $\mathcal{A}$ 赢得游戏的概率。

定义 19.6. *如果对于所有有效对手 $\mathcal{A}$，$\mathsf{OWadv}[\mathcal{A}, \mathbb{G}]$ 的值都是可忽略不计的，我们就称密钥生成算法 G 是单向的。*

例 19.4. 对于例 19.1 介绍的 Schnorr 的 Sigma 协议，最自然的密钥生成算法是计算 $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$ 和 $u \leftarrow g^\alpha \in \mathbb{G}$，并输出 $pk := u$ 和 $sk := (\alpha, u)$。在离散对数假设下，这种密钥生成算法显然是单向的。

例 19.5. 考虑 19.5.5 小节介绍的 GQ 协议，其中 RSA 公钥 (n, e) 被视为一个系统参数。最自然的密钥生成算法是计算 $x \xleftarrow{\text{R}} \mathbb{Z}_n^*$ 和 $y \leftarrow x^e \in \mathbb{Z}_n^*$，并输出 $pk := y$ 和 $sk := (x, y)$。在 RSA 假设下，这种密钥生成算法显然也是单向的（见定理 10.5）。

一个含有密钥生成算法 G 的 Sigma 协议 (P, V) 能够提供一个身份识别方案 (G, P, V)。接下来的两个定理将证明它对窃听攻击是安全的。

定理 19.14. *令 (P, V) 是一个具有大挑战空间的有效关系 $\mathcal{R}$ 上的 Sigma 协议。令 G 是 $\mathcal{R}$ 的一个密钥生成算法。如果 (P, V) 提供知识健全性，并且 G 是单向的，那么身份识别方案 $\mathcal{I} := (G, P, V)$ 对于直接攻击是安全的。*

特别地，假设 $\mathcal{A}$ 是一个有效冒充对手，他通过攻击游戏 *18.1* 中的直接攻击来攻击 $\mathcal{I}$，其优势为 $\epsilon := \mathrm{ID1adv}[\mathcal{A}, \mathcal{I}]$。则必然存在一个有效对手 $\mathcal{B}$ 如攻击游戏 *19.2* 那样攻击 G，其运行时间大约是 $\mathcal{A}$ 的两倍，其优势 $\epsilon' := \mathrm{OWadv}[\mathcal{A}, \mathbb{G}]$，那么：

$$\epsilon' \geq \epsilon^2 - \frac{\epsilon}{N} \tag{19.17}$$

其中 N 是挑战空间的大小，这意味着：

$$\epsilon \leq \frac{1}{N} + \sqrt{\epsilon'} \tag{19.18}$$

证明. 我们可以模仿定理 19.1 的证明。利用冒充对手 $\mathcal{A}$，我们可以使用下述方法建立一个打破 G 的单向性的对手 $\mathcal{B}$。对手 $\mathcal{B}$ 从其挑战者那里得到一个公钥 $pk = y$，而我们的目标是让 $\mathcal{B}$ 在 $\mathcal{A}$ 的帮助下计算出一个满足 $(\hat{x}, y) \in \mathcal{R}$ 的 $\hat{x}$。$\mathcal{B}$ 的行为可以分为两个阶段。

在第一阶段，$\mathcal{B}$ 扮演 $\mathcal{A}$ 的挑战者的角色，给 $\mathcal{A}$ 提供值 $pk = y$ 作为验证公钥。使用与定理 19.1 的证明中相同的回溯方法，对手 $\mathcal{B}$ 可以以至少 $\epsilon^2 - \epsilon/N$ 的概率获得 y 的两个接受对话 (t, c, z) 和 (t, c', z')，其中 $c \neq c'$。更具体地说，$\mathcal{B}$ 等待来自 $\mathcal{A}$ 的承诺 t，接着向 $\mathcal{A}$ 发送一个随机挑战 c，并等待 $\mathcal{A}$ 的应答 z。在此之后，$\mathcal{B}$ 将 $\mathcal{A}$ 的内部状态回溯到它产生 t 的那一时刻，然后向 $\mathcal{A}$ 发送另一个随机挑战 c'，并等待 $\mathcal{A}$ 的应答 z'。根据回溯引理（引理 19.2），这个过程将以至少 $\epsilon^2 - \epsilon/N$ 的概率产生两个接受对话。

在计算的第二阶段，$\mathcal{B}$ 将这两个接受对话送入见证提取器（由知识健全性保证存在），以为 y 提取一个见证 $\hat{x}$。

这样我们就证明了式 19.17，而式 19.18可以通过与定理 19.1 相同的方式得到。 □

定理 19.3 显然适用于从特殊 HVZK 的 Sigma 协议衍生出来的身份识别协议：

定理 19.15. 令 (P, V) 是一个有效关系 $\mathcal{R}$ 上的 *Sigma* 协议。令 G $\mathcal{R}$ 的一个密钥生成算法。如果一个身份识别协议 $\mathcal{I} = (G, P, V)$ 对于直接攻击是安全的，且 (P, V) 是特殊 *HVZK* 的，那么 $\mathcal{I}$ 对于窃听攻击也是安全的。

特别地，对于每一个通过攻击游戏 *18.2* 中的窃听攻击来攻击 $\mathcal{I}$ 的冒充对手 $\mathcal{A}$，必然存在一个通过攻击游戏 *18.1* 中的直接攻击来攻击 $\mathcal{I}$ 的对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，使得：

$$\mathrm{ID2adv}[\mathcal{A}, \mathcal{I}] = \mathrm{ID1adv}[\mathcal{B}, \mathcal{I}]$$

例 19.6. 如果我们用例 19.5 中的密钥生成算法 G 来增强 GQ 协议 (P, V)，我们就能够得到识别方案 $\mathcal{I}_{\mathrm{GQ}} = (G, P, V)$，只要挑战空间足够大，该识别方案在 RSA 假设下对窃听攻击就是安全的。

19.6.1 用于签名的 Fiat-Shamir 启发式算法

使用 19.2 节中介绍的技术，我们就可以将 Sigma 协议转换为签名方案。这个普适的技术最早由 Fiat 和 Shamir 提出，它包含如下几个部分：

- 一个关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议 (P, V)。我们假设对话形如 (t, c, z)，其中 $t \in \mathcal{T}$，$c \in \mathcal{C}$，$z \in \mathcal{Z}$；
- 一个关系 $\mathcal{R}$ 的密钥生成算法 G；

- 一个哈希函数 $H : \mathcal{M} \times \mathcal{T} \to \mathcal{C}$，它被建模为一个随机预言机。集合 $\mathcal{M}$ 也同样是签名方案的消息空间。

由 G 和 (P, V) 派生出的 **Fiat-Shamir 签名方案**的工作原理如下：

- 密钥生成算法为 G，所以公钥形如 $pk = y$, 其中 $y \in \mathcal{Y}$，私钥形如 $sk = (x, y) \in \mathcal{R}$。
- 使用私钥 $sk = (x, y)$ 对消息 $m \in \mathcal{M}$ 进行签名，签名算法运行如下：
  - 运行验证者 $P(x, y)$ 并获得一个承诺 $t \in \mathcal{T}$；
  - 计算挑战 $c \leftarrow H(m, t)$；
  - 最后，将 c 发送给证明者并得到一个应答 z，然后输出签名 $\sigma := (t, z) \in \mathcal{T} \times \mathcal{Z}$。
- 为了使用公钥 $pk = y$ 验证对消息 $m \in \mathcal{M}$ 的签名 $\sigma = (t, z) \in \mathcal{T} \times \mathcal{Z}$，验证算法计算 $c \leftarrow H(m, t)$，并检查 (t, c, z) 是否是 y 的接受对话。

正如我们对 Schnorr 协议所做的那样，我们下面会证明如果对应的识别方案 (G, P, V) 对窃听攻击是安全的，那么 Fiat-Shamir 签名方案在随机预言机模型下是安全的。然而，我们还需要一个技术上的假设，基本上所有我们感兴趣的 Sigma 协议都满足这个假设。

定义 19.7 (不可预测承诺). *令 (P, V) 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议，并假定所有对话 (t, c, z) 都位于 $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$ 中。如果对于任意 (x, y) 和 $\hat{t} \in \mathcal{T}$，$P(x, y)$ 和 $V(y)$ 之间的交互会以最多 δ 的概率产生一个对话 (t, c, z) 满足 $t = \hat{t}$，则称 (P, V) 有 δ-**不可预测承诺** (δ-**unpredictable commitments**)。如果 (P, V) 有 δ-不可预测承诺，且 δ 的值不可忽略不计，我们就称 (P, V) 有不可预测承诺。*

定理 19.16. *如果 H 被建模为一个随机预言机，身份识别方案 $\mathcal{I} = (G, P, V)$ 对窃听攻击是安全的，并且 (P, V) 有不可预测承诺，那么从 G 和 (P, V) 派生出的 Fiat-Shamir 签名方案 $\mathcal{S}$ 是安全的。*

> *特别地，令 $\mathcal{A}$ 是一个像在攻击游戏 13.1 的随机预言机版本中一样的攻击 $\mathcal{S}$ 的对手。此外，假设 $\mathcal{A}$ 最多可以发起 Q_{s} 次签名查询和 Q_{ro} 次随机预言机查询，并且 (P, V) 有 δ-不可预测承诺。那么必然存在一个 $(Q_{\mathrm{ro}} + 1)$ 次重复冒充攻击对手 $\mathcal{B}$，它能够通过攻击游戏 19.1 中的窃听攻击来攻击 $\mathcal{I}$，其中 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，使得：*
>
> $$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}] \leq Q_{\mathrm{s}}(Q_{\mathrm{s}} + Q_{\mathrm{ro}} + 1)\delta + \mathrm{rID2}\mathsf{adv}[\mathcal{B}, \mathcal{I}, Q_{\mathrm{ro}} + 1]$$

这个定理的证明与定理 19.7 的证明几乎相同，读者可以尝试自己证明该定理。

把以上结论结合起来，假设我们从一个 Sigma 协议 (P, V) 开始，该协议是特殊 HVZK 的，并且它能提供知识健全性。此外，假设 (P, V) 有不可预测承诺和一个大的挑战空间。那么，如果我们把 (P, V) 和一个单向密钥生成算法 G 结合起来，就可以利用 Fiat-Shamir 签名构造法基于一个随机预言机 H 构建一个安全的签名方案。事实上，Schnorr 签名方案就是这个结构的一个特例。

就像我们对 Schnorr 签名所做的那样，我们也可以使用引理 19.6 将问题从 r 次重复冒充规约到 1 次冒充。但是更严格的规约也是有可能的，事实上引理 19.8 的证明在这里也是成立的，基本上不需要更改太多内容：

引理 19.17. *令 (P, V) 是一个关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的特殊 HVZK 的 Sigma 协议，G 是 $\mathcal{R}$ 上的一个密钥生成算法，考虑由此产生的身份识别协议 $\mathcal{I} = (G, P, V)$。假设 $\mathcal{A}$ 是一个如攻击游戏 19.1 中那样的针*

对 $\mathcal{I}$ 的有效的 r 次重复冒充窃听攻击对手，其优势为 $\epsilon := \mathrm{rID2adv}[\mathcal{A}, \mathcal{I}, r]$。那么必然存在一个如攻击游戏 *19.2* 中那样的针对 G 的有效对手 $\mathcal{B}$，其运行时间大约是 $\mathcal{A}$ 的两倍，其优势为 $\epsilon := \mathrm{OWadv}[\mathcal{B}, G]$，使得：

$$\epsilon' \geq \frac{\epsilon^2}{r} - \frac{\epsilon}{N} \tag{19.19}$$

其中 N 是挑战空间的大小，这意味着：

$$\epsilon \leq \frac{r}{N} + \sqrt{r\epsilon'} \tag{19.20}$$

利用这一点，在 (P, V) 是特殊 HVZK 的情况下，我们可以进一步确定定理 19.16 中给出的安全上界：

> 假设 $\mathcal{A}$ 是一个如攻击游戏 *13.1* 的随机预言机版本那样攻击 $\mathcal{S}$ 的有效对手。此外，假设 $\mathcal{A}$ 最多发起 Q_{s} 次签名查询和 Q_{ro} 次随机预言机查询。那么必然存在一个如攻击游戏 *19.2* 中那样的针对 G 的有效对手 $\mathcal{B}$，其运行时间大约是 $\mathcal{A}$ 的两倍，使得：
>
> $$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}] \leq Q_{\mathrm{s}}(Q_{\mathrm{s}} + Q_{\mathrm{ro}} + 1)\delta + \frac{Q_{\mathrm{ro}} + 1}{N} + \sqrt{(Q_{\mathrm{ro}} + 1) \cdot \mathrm{OWadv}[\mathcal{B}, G]} \tag{19.21}$$
>
> 其中 N 是挑战空间的大小。

19.6.1.1 GQ 签名方案

将上面介绍的 Fiat-Shamir 签名构造应用于 GQ 协议（见 19.5.5 小节），我们就可以构造出一个基于 RSA 的新签名方案。该方案利用 RSA 公钥 (n, e) 作为系统参数，其中的指数 e 是一个大素数。如果有需要的话，该系统参数可以被许多用户共享。我们需要一个哈希函数 $H : \mathcal{M} \times \mathcal{T} \to \mathcal{C}$，其中 $\mathcal{T}$ 是由 $\mathbb{Z}_n^*$ 中的所有元素编码后组成的集合，$\mathcal{M}$ 是签名方案的消息空间，$\mathcal{C}$ 是 $\{0, \ldots, e-1\}$ 的一个子集。GQ 签名方案 $\mathcal{S}_{\mathrm{GQ}} = (G, S, V)$ 的组成部分如下：

- 密钥生成算法 G 计算：

  $$x \xleftarrow{\mathrm{R}} \mathbb{Z}_n^*, \quad y \leftarrow x^e$$

  公钥为 $pk := y$，私钥为 $sk := x$。

- 为了使用私钥 $sk = x$ 对消息 $m \in \mathcal{M}$ 签名，签名算法 $S(sk, m)$ 计算：

  $$x_{\mathrm{t}} \xleftarrow{\mathrm{R}} \mathbb{Z}_n^*, \quad y_{\mathrm{t}} \leftarrow x_{\mathrm{t}}^e, \quad c \leftarrow H(m, y_{\mathrm{t}}), \quad x_{\mathrm{z}} \leftarrow x_{\mathrm{t}} \cdot x^c$$

  输出签名 $\sigma := (y_{\mathrm{t}}, x_{\mathrm{z}})$。

- 为了使用公钥 $pk = y$ 验证对消息 $m \in \mathcal{M}$ 的签名 $\sigma = (y_{\mathrm{t}}, x_{\mathrm{z}})$，签名验证算法 V 计算 $c := H(m, y_t)$，当 $x_{\mathrm{z}}^e = y_{\mathrm{t}} \cdot y^c$ 时输出 accept，否则输出 reject。

正如我们在例 19.6 中看到的那样，在 RSA 假设下，只要挑战空间足够大，GQ 识别方案对窃听攻击就是安全的。另外，可以观察到 GQ 协议有 $1/\phi(n)$-不可预测承诺。由定理 19.16 可知，在 RSA 假设下，相应的签名方案在随机预言机模型下是安全的。

GQ 签名相比 RSA 签名（如 $\mathcal{S}_{\mathrm{RSA-FDH}}$）的优势在于，它的签名算法要快得多。使用 $\mathcal{S}_{\mathrm{RSA-FDH}}$ 签名需要进行一个大指数运算，而 GQ 签名虽然需要进行 e 和 c 这两个指数运算，但这两个指数都只有

128 位。当签名者是一个计算力孱弱的设备时，快速签名的能力是很重要的，比如使用芯片的信用卡在每笔交易中签名的场景。

一个优化. GQ 签名方案可以用和 Schnorr 签名方案相同的方式进行优化。在之前，我们定义的对消息 m 的签名是一个数对 $(y_{\mathrm{t}}, x_{\mathrm{z}})$，它满足：

$$x_{\mathrm{z}}^{e} = y_{\mathrm{t}} \cdot y^{c}$$

其中 $c := H(m, y_{\mathrm{t}})$。为了进一步优化该签名方案，我们可以将签名定义为数对 (c, x_{z})，它满足：

$$c = H(m, y_{\mathrm{t}})$$

其中 $y_{\mathrm{t}} := x_{\mathrm{z}}^{e}/y^{c}$。此外，我们可以在公钥中存储 y^{-1} 而不是 y，这能够进一步将加快验证速度。

事实证明，这种优化也可以应用于 Fiat-Shamir 签名构造的大多数实例，参见练习 19.17。

19.7 Sigma 协议的组合：AND 和 OR 证明

在本节中，我们将展示如何将 Sigma 协议组合起来构成新的 Sigma 协议。在 AND 证明结构中，证明者可以说服验证者他 “知道” 一对陈述的见证。在 OR 证明结构中，证明者可以说服验证者他 “知道” 两个陈述中其中一个的见证。

19.7.1 AND 证明构造

假设我们有一个 $\mathcal{R}_0 \subseteq \mathcal{X}_0 \times \mathcal{Y}_0$ 上的 Sigma 协议 (P_0, V_0) 和一个 $\mathcal{R}_1 \subseteq \mathcal{X}_1 \times \mathcal{Y}_1$ 上的 Sigma 协议 (P_1, V_1)。此外，我们假设这两个 Sigma 协议都使用相同的挑战空间 $\mathcal{C}$。我们可以把它们组合起来构造一个关系：

$$\mathcal{R}_{\mathrm{AND}} = \left\{ \big((x_0, x_1), (y_0, y_1)\big) \in (\mathcal{X}_0 \times \mathcal{X}_1) \times (\mathcal{Y}_0 \times \mathcal{Y}_1) : \ (x_0, y_0) \in \mathcal{R}_0, \ (x_1, y_1) \in \mathcal{R}_1 \right\} \tag{19.22}$$

上的 Sigma 协议。换句话说，给定一对陈述 $y_0 \in \mathcal{Y}_0$ 和 $y_1 \in \mathcal{Y}_1$，这个 **AND 协议**允许证明者说服持怀疑态度的验证者，他同时 “知道” y_0 的一个见证和 y_1 的一个见证。该协议 (P, V) 运行如下，其中验证者 P 由 $((x_0, x_1), (y_0, y_1)) \in \mathcal{R}_{\mathrm{AND}}$ 初始化，验证者 V 由 $(y_0, y_1) \in \mathcal{Y}_0 \times \mathcal{Y}_1$ 初始化：

1. P 运行 $P_0(x_0, y_0)$ 得到一个承诺 t_0，运行 $P_1(x_1, y_1)$ 得到一个承诺 t_1，并将承诺对 (t_0, t_1) 发送给 V；
2. V 选取 $c \xleftarrow{\mathrm{R}} \mathcal{C}$，并将挑战 c 发送给 P；
3. P 将挑战 c 投入 $P_0(x_0, y_0)$ 和 $P_1(x_1, y_1)$，得到应答 z_0 和 z_1，并将应答对 (z_0, z_1) 发送给 V；
4. V 检查 (t_0, c, z_0) 是否是 y_0 的一个接受对话，(t_1, c, z_1) 是否是 y_1 的一个接受对话。

定理 19.18. *AND 协议 (P, V) 是式 19.22 中定义的关系 $\mathcal{R}_{\mathrm{AND}}$ 上的 Sigma 协议。若 (P_0, V_0) 和 (P_1, V_1) 提供知识健全性，则 (P, V) 亦然。若 (P_0, V_0) 和 (P_1, V_1) 都是特殊 HVZK 的，则 (P, V) 亦然。*

证明简述. 正确性显然成立。

对于知识健全性，如果 (P_0, V_0) 有提取器 Ext_0，(P_1, V_1) 有提取器 Ext_1，那么 (P, V) 的提取器是：

$$Ext\Big((y_0, y_1), ((t_0, t_1), c, (z_0, z_1)),\ \ ((t_0, t_1), c', (z_0', z_1'))\Big) := \\ \Big(Ext_0(y_0, (t_0, c, z_0), (t_0, c', z_0')),\ \ Ext_1(y_1, (t_1, c, z_1), (t_1, c', z_1'))\Big)$$

对于特殊 HVZK，如果 (P_0, V_0) 有模拟器 Sim_0，(P_1, V_1) 有模拟器 Sim_1，那么 (P, V) 的模拟器是：

$$Sim((y_0, y_1), c) := ((t_0, t_1), (z_0, z_1))$$

其中：

$$(t_0, z_0) \xleftarrow{\mathrm{R}} Sim_0(y_0, c),\ \ (t_1, z_1) \xleftarrow{\mathrm{R}} Sim_1(y_1, c)$$

读者可以自己填补证明的细节部分。我们必须指出，在构建模拟器 Sim 的过程中，我们利用了这样一个事实，即在特殊 HVZK 的定义中，挑战是对模拟器的输入，我们可以将其交给 Sim_0 和 Sim_1 两方。 □

19.7.2 OR 证明构造

假设我们有一个 $\mathcal{R}_0 \subseteq \mathcal{X}_0 \times \mathcal{Y}_0$ 上的 Sigma 协议 (P_0, V_0) 和一个 $\mathcal{R}_1 \subseteq \mathcal{X}_1 \times \mathcal{Y}_1$ 上的 Sigma 协议 (P_1, V_1)。我们还要附加几个假设：

- 这两个 Sigma 协议都使用相同的挑战空间 $\mathcal{C}$，形如 $\mathcal{C} = \{0,1\}^n$。(注意，在我们看到的挑战是数字的例子中，我们总是可以将比特串编码为二进制符号的数字。)
- 这两个 Sigma 协议都是特殊 HVZK 的，它们分别有模拟器 Sim_0 和 Sim_1。

我们可以把它们组合起来构造一个关系：

$$\mathcal{R}_{\mathrm{OR}} = \left\{ \big((b, x), (y_0, y_1)\big) \in \big(\{0,1\} \times (\mathcal{X}_0 \cup \mathcal{X}_1)\big) \times \big(\mathcal{Y}_0 \times \mathcal{Y}_1\big) :\ (x, y_b) \in \mathcal{R}_b \right\} \tag{19.23}$$

上的 Sigma 协议。换句话说，给定一对陈述 $y_0 \in \mathcal{Y}_0$ 和 $y_1 \in \mathcal{Y}_1$，这个 **OR 协议**允许证明者说服持怀疑态度的验证者，他 “知道” y_0 的一个见证或 y_1 的一个见证，除此之外不会再透露任何其他信息。更具体地，该协议不会透露证明者所拥有的到底是 y_0 的还是 y_1 的见证。

协议 (P, V) 运行如下，其中验证者 P 由 $((b, x), (y_0, y_1)) \in \mathcal{R}_{\mathrm{OR}}$ 初始化，验证者 V 由 $(y_0, y_1) \in \mathcal{Y}_0 \times \mathcal{Y}_1$ 初始化，且 $d := 1 - b$：

1. P 选取 $c_d \xleftarrow{\mathrm{R}} \mathcal{C}$ 并计算 $(t_d, z_d) \xleftarrow{R} Sim_d(y_d, c_d)$
   P 运行 $P_b(x, y_b)$ 得到一个承诺 t_b，并将承诺对 (t_0, t_1) 发送给 V；
2. V 选取 $c \xleftarrow{\mathrm{R}} \mathcal{C}$，并将挑战 c 发送给 P；
3. P 计算 $c_b \leftarrow c \oplus c_d$
   P 将挑战 c_b 转发给 $P_b(x, y_b)$，得到一个响应 z_b，并将 (c_0, z_0, z_1) 发送给 V；
4. V 计算 $c_1 \leftarrow c \oplus c_0$，并检查 (t_0, c_0, z_0) 是否是 y_0 的接受对话，以及 (t_1, c_1, z_1) 是否是 y_1 的接受对话。

定理 19.19. *OR* 协议 (P,V) 是式 *19.23* 中定义的关系 $\mathcal{R}_{\mathrm{OR}}$ 上的特殊 *HVZK* 的 *Sigma* 协议。若 (P_0,V_0) 和 (P_1,V_1) 提供知识健全性，则 (P,V) 亦然。

证明简述. 正确性显然成立。

对于知识健全性，如果 (P_0,V_0) 有提取器 Ext_0，(P_1,V_1) 有提取器 Ext_1，那么 (P,V) 的提取器 Ext 的输入是 (y_0,y_1) 和一对接受对话：

$$\big((t_0,t_1),\ c,\ (c_0,z_0,z_1)\big),\quad \big((t_0,t_1),\ c',\ (c_0',z_0',z_1')\big)$$

令 $c_1 := c\oplus c_0$，$c_1' := c\oplus c_0'$。一个重要的观察是，如果 $c\neq c'$，那么 $c_0\neq c_0'$ 和 $c_1\neq c_1'$ 中必有一个成立。因此 Ext 按如下方式运行：

如果 $c_0\neq c_0'$
　　则输出 $\Big(0,\ Ext_0(y_0,(t_0,c_0,z_0),(t_0,c_0',z_0'))\Big)$
　　否则输出 $\Big(1,\ Ext_1(y_1,(t_1,c_1,z_1),(t_1,c_1',z_1'))\Big)$

对于特殊 HVZK，(P,V) 的模拟器是：

$$Sim((y_0,y_1),c) := ((t_0,t_1),(c_0,z_0,z_1))$$

其中：

$$c_0 \xleftarrow{\mathrm{R}} \mathcal{C},\quad c_1 \leftarrow c\oplus c_0,\quad (t_0,z_0)\xleftarrow{\mathrm{R}} Sim_0(y_0,c_0),\quad (t_1,z_1)\xleftarrow{\mathrm{R}} Sim_1(y_1,c_1)$$

读者可以自己填补证明的细节部分。我们必须指出，为了确保正确性，我们利用了这样一个事实：对于特殊 HVZK，模拟器总是会输出一个接受对话。 □

19.8 见证独立性及其应用

我们下面研究 Sigma 协议的一个有用的属性，称为**见证独立性 (witness independence)**。

我们已经知道，一个给定陈述可能有若干个对应的见证。粗略地说，见证独立性意味着，如果一个“作弊”的验证者 V^*（他不需要遵守协议）与诚实证明者 P 交互，V^* 无法知道 P 在使用哪个见证。特别地，即使 V^* 非常强大或者非常聪明，以至于它在与 P 交互后能够计算出一个见证，这个见证也与 P 所使用的见证无关。当然，只有在一个特定陈述对应着一个以上的见证时，这个属性才有意义。

下面，我们首先更精确地定义这个属性。其次我们将表明，特殊 HVZK 能够导出见证独立性。这也许有点令人惊讶，因为 HVZK 是一个关于诚实验证者的属性，而见证独立性适用于所有的验证者（甚至是没有计算性上界的作弊验证者）。最后，我们会展示如何使用见证独立性来设计能够抵抗主动攻击的身份识别协议。这些识别协议简单而高效，其安全性可以基于离散对数假设或 RSA 假设，并且不依赖随机预言机启发法。

19.8.1 见证独立性的定义

我们用一个攻击游戏来定义见证独立性。

攻击游戏 19.3 (见证独立性). 令 $\Pi = (P, V)$ 是 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个 Sigma 协议。对于一个给定对手 $\mathcal{A}$，我们为每个 $(x, y) \in \mathcal{R}$ 定义一个实验 (x, y)，它按如下方式运行：

- 最初，对手 $\mathcal{A}$ 被赋予一个值 y。
- 然后，对手 $\mathcal{A}$ 与证明者 $P(x, y)$ 的几个实例进行交互。在这些交互中，挑战者进行证明者的计算，而对手扮演作弊验证者的角色（即他不需要遵循 V 的协议）。这些交互可能是并行的（特别地，对手可能会发出挑战，这些挑战取决于所有证明者实例到目前为止所输出的承诺和应答）。
- 在游戏结束时，对手 $\mathcal{A}$ 输出某个值 s，它属于一个有限的**输出空间** $\mathcal{S}$（该空间可能取决于 $\mathcal{A}$）。

对于每个 $(x, y) \in \mathcal{R}$ 和 $s \in \mathcal{S}$，我们定义 $\theta_{\mathcal{A},\Pi}(x, y, s)$ 为 $\mathcal{A}$ 在实验 (x, y) 中输出 s 的概率。

定义 19.8. *令 $\Pi = (P, V)$ 是 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个 Sigma 协议。如果对于任意对手 $\mathcal{A}$ 和任意满足 $(x, y) \in \mathcal{R}$ 与 $(x', y) \in \mathcal{R}$ 的 $y \in \mathcal{Y}$ 和 $x, x' \in \mathcal{X}$，以及任意 $\mathcal{A}$ 的输出空间中的 s，都有：*

$$\theta_{\mathcal{A},\Pi}(x, y, s) = \theta_{\mathcal{A},\Pi}(x', y, s)$$

*我们就称 (P, V) 是**见证独立**的。*

该定义指出，对于任意 $y \in \mathcal{Y}$ 和 $s \in \mathcal{S}$，$\theta_{\mathcal{A},\Pi}(x, y, s)$ 的值对于所有满足 $(x, y) \in \mathcal{R}$ 的 $x \in \mathcal{X}$ 都相同。注意，在这个定义中，$\mathcal{A}$ 不需要是有效的。我们还可以注意到，在该定义中，如果 Sigma 协议使用了一个系统参数，而这个参数本身可能是随机产生的，我们坚持认为该属性应该对每个可能的系统参数的选择都成立。

这个定义在很强的意义上抓住了一个想法，即对手的行为只取决于陈述，而不取决于证明者所选用的特定见证。

在分析识别方案时，有时可以很方便地应用见证独立性的定义。假设 (P, V) 是 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议，G 是 $\mathcal{R}$ 的一个密钥生成算法。假设我们运行密钥生成算法得到 $pk = y$ 和 $sk = (x, y)$，然后用对手 $\mathcal{A}$ 运行攻击游戏 19.3 中的实验 (x, y)。让我们定义随机变量 X，Y，S 如下：

- X 代表由 G 生成的见证 x；
- Y 代表由 G 生成的陈述 y；
- S 代表对手的输出 $s \in \mathcal{S}$。

事实 19.20. *如果 (P, V) 是见证独立的，那么对于所有的 $(x, y) \in \mathcal{R}$ 和 $s \in \mathcal{S}$，我们都有：*

$$\Pr[\mathsf{X} = x \wedge \mathsf{S} = s \,|\, \mathsf{Y} = y] = \Pr[\mathsf{X} = x \,|\, \mathsf{Y} = y] \cdot \Pr[\mathsf{S} = s \,|\, \mathsf{Y} = y] \tag{19.24}$$

我们把事实 19.20 的证明留给读者，作为一个简单的练习。式 19.24 表明，对于任何特定的以 $\mathsf{Y} = y$ 为条件的 y，随机变量 X 和 S 都是独立的。我们可以用许多不同的方式重写式 19.24，举例来说，它等价于：

$$\Pr[\mathsf{X} = x \,|\, \mathsf{S} = s \wedge \mathsf{Y} = y] = \Pr[\mathsf{X} = x \,|\, \mathsf{Y} = y] \tag{19.25}$$

例 19.7. 下面的定理 19.21 将表明，19.7.2 小节介绍的 OR 协议和 19.5.1 小节介绍的 Okamoto 协议都是见证独立的协议。

19.8.2 特殊 HVZK 意味着见证独立性

我们下面证明特殊 HVZK 能够导出见证独立性。

定理 19.21 (特殊 HVZK $\Longrightarrow$ 见证独立性). *如果一个 Sigma 协议是特殊 HVZK 的，那么它必然也是见证独立的。*

证明. 令 (P,V) 是 $\mathcal{R}\subseteq\mathcal{X}\times\mathcal{Y}$ 上的一个 Sigma 协议。假设所有的对话 (t,c,z) 都落在 $\mathcal{T}\times\mathcal{C}\times\mathcal{Z}$ 中。

令 Coins 为一个随机变量，它代表 P 可能做出的随机选择 $coins$。比如在 Schnorr 协议中，$coins$ 就是 $\alpha_{\mathrm{t}}\in\mathbb{Z}_q$ 的取值，并且 Coins 在 $\mathbb{Z}_q$ 上均匀分布。证明者 P 的逻辑可以完全由某个函数 γ 来描述，该函数将 $(x,y,c,coins)$ 映射到 (t,z) 上，其中 $(x,y)\in\mathcal{R}$，$(t,c,z)\in\mathcal{T}\times\mathcal{C}\times\mathcal{Z}$。

考虑 $P(x,y)$ 和 $V(y)$ 之间的真实交互产生一个特定对话 (t,c,z) 的概率，其值为：

$$\Pr[\gamma(x,y,c,\mathsf{Coins})=(t,z)]\,/\,|\mathcal{C}| \tag{19.26}$$

现在考虑一个由特殊 HVZK 属性保证的模拟器 Sim。对于任意 $(x,y)\in\mathcal{R}$，$c\in\mathcal{C}$ 和 $(t,z)\in\mathcal{T}\times\mathcal{Z}$，我们定义 $p(y,t,c,z)$ 为 $Sim(y,c)$ 输出 (t,z) 的概率。在随机挑战上运行模拟器所产生的对话等于特定对话 (t,c,z) 的概率为：

$$p(y,t,c,z)\,/\,|\mathcal{C}| \tag{19.27}$$

由于式 19.26 和式 19.27 中的概率必然相等，我们能够得出结论，对于任意的 $(x,y)\in\mathcal{R}$ 和 $(t,c,z)\in\mathcal{T}\times\mathcal{C}\times\mathcal{Z}$，都有：

$$\Pr[\gamma(x,y,c,\mathsf{Coins})=(t,z)]=p(y,t,c,z)$$

且该值并不取决于 x。虽然细节有点复杂，但这个事实是证明的关键。

现在考虑攻击游戏 19.3 中的实验 (x,y)，假设对手 $\mathcal{A}$ 与证明者 P 的 Q 个拷贝交互。我们用函数 γ^* 来描述所有这些证明者实例的逻辑，该函数将 $(x,y,c^*,coins^*)$ 映射到 (t^*,z^*) 上，其中 t^*，c^*，z^* 和 $coins^*$ 是长度为 Q 的向量。此外，如果用 Coins^* 表示随机变量 Coins 的 Q 个独立副本组成的向量，那么对于任意 $(x,y)\in\mathcal{R}$ 和 $(t^*,c^*,z^*)\in\mathcal{T}^Q\times\mathcal{C}^Q\times\mathcal{Z}^Q$，我们都有：

$$\Pr[\gamma^*(x,y,c^*,\mathsf{Coins}^*)=(t^*,z^*)]=\prod_i p(y,t^*[i],c^*[i],z^*[i])$$

这个概率也是不取决于 x 的。

令 Coins' 为一个随机变量，它代表对手可能做出的随机选择 $coins'$。用函数 γ' 描述对手的运行逻辑，该函数将 $(y,t^*,z^*,coins')$ 映射到 (c^*,s) 上，其中 $(t^*,c^*,z^*)\in\mathcal{T}^Q\times\mathcal{C}^Q\times\mathcal{Z}^Q$，$s\in\mathcal{S}$ 是对手的输出，$coins'$ 表示对手的特定随机选择。

令 $\mathsf{S}_{x,y}$ 为一个随机变量，它代表对手 $\mathcal{A}$ 在攻击游戏中的实验 (x,y) 中的输出。令 $\mathsf{T}_{x,y}$ 为一个随机变量，它代表可能的交互记录 $t=(t^*,c^*,z^*)$。对于 $s\in\mathcal{S}$ 和 $t=(t^*,c^*,z^*)$，定义事件 $\mathbf{\Gamma}^*(x,y;t)$ 和 $\mathbf{\Gamma}'(y,s;t)$ 如下：

$$\mathbf{\Gamma}^*(x,y;t):\gamma^*(x,y,c^*,\mathsf{Coins}^*)=(t^*,z^*),\qquad \mathbf{\Gamma}'(y,s;t):\gamma'(y,t^*,z^*,\mathsf{Coins}')=(c^*,s)$$

注意，$\mathbf{\Gamma}^*(x,y;t)$ 和 $\mathbf{\Gamma}'(y,s;t)$ 是独立事件。另外，正如我们上面所观察到的，概率 $\Pr[\mathbf{\Gamma}^*(x,y;t)]$ 与 x 无关。

对于 $s \in \mathcal{S}$，通过对所有可能的交互记录 t 进行求和，用全概率公式可以计算出 $\Pr[\mathsf{S}_{x,y} = s]$：

$$\begin{aligned}\Pr[\mathsf{S}_{x,y} = s] &= \sum_t \Pr[\mathsf{S}_{x,y} = s \wedge \mathsf{T}_{x,y} = t] \\ &= \sum_t \Pr[\boldsymbol{\Gamma}^*(x,y;t) \wedge \boldsymbol{\Gamma}'(y,s;t)] \\ &= \sum_t \Pr[\boldsymbol{\Gamma}^*(x,y;t)] \cdot \Pr[\boldsymbol{\Gamma}'(y,s;t)] \quad \text{（独立性）}\end{aligned}$$

在最后一个式子中，我们可以观察到 $\Pr[\boldsymbol{\Gamma}^*(x,y;t)]$ 和 $\Pr[\boldsymbol{\Gamma}'(y,s;t)]$ 都与 x 无关，这就证明了定理。 □

19.8.3 主动安全的身份识别协议

我们下面将展示如何使用见证独立性来设计主动安全的识别协议。该构造是相当通用的，其基本构件是一个 Sigma 协议和一个单向密钥生成算法。我们还利用了 19.7.2 小节中介绍的 OR 证明构造。

令 (P,V) 是 $\mathcal{R} \subseteq \mathcal{X}\times\mathcal{Y}$ 上的一个 Sigma 协议。我们假设 (P,V) 是特殊 HVZK 的，其挑战空间形如 $\mathcal{C} = \{0,1\}^n$。以上假设使得我们可以应用 19.7.2 小节中介绍的 OR 证明构造。在安全分析中，我们还需要假设 (P,V) 具备知识健全性。

正如我们在 19.6 节中所介绍的，为了使用 (P,V) 构建一个身份识别协议，我们还需要一个关系 $\mathcal{R}$ 上的单向密钥生成算法 G。识别方案 $\mathcal{I} := (G,P,V)$ 对*窃听*是安全的。然而不需要太多努力，也不需要做任何额外的假设，我们就可以建立一个能够抵抗*主动*攻击的身份识别协议（如 18.6 节的定义）。

首先，我们通过对 $(P_0,V_0) := (P,V)$ 和 $(P_1,V_1) := (P,V)$ 应用 OR 证明构造建立一个新的 Sigma 协议 (P',V')。令 $\mathcal{R}' := \mathcal{R}_{\text{OR}}$ 为对应于该 OR 构造的关系：$\mathcal{R}'$ 上的陈述形如 $Y = (y_0,y_1) \in \mathcal{Y}^2$，而 Y 的见证形如 $X = (b,x) \in \{0,1\} \times \mathcal{X}$，其中 $(x,y_b) \in \mathcal{R}$。对于一个见证 $X = (b,x)$，我们称 b 这一比特为其**类型 (type)**。

其次，我们为关系 $\mathcal{R}'$ 构造一个新的密钥生成算法 G'，它按以下方式运行：

$(y_0,(x_0,y_0)) \xleftarrow{\text{R}} G()$，$(y_1,(x_1,y_1)) \xleftarrow{\text{R}} G()$
$b \xleftarrow{\text{R}} \{0,1\}$
$Y \leftarrow (y_0,y_1)$
$X \leftarrow (b,x_b)$
输出 $(Y,(X,Y))$

G' 的一个关键属性是，作为随机变量，Y 和 b 是相互独立的。也就是说，当我们获取陈述 Y 时，我们无法推断 X 是 $(0,x_0)$ 还是 $(1,x_1)$。

我们下面证明识别协议 $\mathcal{I}' := (G',P',V')$ 对于主动攻击是安全的。

定理 19.22. *令 (P,V) 是有效关系 $\mathcal{R}$ 上的 Sigma 协议，有形如 $\{0,1\}^n$ 的大挑战空间。假设 (P,V) 是特殊 HVZK 的，且能提供知识健全性。进一步地，假设 $\mathcal{R}$ 上的密钥生成算法 G 是单向的。那么上面定义的识别方案 $\mathcal{I}' := (G',P',V')$ 对主动攻击是安全的。*

特别地，假设 $\mathcal{A}$ 是一个冒充对手，它使用攻击游戏 18.3 中的主动攻击方式来攻击 $\mathcal{I}'$，其优势为 $\epsilon := \mathsf{ID3adv}[\mathcal{A},\mathcal{I}']$。那么必然存在一个有效对手 $\mathcal{B}$，其运行时间大约是 $\mathcal{A}$ 的两倍，使得：

$$\mathrm{OWadv}[\mathcal{B}, G] \geq \frac{1}{2}\left(\epsilon^2 - \frac{\epsilon}{N}\right)$$

其中 $N := 2^n$。

证明. 我们首先回顾一下针对 (P', V') 的主动冒充攻击是怎么进行的。它包含三个阶段。

密钥生成阶段。挑战者运行密钥生成算法 G'，获得一个公钥 $pk' = Y$ 和一个私钥 $sk' = (X, Y)$，并将 pk' 发送给对手 $\mathcal{A}$。

主动探测阶段。对手 $\mathcal{A}$ 与验证者 $P'(sk')$ 交互。这里挑战者扮演证明者的角色，而对手 $\mathcal{A}$ 扮演可能作弊的验证者的角色。对手可能与多个证明者实例同时交互。

冒充尝试。与直接攻击一样，对手现在与验证者 $V'(pk')$ 进行交互，试图使其输出 accept。这里挑战者扮演验证者的角色，而对手 $\mathcal{A}$ 则扮演可能作弊的证明者。在该阶段，对手会提供一个承诺，挑战者将用一个随机的挑战来应答。如果对手对随机挑战的应答能够产生一个接受对话，它就赢得了游戏。

所以，令 ϵ 为 $\mathcal{A}$ 赢得该游戏的概率。

现在，我们描述我们想要构造的对手 $\mathcal{B}$，他被设计用来打破 G 的单向性假设。首先，$\mathcal{B}$ 的挑战者计算 $(y^*, (x^*, y^*)) \xleftarrow{\mathrm{R}} G()$，并将 y^* 发送给 $\mathcal{B}$。$\mathcal{B}$ 的目标是计算一个 y^* 的见证。

对手 $\mathcal{B}$ 首先扮演 $\mathcal{A}$ 的挑战者，在所有三个阶段中运行一次 $\mathcal{A}$。在密钥生成阶段，$\mathcal{B}$ 计算 $(pk', sk') = (Y, (X, Y))$，方法如下：

$b \xleftarrow{R} \{0, 1\}$
$(y, (x, y)) \xleftarrow{\mathrm{R}} G()$
如果 $b = 0$
　　则 $Y \leftarrow (y, y^*)$
　　否则 $Y \leftarrow (y^*, y)$
$X \leftarrow (b, x)$

注意 (pk', sk') 的分布正好与 G' 的输出分布相同。

在运行所有三个阶段后，$\mathcal{B}$ 将 $\mathcal{A}$ 回溯到第三阶段中挑战者（作为验证者）向 $\mathcal{A}$ 发送随机挑战的那一时刻，并向 $\mathcal{A}$ 发送一个新的随机挑战。如果这样做能够构造出两个不同挑战的接受对话，那么根据知识健全性，$\mathcal{B}$ 必然可以为陈述 Y 提取一个见证 $\widehat{X} = (\hat{b}, \hat{x})$。此外，如果 $\hat{b} \neq b$，那么 $\hat{x}$ 就是 y^* 的一个见证。

因此，剩下的工作就是分析 $\mathcal{B}$ 的成功概率了。现在，如果 $\mathcal{B}$ 从 $\mathcal{A}$ 中提取了一个见证 $\widehat{X}$，并且 $\widehat{X}$ 和 X 的类型不同，那么 $\mathcal{B}$ 就成功了。根据回溯引理（引理 19.2），我们知道 $\mathcal{B}$ 从 $\mathcal{A}$ 中提取某个见证 $\widehat{X}$ 的最小概率为 $\epsilon^2 - \epsilon/N$。此外，我们知道 Y 本身没有向 $\mathcal{A}$ 透露任何关于 X 的类型的信息，而见证独立性在本质上说，主动探测阶段没有向 $\mathcal{A}$ 透露更多关于 X 的类型的信息。因此，对于 $\mathcal{B}$ 所提取的任何**特定**见证，其类型与 X 匹配的概率恰好是 $1/2$，这意味着 $\mathcal{B}$ 的总体成功概率最小是 $1/2 \times (\epsilon^2 - \epsilon/N)$，恰如要求。

如果我们愿意，我们也可以直接使用见证独立性的定义（以式 19.25 的形式），使上述关于 $\mathcal{B}$ 的成功概率的论证更严格一些。为此，我们用 X，$\widehat{\mathsf{X}}$，Y 表示随机变量，用 X，$\widehat{X}$，Y 来表示这些随机变量的特定取值。如果 $\mathcal{B}$ 未能提取到一个见证，我们就定义 $\widehat{\mathsf{X}} := \perp$。记 σ 为 $\mathcal{B}$ 的成功概率，那么我们有：

$$\sigma = \Pr[(\widehat{\mathsf{X}}, \mathsf{Y}) \in \mathcal{R}' \ \wedge \ \mathrm{type}(\mathsf{X}) \neq \mathrm{type}(\widehat{\mathsf{X}})]$$

使用全概率公式，我们可以对所有 $(\widehat{X}, Y) \in \mathcal{R}'$ 求和：

$$\begin{aligned}
\sigma &= \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\text{type}(\mathsf{X}) \neq \text{type}(\widehat{\mathsf{X}}) \wedge \widehat{\mathsf{X}} = \widehat{X} \wedge \mathsf{Y} = Y] \\
&= \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\text{type}(\mathsf{X}) \neq \text{type}(\widehat{\mathsf{X}}) \mid \widehat{\mathsf{X}} = \widehat{X} \wedge \mathsf{Y} = Y] \cdot \Pr[\widehat{\mathsf{X}} = \widehat{X} \wedge \mathsf{Y} = Y] \\
&= \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\text{type}(\mathsf{X}) \neq \text{type}(\widehat{\mathsf{X}}) \mid \mathsf{Y} = Y] \cdot \Pr[\widehat{\mathsf{X}} = \widehat{X} \wedge \mathsf{Y} = Y] \quad \text{（见证独立性）} \\
&= \frac{1}{2} \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\widehat{\mathsf{X}} = \widehat{X} \wedge \mathsf{Y} = Y] \quad \text{（}\mathsf{Y}\text{ 与 type}(\mathsf{X})\text{ 相互独立）} \\
&= \frac{1}{2} \Pr[(\widehat{\mathsf{X}}, \mathsf{Y}) \in \mathcal{R}'] \geq \frac{1}{2}\left(\epsilon^2 - \frac{\epsilon}{N}\right)
\end{aligned}$$

□

具体的实例化. 上述构造立即就为我们提供了两个具体的识别协议，它们对于主动攻击都是安全的。一个来自 Schnorr 协议，其安全性基于离散对数假设；另一个来自 GQ 协议，其安全性基于 RSA 假设。这两个主动安全协议的成本（在计算和带宽方面）大约是其抗窃听的对应协议的两倍。

19.8.4 Okamoto 身份识别协议

我们刚刚看到了如何建立一个身份识别协议，其对主动攻击的安全性是基于离散对数假设的。现在，我们介绍一种更高效的方法，这种方法基于 Okamoto 协议。

首先让我们回顾一下 19.5.1 小节介绍的 Okamoto 协议 (P, V)。除了由 $g \in \mathbb{G}$ 生成的 q 阶循环群 $\mathbb{G}$，这个协议还利用了第二个群元素 $h \in \mathbb{G}$，我们将其看作是一个系统参数。该协议最自然的密钥生成算法 G 是计算 $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$，并输出 $pk = u$ 和 $sk = ((\alpha, \beta), u)$，其中 $u := g^\alpha h^\beta \in \mathbb{G}$。这就给我们提供了一个新的身份识别协议 $\mathcal{I}_O = (G, P, V)$，我们称之为 **Okamoto 身份识别协议**。利用见证独立性的概念，我们不难证明 $\mathcal{I}_O$ 对主动攻击是安全的。

定理 19.23. *令 $\mathcal{I}_O = (G, P, V)$ 是一个 Okamoto 身份识别协议。假设其挑战空间很大，同时假设系统参数 h 是从 $\mathbb{G}$ 中均匀随机选取的。那么，只要 $\mathbb{G}$ 上的离散对数假设成立，$\mathcal{I}_O$ 对主动攻击就是安全的。*

> *特别地，假设 $\mathcal{A}$ 是一个冒充攻击对手，他通过攻击游戏 18.3 中的主动攻击来攻击 $\mathcal{I}_O$，其优势为 $\epsilon := \text{ID3adv}[\mathcal{A}, \mathcal{I}_O]$。那么必然存在一个有效对手 $\mathcal{B}$，其运行时间大约是 $\mathcal{A}$ 的两倍，满足：*
>
> $$\text{DLadv}[\mathcal{B}, \mathbb{G}] \geq \left(1 - \frac{1}{q}\right)\left(\epsilon^2 - \frac{\epsilon}{N}\right)$$
>
> *其中 N 是挑战空间的大小。*

证明. 定理 19.23 的证明与定理 19.22 的证明结构基本相同。

假设 $\mathcal{A}$ 在攻击游戏 18.3 中攻击 $\mathcal{I}_O$ 时的优势为 ϵ。我们的离散对数对手 $\mathcal{B}$ 从他的挑战者处收到一个随机的群元素 $h \in \mathbb{G}$。$\mathcal{B}$ 的目标是利用 $\mathcal{A}$ 计算 $\text{Dlog}_g h$。

对手 $\mathcal{B}$ 首先会扮演 $\mathcal{A}$ 的挑战者的角色，在攻击游戏 18.3 中的所有三个阶段运行一次 $\mathcal{A}$。我们的对手 $\mathcal{B}$ 使用群元素 h 作为 Okamoto 协议的系统参数，但在其他方面遵循攻击游戏 18.3 中挑战者的逻辑，没有其他修改。其运行流程如下：

密钥生成阶段。 $\mathcal{B}$ 计算 $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$，$u \leftarrow g^\alpha h^\beta$，并将公钥 $pk := u$ 发送给 $\mathcal{A}$，自己保留私钥 $sk := ((\alpha, \beta), u)$。

主动探测阶段。 $\mathcal{A}$ 与证明者 $P(sk)$ 的若干实例进行交互（可能是同时进行的）。这些证明者的角色由 $\mathcal{B}$ 来扮演。

冒充尝试。 $\mathcal{A}$ 试图让验证者 $V(pk)$ 输出 accept。验证者的角色由 $\mathcal{B}$ 扮演。

在运行所有三个阶段后，$\mathcal{B}$ 将 $\mathcal{A}$ 回溯到第三阶段中验证者向 $\mathcal{A}$ 发送随机挑战的那一时刻，并向 $\mathcal{A}$ 发送一个新的随机挑战。如果这样做能够构造出两个不同挑战的接受对话，那么根据知识健全性，$\mathcal{B}$ 必然可以为陈述 u 提取一个见证 $(\hat{\alpha}, \hat{\beta})$。此外，如果 $(\alpha, \beta) \neq (\hat{\alpha}, \hat{\beta})$，我们就拥有 u 的两个不同的表示，这样 $\mathcal{B}$ 就能采用事实 10.3 中的方法计算 $\mathsf{Dlog}_g h$。

如果我们的对手 $\mathcal{B}$ 能从 $\mathcal{A}$ 中提取一个与 (α, β) 不同的见证，那么它就成功了。根据回溯引理（引理 19.2），我们知道 $\mathcal{B}$ 从 $\mathcal{A}$ 中提取出某个见证的最小概率为 $\epsilon^2 - \epsilon/N$。此外，我们知道 u 本身并没有揭示 $\mathcal{B}$ 使用的是 u 的 q 个可能见证中的哪一个，而见证独立性表明，主动探测阶段没有向 $\mathcal{A}$ 透露更多关于这个见证的信息。因此，对于 $\mathcal{B}$ 从 $\mathcal{A}$ 那里提取的任何特定见证，它恰好等于 (α, β) 的概率是 $1/q$。这意味着 $\mathcal{B}$ 的总体成功概率最小是 $(1 - 1/q) \times (\epsilon^2 - \epsilon/N)$，恰如要求。 □

19.9 一个有趣的应用：一种两轮见证独立协议

待写。

19.10 笔记

要添加的文献引用。

19.11 练习

第二十章　零知识的属性证明

在上一章中，我们介绍了如何使用 Sigma 协议来构建身份识别和签名方案。在这些应用中，我们把 Sigma 协议作为 “知识证明”：利用回溯和知识健全性，我们可以有效地从任何有说服力的证明者那里提取出一个见证。

在本章中，我们将介绍如何使用 Sigma 协议来证明某些事实是真实的（不需要披露太多其他信息）。在以这种方式使用 Sigma 协议的应用中，安全性取决于所谓事实的*真实性*，而不是任何*知识*的概念。例如，Chaum-Pedersen 协议（见 19.5.2 小节）允许证明者说服验证者相信一个给定的群元素构成的三元组是一个 DH 三元组。这种能力本身就是构建和分析有趣的加密协议的一个有用工具。

在 20.1 节中，我们首先会定义与有效关系相关的*真实陈述的语言*：即存在相应见证的陈述的集合。然后，我们会为 Sigma 协议定义一个*存在健全性*的概念，它意味着任何证明者想要使验证者接受一个不真实的陈述（即不存在对应的见证）是不可行的。这个概念与*知识健全性*不同，因为我们不需要任何种类的见证提取器。然而，我们将会看到，知识健全性本身就意味着存在健全性。

在 20.2 节中，我们将提出一系列的例子来说明存在健全性。这些例子围绕着在加密数据上证明属性的想法而展开。

在 20.3 节中，然后我们会展示如何使用 Fiat-Shamir 变换（见 19.6.1 小节）的变体将 Sigma 协议变成非交互式证明。

在更后面的章节中，我们将研究构建证明系统的更高级的技术。

20.1　语言与存在健全性

让我们从一个定义开始。

定义 20.1 (真实陈述的语言). *令 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 是一个有效关系。如果对于某个 $x \in \mathcal{X}$ 有 $(x, y) \in \mathcal{R}$，那么我们就称陈述 $y \in \mathcal{Y}$ 是一个**真实陈述 (true statement)**，否则称其为**虚假陈述 (false statement)**。我们称 $L_{\mathcal{R}}$ 为**由 $\mathcal{R}$ 定义的语言 (language defined by $\mathcal{R}$)**，即定义在 $\mathcal{R}$ 上的所有真实陈述，也就是说，$L_{\mathcal{R}} := \{y \in \mathcal{Y} : (x, y) \in \mathcal{R} \text{ for some } x \in \mathcal{X}\}$。*

上述定义中的 “语言 (language)” 一词来自复杂性理论。在本章中，我们将研究一些有趣的关系 $\mathcal{R}$ 以及由它们所定义的语言集合 $L_{\mathcal{R}}$。举一个上一章的例子，回顾一下，Chaum-Pedersen 协议是一个定义在关系

$$\mathcal{R} := \left\{ \big(\beta, (u, v, w)\big) \in \mathbb{Z}_q \times \mathbb{G}^3 : v = g^\beta, w = u^\beta \right\}$$

上的 Sigma 协议。那么，由 $\mathcal{R}$ 定义的语言 $L_{\mathcal{R}}$ 就是所有 DH 三元组 $(u, v, w) \in \mathbb{G}^3$ 的集合。

于是，我们就可以用下面的攻击游戏来定义存在健全性的概念：

攻击游戏 20.1 (存在健全性). 令 $\Pi = (P, V)$ 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个 Sigma 协议。对于一个给定对手 $\mathcal{A}$，攻击游戏按照如下方式运行：

- 对手 $\mathcal{A}$ 选择一个陈述 $y \in \mathcal{Y}$，并将其发送给挑战者。
- 对手 $\mathcal{A}$ 与验证者 $V(y)$ 进行交互，其中挑战者扮演验证者的角色，对手 $\mathcal{A}$ 扮演 "作弊的" 证明者的角色。

如果 $V(y)$ 在交互结束时输出 accept 但 $y \notin L_{\mathcal{R}}$，我们就称对手 $\mathcal{A}$ 赢得了该游戏。我们定义 $\mathsf{ESadv}[\mathcal{A}, \Pi]$ 为对手 $\mathcal{A}$ 对于 Π 的优势，其值为对手 $\mathcal{A}$ 赢得该游戏的概率。

定义 20.2. 如果对于所有的有效对手 $\mathcal{A}$，$\mathsf{ESadv}[\mathcal{A}, \Pi]$ 的值都可以忽略不计，我们就称 *Sigma* 协议 Π 是***存在健全 (existentially sound)*** 的。

定理 20.1. 令 Π 是一个有大挑战空间的 *Sigma* 协议，如果 Π 提供知识健全性，那么 Π 一定是存在健全的。

特别地，对于所有对手 $\mathcal{A}$，我们都有：

$$\mathsf{ESadv}[\mathcal{A}, \Pi] \leq \frac{1}{N} \tag{20.1}$$

其中 N 是挑战空间的大小。

证明. 我们只需证明，如果 $\mathcal{A}$ 选择了一个虚假陈述 y 和一个承诺 t，那么最多只能有一个挑战 c 使得存在一个应答 z 能够产生一个 y 的接受对话 (t, c, z)。观察到，如果有两个这样的挑战，那么 y 将有两个接受对话 (t, c, z) 和 (t, c', z')，其中 $c \neq c'$，而知识健全性意味着仅存在一个 y 的见证，这与事实相矛盾。 □

我们指出，对于任意强大的对手，上述定理都无条件成立。我们在下一节将这些想法付诸实施。

20.2 证明加密数据的属性

在许多应用中，以下情况都会出现。Alice 用 Bob 的公钥加密一个消息 m，得到一个密文 c。此外，Alice 想向第三方，比如 Charlie（他可以看到 c，但看不到 m），证明加密后的明文 m 满足某个属性，但不向 Charlie 透露任何关于 m 的其他信息。

一个存在健全的、特殊 HVZK 的 Sigma 协议可以用来解决这类问题。然而，这样的协议并不是一个完整的解决方案。一个问题是，只有 Charlie 诚实地遵循验证协议时，HVZK 属性才能保证不泄露关于 m 的信息。解决这个问题的一个方法就是使用我们在 19.6.1 小节介绍想法，将交互式识别协议变成签名。也就是说，我们不使用实际的验证者来生成随机挑战，而是使用哈希函数来生成挑战。我们将在下一节中详细介绍这种方法。现在，让我们先来看几个有趣而重要的例子，看看我们如何使用 Sigma 协议来证明加密数据的特定属性。

在我们的例子中，简便起见，我们使用 ElGamal 加密方案的乘法变体，我们曾在练习 11.5 中讨论过。该方案利用由 $g \in \mathbb{G}$ 生成的素阶 q 的循环群 $\mathbb{G}$，私钥是随机选择的 $\alpha \in \mathbb{Z}_q$，公钥是 $u := g^\alpha \in \mathbb{G}$。明文 m 的加密是 $(v, e) \in \mathbb{G}^2$，其中 $v := g^\beta$，$e := u^\beta \cdot m$，并且 β 也是从 $\mathbb{Z}_q$ 中随机选取的。要用私钥 α 解密 (v, e)，我们需要计算 $m := e/v^\alpha$。正如我们曾在练习 11.5 中要求的那样，可以证明当 $\mathbb{G}$ 满足 DDH 假设时，上述方案是语义安全的。

例 20.1 (明文相等). 假设 Alice 用 Bob 的公钥 u_0 加密了消息 m，得到密文 (v_0, e_0)。然后 Alice 又用 Bill 的公钥 u_1 加密了同一个消息 m，得到了密文 (v_1, e_1)。她想让 Charlie 相信两个密文对应的明文相同，但又不透露任何关于明文的信息。比如说，一些协议可能要求 Alice 向 Bob 和 Bill 广播相同的消息。针对这种场景的协议要求在保持消息加密的同时能够证明 Alice 所加密的明文消息确实是相同的。

因此，所以我们希望有一个关系：

$$\mathcal{R} := \left\{ \Big((\beta_0, \beta_1, m), (u_0, v_0, e_0, u_1, v_1, e_1) \Big) : v_0 = g^{\beta_0},\ e_0 = u_0^{\beta_0} \cdot m,\ v_1 = g^{\beta_1},\ e_1 = u_1^{\beta_1} \cdot m \right\}$$

上的 Sigma 协议。语言 $L_{\mathcal{R}}$ 是元组 $(u_0, v_0, e_0, u_1, v_1, e_1)$，使得 (v_0, e_0) 和 (v_1, e_1) 对应在公钥 u_0 和 u_1 下加密的同一消息 m。

为了设计一个关系 $\mathcal{R}$ 上的有效 Sigma 协议，我们注意到 $(u_0, v_0, e_0, u_1, v_1, e_1) \in L_{\mathcal{R}}$ 就等价于：

$$v_0 = g^{\beta_0},\ v_1 = g^{\beta_1},\ e_0/e_1 = u_0^{\beta_0} u_1^{-\beta_1},\ \text{ for some } \beta_0, \beta_1 \in \mathbb{Z}_q$$

基于这一观察，我们可以使用 19.5.3 小节中介绍的通用线性协议实现一个关系 $\mathcal{R}$ 上的 Sigma 协议。具体来说，Alice 需要向 Charlie 证明存在 β_0 和 β_1 使得方程组：

$$v_0 = g^{0},\ v_1 = g^{\beta_1},\ e_0/e_1 = u_0^{\beta_0} u_1^{-\beta_1}$$

成立。这样就得到了一个关系 $\mathcal{R}$ 上的存在健全的、特殊 HVZK 的 Sigma 协议。

请注意，尽管 Alice 在上述协议中没有显式地使用消息 m，但 Alice 还是需要知道它，因为她需要同时知道 β_0 和 β_1，其中任何一个都决定了 m。

例 20.2 (明文相等 2). 考虑上个例子的一个变体。现在 Alice 有两个密文 (v_0, e_0) 和 (v_1, e_1)，这两个密文均来自于 Bob 的公钥 u 所加密的相同的明文。不同的地方在于，现在两个密文均对应着使用**相同**公钥加密的相同明文。同样的，Alice 想要向 Charlie 证明这一事实，但又不向他透露其他信息。注意到，如果 (v_0, e_0) 和 (v_1, e_1) 来自相同的明文，那么：

$$v_0 = g^{\beta_0},\ e_0 = u^{\beta_0} \cdot m,\ v_1 = g^{\beta_1},\ e_1 = u^{\beta_1} \cdot m$$

其中 $\beta_0, \beta_1 \in \mathbb{Z}_q$，$m \in \mathbb{G}$。用第一式除以第三式，第二式除以第四式，我们可以得到：

$$v_0/v_1 = g^{\beta},\ \ e_0/e_1 = u^{\beta} \tag{20.2}$$

其中 $\beta := \beta_0 - \beta_1$。此外，不难看出，如果式 20.2 对于某个 $\beta \in \mathbb{Z}_q$ 成立，则 (v_0, e_0) 和 (v_1, e_1) 必然来自对相同明文的加密。

因此，Alice 需要做的就是让 Charlie 相信存在满足式 20.2 的 β。为此，她可以使用 19.5.3 小节介绍的通用线性协议。在本例的场景下，它其实就是证明 $(u, v_0/v_1, e_0/e_1)$ 是一个 DH 三元组的 Chaum-Pedersen 协议（见 19.5.2 小节）。

需要注意的是，为了证明 (v_0, e_0) 和 (v_1, e_1) 加密了相同的消息，Alice 只需要知道满足上式的 β，她并不需要知道明文消息 m 本身。特别地，Alice 不需要是产生这些密文的一方。事实上，她可以从其他方收到密文 (v_0, e_0)，然后通过计算 $v_1 := v_0 \cdot g^{\beta}$ 和 $e_1 := e_0 \cdot u^{\beta}$ 为她选择的 β 创建同一消息 m 的新密文 (v_1, e_1)。一些匿名服务就能执行类似的功能，他们能够利用这个协议对一个加密消息进行重加密。这个协议可以用来确保这一过程被正确完成。

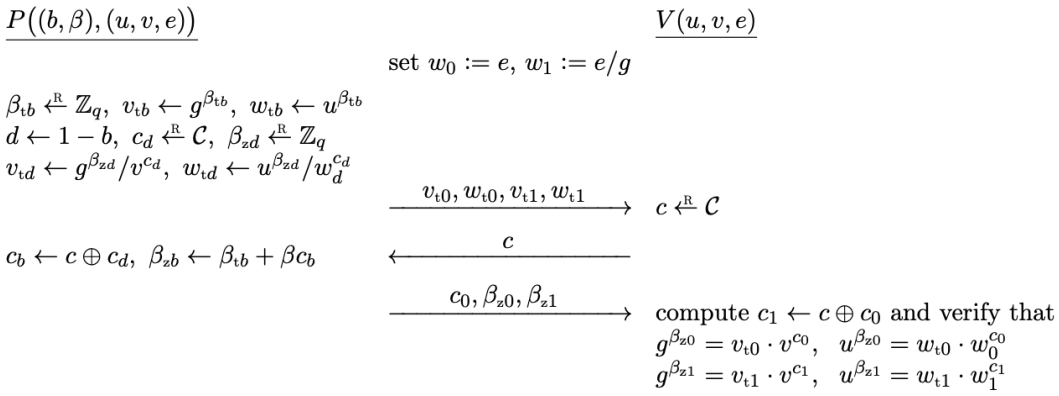


图 20.1: 用于加密比特的 Sigma 协议

例 20.3 (加密后的比特). 为了加密一个比特 $b \in \{0,1\}$，比较方便的做法是将 b 编码为群元素 $g^b \in \mathbb{G}$，然后用乘性 ElGamal 对 g^b 进行加密。因此，假设 Alice 以这种方式用 Bob 的公钥 u 对比特 b 进行加密，产生一个密文 $(v,e) = (g^\beta, u^\beta \cdot g^b)$。她想让 Charlie 相信 (v,e) 确实是由 Bob 的公钥加密一个比特得到的（而不是加密其他的什么东西，比如 g^{17}），而又不向 Charlie 透露任何其他信息。

因此，我们希望有一个关系：

$$\mathcal{R} := \left\{ \Big((b,\beta),(u,v,e)\Big) : v = g^\beta,\ e = u^\beta \cdot g^b,\ b \in \{0,1\} \right\}$$

上的 Sigma 协议。与此关系相对应的语言 $L_{\mathcal{R}}$ 是一个元组 (u,v,e)，满足 (v,e) 是用公钥 u 加密一个比特得到的。

我们的 $\mathcal{R}$ 上的 Sigma 协议基于下面的观察，即 $(u,v,e) \in L_{\mathcal{R}}$ 等价于：

$$\text{either } (u,v,e) \text{ or } (u,v,e/g) \text{ is a DH-triple}$$

19.5.2 中介绍的 Chaum-Pedersen 协议允许一方证明一个给定的三元组是 DH 三元组。我们把它与 19.7.2 中介绍的 OR 证明构造结合起来。这为我们提供了一个关系：

$$\mathcal{R}' := \left\{ \Big((b,\beta),\ ((u_0,v_0,w_0),\ (u_1,v_1,w_1))\Big) : v_b = g^\beta,\ w_b = u_b^\beta \right\}$$

上的 Sigma 协议。如果 $(u_0,v_0,w_0),(u_1,v_1,w_1)$ 中至少有一个是 DH 三元组，则语句 $((u_0,v_0,w_0),(u_1,v_1,w_1))$ 就在 $L_{\mathcal{R}'}$ 中。于是，我们有：

$$(u,v,e) \in L_{\mathcal{R}} \iff \Big((u,v,e),(u,v,e/g)\Big) \in L_{\mathcal{R}'}$$

因此，为了让 Alice 向 Charlie 证明 $(u,v,e) \in L_{\mathcal{R}}$，可以使用陈述 $((u,v,e),(u,v,e/g))$ 和见证 (b,β) 运行一个 $\mathcal{R}'$ 上的 Sigma 协议。完整起见，我们在图 20.1 中展示了完整的 $\mathcal{R}$ 上的 Sigma 协议。在证明者逻辑的第一行，证明者为其知道的见证启动证明过程，第二和第三行为其不知道的见证运行 HVZK 模拟器。由此产生的 $\mathcal{R}$ 上的 Sigma 协议是存在健全的，并且是特殊 HVZK 的。

正如练习 20.6 将要说明的，该协议可以泛化为证明一个密文 (v,e) 在 $B > 2$ 的情况下加密了一个值 $0 \le b < B$。该协议的交互记录随 B 的增大线性增长，因此只适用于相对较小的 B。我们会在 20.4.1 小节介绍如何处理 B 较大的情况。

例 20.4 (加密后的 DH 三元组). 假设 Alice 有一个 DH 三元组 $(g^{\gamma_1}, g^{\gamma_2}, g^{\gamma_3})$，其中 $\gamma_3 = \gamma_1\gamma_2$。她使用 Bob 的公钥 u 对每个元素进行加密，生成了三条密文 (v_1, e_1)，(v_2, e_2)，(v_3, e_3)，其中：

$$v_i = g^{\beta_i},\ e_i = u^{\beta_i} g^{\gamma_i}\ \ (i = 1, 2, 3) \tag{20.3}$$

她把这些密文交给 Charlie，并想让他相信这些密文确实是对一个 DH 三元组的加密，但又不透露任何其他信息。

因此，我们希望有一个关系：

$$\mathcal{R} := \left\{ \Big((\beta_1, \beta_2, \beta_3, \gamma_1, \gamma_2, \gamma_3), (u, v_1, e_1, v_2, e_2, v_3, e_3) \Big) : v_i = g^{\beta_i},\ e_i = u^{\beta_i} g^{\gamma_i}\ (i = 1, 2, 3),\ \gamma_3 = \gamma_1\gamma 2 \right\}$$

上的 Sigma 协议。相应的语言 $L_{\mathcal{R}}$ 是满足密文 (v_1, e_1)，(v_2, e_2)，(v_3, e_3) 是由公钥 u 加密一个 DH 三元组得到的元组 $(u,\ v_1, e_1,\ v_2, e_2,\ v_3, e_3)$。

虽然由于条件 $\gamma_3 = \gamma_1\gamma_2$，关系 $\mathcal{R}$ 本质上是非线性的，但我们还是可以使用 19.5.3 中介绍的通用线性协议为 $\mathcal{R}$ 设计一个 Sigma 协议。其基本思想是，Alice 向 Charlie 证明，存在 β_1、β_3、γ_1 和 τ 满足方程组：

$$v_1 = g^{\beta_1},\ \ e_1 = u^{\beta_1} g^{\gamma_1},\ \ v_3 = g^{\beta_3},\ \ v_2^{\gamma_1} = g^{\tau},\ \ e^{\gamma_1} u^{\beta_3} = e_3 u^{\tau} \tag{20.4}$$

为了证明这一点，我们声称 $(u, v_1, e_1, v_2, e_2, v_3, e_3) \in L_{\mathcal{R}}$ 成立当且仅当存在能够满足式 20.4 的 $\beta_1, \beta_3, \gamma_1, \tau$。注意到，密文 (v_1, e_1)，(v_2, e_2)，(v_3, e_3) 唯一决定了 β_i 和满足式 20.3 的 γ_i。β_1，β_3 和 γ_1 也是满足式 20.4 中前三个等式的唯一值。式 20.4 中的第四个方程可以通过令 $\tau := \gamma_1\beta_2$ 而得到唯一的满足。因此，剩下的工作就是考虑式 20.4 中最后一个方程，其等号左边为：

$$e^{\gamma_1} u^{\beta_3} = (u^{\beta_2} g^{\gamma_2})^{\gamma_1} u^{\beta_3} = u^{\beta_3 + \tau} g^{\gamma_1\gamma_2}$$

而等号右边为：

$$e_3 u^{\tau} = (u^{\beta_3} g^{\gamma_3}) u^{\tau} = u^{\beta_3 + \tau} g^{\gamma_3}$$

因此，当且仅当 $\gamma_1\gamma_2 = \gamma_3$ 时，该方程成立。这就证明了我们的声称。

这样，我们就得到了 $\mathcal{R}$ 上的 Sigma 协议。为了运行该协议，Alice 使用见证 $(\beta_1, \beta_3, \gamma_1, \tau := \gamma_1\beta_2)$ 运行式 20.4 的通用线性协议。该协议的正确性、存在健全性和特殊 HVZK 性都来自于通用线性协议的相应属性。

例 20.5 (加密后的比特 2). 利用上一个例子的思路，我们可以得到用于例 20.3 中加密比特问题的另一个 Sigma 协议。

如果 Alice 想向 Charlie 证明密文 (v, e) 形如 $v = g^{\beta}$，$e = u^{\beta} g^{b}$，其中 $b = \{0, 1\}$，她只需要说明 $b^2 = b$，因为 $b \in \mathbb{Z}_q$ 中满足 $b^2 = b$ 的就只有 $b = 0$ 或者 $b = 1$。

因此，使用通用线性协议，Alice 只要向 Charlie 证明存在 $b, \beta, \tau(= \beta b)$ 满足方程组：

$$v = g^{\beta},\ \ e = u^{\beta} g^{b},\ \ v^{b} = g^{\tau},\ \ e^{b} = u^{\tau} g^{b}$$

读者可以自行验证这是否能够产生针对例 20.3 中关系 $\mathcal{R}$ 的一个存在健全的、特殊 HVZK 的 Sigma 协议。所得到的协议与例 20.3 中的加密比特协议具有类似的性能表现。

这个协议可以推广到向 Charlie 证明，一个密文 (v, e) 来自对值 $0 \le b < B$ 的加密，其中 $B > 2$。这种推广使用到了下一个例子中将要介绍的 Sigma 协议，用于向 Charlie 证明 b 满足多项式关系 $b(b-1)(b-2)\cdots(b-(B-1)) = 0$。该关系意味着 $0 \le b < B$。该协议的交互记录同样随 B 的增大而线性增长，因此也只适用于相对较小的 B。

例 20.6 (多项式关系). 我们可以进一步扩展例 20.4 的想法。假设 Alice 用 Bob 的公钥 u 生成两个密文 (v,e) 和 (v',e')，第一个密文来自对群元素 g^γ 的加密，第二个则来自 $g^{\gamma'}$。Alice 想让 Charlie 相信，对于某个特定的多项式 $f(x)=\sum_{i=0}^{d}\lambda_i x^i$，有 $\gamma'=f(\gamma)$。我们将假设多项式 $f(x)$ 的阶 d 和系数 $\lambda_0,\ldots,\lambda_d$ 都是固定且公开的值（即为常数或者系统参数）。

因此，我们想要一个关系：

$$\mathcal{R}=\left\{\Big((\beta,\gamma,\beta',\gamma'),(u,v,e,v',e')\Big):v=g^\beta,\ e=u^\beta\cdot g^\gamma,\ v'=g^{\beta'},\ e'=u^{\beta'}\cdot g^{\gamma'},\ \gamma'=f(\gamma)\right\}$$

上的 Sigma 协议。

为了得到 $\mathcal{R}$ 上的 Sigma 协议，Alice 和 Charlie 使用通用线性协议，其中 Alice 向 Charlie 证明，存在：

$$\beta,\quad \gamma_1,\ldots,\gamma_d,\quad \tau_1,\ldots,\tau_{d-1},\quad \beta',\ \gamma'$$

满足方程组：

$$\begin{aligned}&v=g^\beta,\ e=u^\beta g^{\gamma_1},\ v'=g^{\beta'},\ e'=u^{\beta'}g^{\gamma'},\ \gamma'=\lambda_0+\lambda_1\gamma_1+\cdots+\lambda_d\gamma_d,\\&v^{\gamma_i}=g^{\tau_i},\ e^{\gamma_i}=u^{\tau_i}g^{\gamma_{i+1}}\quad(i=1,\ldots,d-1)\end{aligned}$$

请注意，这里我们使用的是通用线性协议的推广版本，它能够处理 $\mathbb{G}$ 和 $\mathbb{Z}_q$ 上的方程（见定理 19.11 之后的讨论）。Alice 使用 $\gamma_i:=\gamma^i,\ \ i=1,\ldots,d$ 和 $\tau_i:=\beta\gamma^i,\ \ i=1,\ldots,d-1$ 运行该协议。读者可以证明，这些实际上是满足这个方程组的唯一取值。通过一个简单的归纳论证可以很容易地证明这一点。由此可见，所得到的 Sigma 协议是一个关系 $\mathcal{R}$ 上存在健全的，特殊 HVZK 的 Sigma 协议。

以上的例子展示了*语言规约*的概念。一般来说，这种从 $\mathcal{R}\subseteq\mathcal{X}\times\mathcal{Y}$ 到 $\mathcal{R}'\subseteq\mathcal{X}'\times\mathcal{Y}'$ 的规约是一对可有效计算的映射 $f:\mathcal{X}\times\mathcal{Y}\to\mathcal{X}'$ 和 $g:\mathcal{Y}\to\mathcal{Y}'$，使得：

(i) 对于所有 $(x,y)\in\mathcal{R}$ 都有 $(f(x,y),g(y))\in\mathcal{R}'$ 成立，并且

(ii) 对于所有 $y\in\mathcal{Y}$，$g(y)\in L_{\mathcal{R}'}\implies y\in L_{\mathcal{R}}$。

利用这样的规约，我们就可以利用 $\mathcal{R}'$ 上的 Sigma 协议 Π' 来构造一个 $\mathcal{R}$ 上的 Sigma 协议 Π。上面的第一个条件能够确保 Π 能够从 Π' 处继承正确性和特殊 HVZK 性，第二个条件确保 Π 能够从 Π' 处继承存在健全性。注意，知识可靠性不一定需要继承。也就是说，我们不要求可以从 $g(y)$ 的见证中恢复 y 的见证。在上述几乎所有的例子中，关系 $\mathcal{R}'$ 都是通用线性关系的特例。唯一的例外是例 20.3，其关系 $\mathcal{R}'$ 来自 OR 证明构造。

20.2.1 一种用于非线性关系的通用协议

在上面的几个例子中，我们可以看到，通用线性协议可以用于证明某些非线性关系。我们下面将展示更加通用的情况。正如我们将要看到的，例 20.6 中的多项式求值协议可以很容易地推导出这种构造的一个特例。同样的通用构造也可以用来推导出针对例 20.4 和例 20.4 中的问题的协议；然而，所得到的协议不会像这两个例子中的协议那样高效。

与往常一样，令 $\mathbb{G}$ 是一个由 $g\in\mathbb{G}$ 生成的素阶 q 的循环群。考虑 19.5.3 小节中介绍的通用线性协议，该协议适用于形如式 19.13 所描述的公式 ϕ。假设我们现在也允许 ϕ 中存在形如 $x_i=x_j\cdot x_k$ 的非

线性方程。为了这个构造仍然有效，我们要求对于每个这样的非线性方程，ϕ 中还需要包含形如下面这样的两个辅助方程：

$$v = g^{x_\ell}, \quad e = u^{x_\ell} g^{x_j} \tag{20.5}$$

其中 u，v 和 e 都是群 $\mathbb{G}$ 中的元素，而 x_ℓ 是某个变量。为了简单起见，我们假设在对 ϕ 的描述中，存在从每个非线性方程到对应辅助方程的指针。

我们可以将这样的公式 ϕ 转化为可以使用通用线性协议处理的方程 ϕ'，方法如下。对于 ϕ 中的每个非线性方程 $x_i = x_j \cdot x_k$ 以及式 20.5 中相应的辅助方程，我们引入一个临时变量 t，并将非线性方程 $x_i = x_j \cdot x_k$ 替换为下面的一对方程：

$$v^{x_k} = g^t, \quad e^{x_t} = u^t h^{x_i} \tag{20.6}$$

这种变换的结果就是一个可以用通用线性协议处理的方程 ϕ'。用于 ϕ 的 Sigma 协议的工作原理如下。证明者和验证者都可以将 ϕ 转化为 ϕ'。假设证明者有变量 $(x_1, \ldots, x_n)$ 的一个赋值 $(\alpha_1, \ldots, \alpha_n)$，该赋值能够使得 ϕ 的值为 true。那么对于 ϕ 中的每个非线性方程 $x_i = x_j \cdot x_k$，证明者为式 20.6 中的临时变量 t 赋值 $\alpha_k \alpha_\ell$，然后用这个扩展赋值与验证者一起运行 ϕ' 的通用线性协议。

读者可以验证，这种变换得到的 Sigma 协议是特殊 HVZK 的，并且为关系 19.14 提供知识健全性，其中方程 ϕ 现在被允许具有上述的非线性形式。

多项式计算. 例 20.6 中的协议可以用这种转换方式导出。Alice 向 Charlie 证明，存在：

$$\beta, \quad \gamma_1, \ldots, \gamma_d, \quad \beta', \ \gamma'$$

满足方程组：

$$v = g^\beta, \quad e = u^\beta g^{\gamma_1}, \quad v' = g^{\beta'}, \quad e' = u^{\beta'} g^{\gamma'}, \quad \gamma' = \lambda_0 + \lambda_1 \gamma_1 + \cdots + \lambda_d \gamma_d,$$
$$\gamma_{i+1} = \gamma_1 \cdot \gamma_i \quad (i = 1, \ldots, d-1)$$

读者可以自行验证，从非线性到线性的变换能够将每个方程 $\gamma_{i+1} = \gamma_1 \cdot \gamma_i$ 转换为一对方程 $v^{\gamma_i} = g^{\tau_i}$ 和 $e^{\gamma_i} = u^{\tau_i} g^{\gamma_{i+1}}$。

加密后的比特. 例 20.5 中的协议也可以用这种变换得出。Alice 向 Charlie 证明存在 b 和 β 使得：

$$v = g^\beta, \ e = u^\beta g^b, \ b = b \cdot b$$

成立。读者可以自行证明，从非线性到线性的变换能够得出例 20.5 中的协议。

加密后的 DH 三元组.

我们也可以尝试使用这种技术来设计例 20.4 中的协议。最明显的方法是让 Alice 向 Charlie 证明，存在：

$$\beta_1, \beta_2, \beta_3, \quad \gamma_1, \gamma_2, \gamma_3$$

使得：

$$v_i = g^{\beta_i}, \ e_i = u^{\beta_i} g^{\gamma_i} \quad (i = 1, 2, 3), \ \gamma_3 = \gamma_1 \gamma_2$$

成立。我们可以将这个方程组插入上述从非线性到线性的变换中。这样做是可行的，但得到的协议不会像例 20.4 中的协议那样高效。

去除对非线性方程的约束. 虽然我们的通用变换相当有用，但它仍然受到一定的限制。事实上，我们要求对于每个非线性方程 $x_i = x_j \cdot x_k$，方程组还必须包括描述使用乘性 ElGamal 加密 x_j 或 x_k 的方

程。稍后，在 20.4.3 小节中，我们将看到，如果我们愿意使用相对较弱（但仍然有用）的 HVZK 形式（或较弱的知识健全性形式，见练习 20.5），我们就可以去掉这一要求。

20.3 非交互式证明系统

在上一节中，我们介绍了存在健全的 Sigma 协议的概念。在这一节中，我们将展示如何使用 Fiat-Shamir 变换（见 19.6.1 小节）将任意 Sigma 协议转换为一个**非交互式证明系统**。

基本思想非常简单：我们不依靠验证者来生成随机挑战，而是使用哈希函数 H 从陈述和承诺中推导出挑战。如果将 H 建模为一个随机预言机，那么我们可以证明以下事实：

(i) 如果 Sigma 协议是存在健全的，那么导出的非交互式证明系统也是存在健全的；

(ii) 如果 Sigma 协议是特殊 HVZK 的，那么运行对应的非交互式证明系统不会透露关于证明者的见证的任何有用信息。

第一个属性是对存在健全性概念在非交互式环境中的一个相当直接的应用。第二个属性是一个新型的 “零知识” 属性，定义起来可能有点棘手。

20.3.1 例子：一个投票协议

在给出正式定义之前，我们先通过展示一个投票协议来说明非交互式协议的功能。对投票协议进行正确建模需要相当大的努力，光是全面考虑安全要求就相当具有挑战性，我们不会在这里尝试这样做。相反，我们只会简要阐述基本的想法，并对其他的问题给出简单的提示。

现在假设我们有 n 个投票者，每个投票者都可以投 0 或 1 的票。在投票结束时，所有各方都应该知道票数的总和。当然，每个投票者都可以直接公布他们的投票内容。但是这并不是一个很好的解决方案，因为我们希望允许投票者保有他们的隐私。为此，一些投票协议会使用加密方案，使每个投票者公布其投票的加密信息。

一个方便的方案是 ElGamal 方案的乘性变体，我们已经在 20.2 节讨论过部分内容。同样，我们现在有一个由 $g \in \mathbb{G}$ 生成的素阶 q 的循环群 $\mathbb{G}$。私钥 $\alpha \in \mathbb{Z}_q$，公钥 $u := g^\alpha \in \mathbb{G}$。对消息 $m \in \mathbb{G}$ 的加密能得到 (v, e)，其中 $v := g^\beta$，$e := u^\beta \cdot m$。

我们下面介绍的设计是一个简单的尝试，它能够在一定程度上保护投票者的隐私。假设我们有一个可信服务器，称为**投票统计中心**（VTC），它能够运行密钥生成算法并获得一个公钥 $pk = u$ 和一个私钥 $sk = \alpha$。它向所有投票者公布 pk，然后自己保留 sk。

投票阶段。在投票阶段，第 i 个投票者将他的投票 $b_i \in \{0, 1\}$ 加密，即将 b_i 编码为群元素 $g^{b_i} \in \mathbb{G}$，然后用 VTC 的公钥对 g^{b_i} 进行加密，得到密文 (v_i, e_i)。注意 $v_i = g^{\beta_i}$，$e_i = u^{\beta_i} \cdot g^{b_i}$，其中 $\beta_i \in \mathbb{Z}_q$ 是随机选出的。所有密文都会被公开。

计票阶段。VTC 将所有公布的密文聚合为一个单一的密文 (v_*, e_*)，其中：

$$v_* := \prod_{i=1}^{n} v_i, \quad e_* := \prod_{i=1}^{n} e_i$$

令 $\beta_* := \sum_i \beta_i$，$\sigma := \sum_i b_i$，则有：

$$v_* = g^{\beta_*}, \quad e_* = u^{\beta_*} g^{\sigma}$$

因此，(v_*, e_*) 就相当于对 g^σ 的加密。所以，VTC 可以解密 (v_*, e_*) 并公布结果，这样所有投票者就都可以看到 g^σ。由于 σ 本身是一个小数字，所以不难从 g^σ 还原出 σ，只要通过暴力搜索或查表就可以了。

如果所有的投票者和 VTC 都正确地遵守协议，那么至少从直观上看，ElGamal 加密的语义安全能够确保在投票阶段结束时没有任何一个投票者能知道其他人的投票内容。此外，在计票阶段结束时，任何一个投票者都只能知道票数的总和，没有关于任何单独选票的额外信息会被披露。

上述协议不是很健壮，因为如果任何一个投票者或 VTC 被攻陷，选举结果的正确性和投票内容的隐私都可能受到影响。目前，我们先继续假设 VTC 是诚实的（本章后的一些练习会讨论防止 VTC 作弊的思路）。相对地，让我们把重点放在投票者作弊的可能性上。

投票者作弊的一种方式是将 0 或 1 以外的内容作为投票进行加密。比如说，他可能加密的并不是 g^0 或者 g^1，而是 g^{100}。这就相当于他投了 100 张内容为 1 的票，这将使投票者能够不公平地影响选举的结果。

为了防止这种情况，当一个投票者投票时，我们应该坚持要求他**证明**其加密的投票 (v_i, e_i) 是**有效的**，即它形如 $(g^\beta, u^\beta \cdot g^b)$，其中 $b \in \{0, 1\}$。为了做到这一点，我们对例 20.3 中的 Sigma 协议应用 Fiat-Shamir 变换。投票者（使用见证 (b, β)）只需运行图 20.1 中证明者的逻辑，通过计算陈述和承诺的哈希来自行生成挑战。在此例中为：

$$c \leftarrow H(\ (u, v, e),\ (v_{\mathrm{t}0}, w_{\mathrm{t}0}, v_{\mathrm{t}1}, w_{\mathrm{t}1})\ ) \tag{20.7}$$

投票者随后公布证明：

$$\pi = (\ (v_{\mathrm{t}0}, w_{\mathrm{t}0}, v_{\mathrm{t}1}, w_{\mathrm{t}1}),\ (c_0, \beta_{\mathrm{z}0}, \beta_{\mathrm{z}1})\ ) \tag{20.8}$$

和密文 (v, e)。任何人（特别是 VTC）都可以检查证明 π 的合法性，即检查 π 是否满足图 20.1 中验证者会验证的相应条件，区别只是现在 c 是由哈希函数 H 计算出来的，如式 20.7。

正如我们将看到的，如果我们把哈希函数 H 建模为一个随机预言机，那么该证明就是健全的。也就是说，如果加密后的投票无效，那么构造出一个合法的证明在计算上是不可行的。此外，零知识属性将确保证明本身不会泄露任何关于投票内容的额外信息。事实上，如果我们定义一个新的、增强的加密方案，其中密文的形式为 (v, e, π)，那么我们可以证明这个增强的加密方案是语义安全的（在 DDH 假设下，H 被建模为一个随机预言机）。我们把这个问题留给读者作为练习。

我们可以按照 19.2.3 小节介绍的优化 Schnorr 签名的思路来优化这个证明系统。也就是说，我们可以使用形如：

$$\pi^* = (c_0, c_1, \beta_{\mathrm{z}0}, \beta_{\mathrm{z}1})$$

的证明来代替式 20.8 中的证明 π。

为了验证该证明，我们可以从验证方程中推导出 $v_{\mathrm{t}0}, w_{\mathrm{t}0}, v_{\mathrm{t}1}, w_{\mathrm{t}1}$ 的值（即计算 $v_{\mathrm{t}0} \leftarrow g^{\beta_{\mathrm{z}0}} / v^{c_0}$，以此类推），然后检查 $c_0 \oplus c_1 = H((u, v, e), (v_{\mathrm{t}0}, w_{\mathrm{t}0}, v_{\mathrm{t}1}, w_{\mathrm{t}1}))$ 是否成立。在实践中，人们会使用这个优化后的系统，因为其证明更加紧凑，并且提供与未优化系统相同的安全属性（包括健全性和零知识性）。读者可参见练习 20.14 以了解这类优化可能存在的更多一般条件。练习 20.26 探讨了如何加强这个投票协议以对抗恶意的 VTC。

20.3.2 非交互式证明：基本语法

下面我们开始着手定义非交互式证明的一般情况及其安全属性，以及 Fiat-Shamir 变换的细节。

我们首先定义非交互式证明的基本语法。

定义 20.3 (非交互式证明系统). 令 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 是一个有效关系。$\mathcal{R}$ 上的非交互式证明系统 ***(non-interactive proof system for $\mathcal{R}$)*** 是一对算法 $(Gen, Check)$，其中：

- Gen 是一个有效概率性算法，它的调用方式为 $\pi \xleftarrow{R} Gen(x, y)$，其中 $(x, y) \in \mathcal{R}$，并且 π 属于某个证明空间 ***(proof space)*** $\mathcal{PS}$；
- $Check$ 是一个有效确定性算法，它的调用方式为 $Check(y, \pi)$，其中 $y \in \mathcal{Y}$，$\pi \in \mathcal{PS}$。$Check$ 的输出为 accept 或 reject。如果 $Check(y, \pi) = \text{accept}$，我们就称 π 是 y ***的一个有效证明 (a valid proof for y)***。

我们要求对于所有 $(x, y) \in \mathcal{R}$，$Gen(x, y)$ 的输出总是 y 的有效证明。

20.3.3 Fiat-Shamir 变换

我们下面详细介绍将 Sigma 协议转换为非交互式证明系统的 Fiat-Shamir 变换。

令 $\Pi = (P, V)$ 是一个关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议。假设 Π 的对话 (t, c, z) 属于 $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$。令 $H : \mathcal{Y} \times \mathcal{T} \to \mathcal{C}$ 是一个哈希函数。我们定义 Fiat-Shamir 非交互式证明系统 FS-$\Pi = (Gen, Check)$，其验证空间 $\mathcal{PS} = \mathcal{T} \times \mathcal{Z}$，如下所示：

- 对于输入 $(x, y) \in \mathcal{R}$，Gen 首先运行 $P(x, y)$ 以获得承诺 $t \in \mathcal{T}$，然后将挑战 $c := H(y, t)$ 转发给 $P(x, y)$ 获得应答 $z \in \mathcal{Z}$。Gen 的输出为 $(t, z) \in \mathcal{T} \times \mathcal{Z}$；
- 对于输入 $(y, (t, z)) \in \mathcal{Y} \times (\mathcal{T} \times \mathcal{Z})$，$Check$ 验证 (t, c, z) 是否是 y 的一个接受对话，其中 $c := H(y, t)$。

20.3.4 非交互式存在健全性

我们接下来把存在健全性的定义应用到非交互式的环境中。从本质上说，该定义指为一个错误的陈述制造一个有效证明是很难的。

攻击游戏 20.2 (非交互式存在健全性). 令 $\Phi = (Gen, Check)$ 是 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个非交互式证明系统，其证明空间为 $\mathcal{PS}$。为了攻击 Φ，对抗者 $\mathcal{A}$ 输出一个陈述 $y \in \mathcal{Y}$ 和一个证明 $\pi \in \mathcal{PS}$。

如果 $Check(y, \pi) = \text{accept}$ 但 $y \notin L_{\mathcal{R}}$，我们就称对手 $\mathcal{A}$ 赢得游戏。我们将 $\mathcal{A}$ 相对于 Φ 的优势记为 $\text{niESadv}[\mathcal{A}, \Phi]$，其值为 $\mathcal{A}$ 赢得该游戏的概率。

定义 20.4. 如果对所有有效对手 $\mathcal{A}$，$\text{niESadv}[\mathcal{A}, \Phi]$ 的值都可忽略不计，我们就称 Φ ***是存在健全的 (existential sound)***。

我们下面会说明，在适当的假设下，如果我们将哈希函数建模为随机预言机，那么 Fiat-Shamir 变换可以导出一个存在健全的非交互式证明系统。

定理 20.2. 设 Π 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个 *Sigma* 协议，令 FS-Π 是由 Π 派生的带有哈希函数 H 的 *Fiat-Shamir* 非交互式证明系统，如果 Π 是存在健全的，且 H 被建模为一个随机预言机，那么 FS-Π 是存在健全的。

特别地，令 $\mathcal{A}$ 是攻击游戏 *20.2* 的随机预言机版本中攻击 FS-Π 健全性的对手。此外，假设 $\mathcal{A}$ 最多发出 Q_{ro} 次随机预言机查询。那么必然存在一个攻击游戏 *20.1* 中攻击 Π 存在健全性的对手 $\mathcal{B}$，其中 $\mathcal{B}$ 是 $\mathcal{A}$ 的一个基本包装器，使得：

$$\mathrm{niES}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathrm{FS}_{\Pi}] \leq (Q_{\mathrm{ro}} + 1) \cdot \mathrm{ESadv}[\mathcal{B}, \Pi]$$

证明简述. 基本思想类似于我们在证明 Schnorr 签名方案的安全性时的分析（见定理 19.7）。假设 $\mathcal{A}$ 在一个错误的陈述 y 上产生一个有效的证明 (t, z)，这意味着 (t, c, z) 是 y 的一个有效对话，其中 c 是随机预言机在 (y, t) 点的输出。不失一般性，我们可以假设 $\mathcal{A}$ 在这一点上查询了随机预言机（即使它没有这样做，我们也可以使其如此，即把随机预言机的查询次数增加到 $Q_{\mathrm{ro}} + 1$）。然后，我们的对手 $\mathcal{B}$ 开始（预先）猜测 $\mathcal{A}$ 的随机预言机查询中的哪一个将是相关的。在 $\mathcal{A}$ 进行随机预言机查询的时候，$\mathcal{B}$ 向自己的挑战者发起证明尝试，提供 y 作为陈述，t 作为承诺；$\mathcal{B}$ 的挑战者则以一个随机挑战 c 作为应答，$\mathcal{B}$ 将其转发给 $\mathcal{A}$，就好像它就是 (y, t) 处随机预言机的值。如果 $\mathcal{B}$ 的猜测是正确的，那么 $\mathcal{A}$ 的证明中的值 z 将让 $\mathcal{B}$ 在他的攻击游戏中获胜。安全约束中存在因子 $(Q_{\mathrm{ro}} + 1)$ 是由于 $\mathcal{B}$ 猜中的概率恰好为 $1/(Q_{\mathrm{ro}} + 1)$。 □

20.3.5 非交互式零知识

令 $\Phi = (Gen, Check)$ 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个非交互式证明系统，其证明空间为 $\mathcal{PS}$。我们希望定义一个有用的 "零知识" 的概念。直观地讲，我们希望这个概念能够反映出这样的想法，即 Gen 在输入 (x, y) 上的输出，除了 $y \in L_{\mathcal{R}}$ 这一事实之外不会透露任何其他信息。

定义这样一个概念是相当棘手的。我们采取的方法类似于我们之前定义 HVZK 的方法。即我们想说，有一个模拟器在输入 $y \in L_{\mathcal{R}}$ 时可以忠实地模拟出 $Gen(x, y)$ 的输出分布。不幸的是，如果不给模拟器某种 "内部优势"，这个想法基本上是不能奏效的。事实上，如果一个模拟器能够在输入 $y \in L_{\mathcal{R}}$ 上产生一个有效证明，那么它很可能在输入 $y \notin L_{\mathcal{R}}$ 时也同样能产生一个有效证明，这就违反了存在健全性；此外，如果模拟器未能在输入 $y \notin L_{\mathcal{R}}$ 上输出一个有效证明，我们就可以仅依靠模拟器自身来区分 $L_{\mathcal{R}}$ 中的元素和 $\mathcal{Y} \setminus L_{\mathcal{R}}$ 中的元素，这对于大多数我们感兴趣的语言来说都是计算上不可行的。

我们将只会尝试在随机预言机模型基础上定义非交互式零知识，而我们赋予模拟器的 "内部优势" 是允许它同时管理 Gen 的模拟输出和对随机预言机的访问。

假设 Φ 使用一个哈希函数 $H : \mathcal{U} \to \mathcal{C}$，并且 H 被建模为一个随机预言机。**Φ 的模拟器**是一个交互式机器 Sim[1] ，它能够对一系列的查询作出响应，其中的每个查询都属于以下两类中的一类：

- 不合理证明查询 *(unjustified proof query)*，形如 $y \in \mathcal{Y}$，Sim 的应答为 $\pi \in \mathcal{PS}$；
- 随机预言机查询，形如 $u \in \mathcal{U}$，Sim 的应答为 $c \in \mathcal{C}$。

我们对非交互式零知识 (niZK) 的定义是，一个有效对手无法区分 "真实世界" 和 "模拟世界"，其中前者要求获得真陈述的真实证明，后者只能获得由 Sim 生成的模拟证明。在这两个世界中，哈希函数都 H 被建模为一个随机预言机，对手可以进行随机预言机查询，但在模拟世界中，Sim 也会处理这些查询。

攻击游戏 20.3 (非交互式零知识). 令 $\Phi = (Gen, Check)$ 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个非交互式证明系统，其证明空间为 $\mathcal{PS}$。假设 Φ 使用一个哈希函数 $H : \mathcal{U} \to \mathcal{C}$，它被建模为一个随机预言机。如上所

[1] 正式地说，一个模拟器应当是一个有效接口，如定义 2.12 所示。

初始化：
　　初始化一个空的关联数组 $Map : \mathcal{Y} \times \mathcal{T} \to \mathcal{C}$；
当收到第 i 个不合理证明查询 $y_i \in \mathcal{Y}$ 时：
　　计算 $c_i \xleftarrow{\mathrm{R}} \mathcal{C}$，$(t_i, z_i) \xleftarrow{\mathrm{R}} Sim_1(y_i, c_i)$
　　如果 $(y_i, t_i) \notin \mathrm{Domain}(Map)$，则令 $Map[y_i, t_i] \leftarrow c_i$
　　返回 (t_i, z_i)；
当收到第 j 个随机预言机查询 $(\widehat{y_j}, \widehat{t_j}) \in \mathcal{Y} \times \mathcal{T}$ 时：
　　如果 $(\widehat{y_j}, \widehat{t_j}) \notin \mathrm{Domain}(Map)$，则令 $Map[\widehat{y_j}, \widehat{t_j}] \xleftarrow{R} \mathcal{C}$
　　返回 $Map[\widehat{y_j}, \widehat{t_j}]$

图 20.2: Fiat-Shamir 的 niZK 模拟器

述，令 Sim 为 Φ 的一个模拟器。对于一个给定对手 $\mathcal{A}$，我们定义两个实验：实验 0 和实验 1。在这两个实验中，对手会向挑战者提出一系列查询，形如：

- 合理证明查询 *(unjustified proof query)*，形如 $(x, y) \in \mathcal{R}$，挑战者的应答为 $\pi \in \mathcal{PS}$；
- 随机预言机查询，形如 $u \in \mathcal{U}$，挑战者的应答为 $c \in \mathcal{C}$。

在实验 0（“真实世界”）中，挑战者随机选择 $\mathcal{O} \in \mathrm{Funs}[\mathcal{U}, \mathcal{C}]$，运行 $Gen(x, y) \in \mathcal{R}$ 以应答每个合理证明查询 (x, y)。它用 $\mathcal{O}$ 来代替 H，用 $\mathcal{O}(u)$ 应答每个随机预言机查询 $u \in \mathcal{U}$。

在实验 1（“模拟世界”）中，挑战者向 Sim 转发不合理证明查询 y 以应答每个合理证明查询 $(x, y) \in \mathcal{R}$。它向 Sim 转发随机预言机查询 u 以响应随机预言机查询 $u \in \mathcal{U}$。

对于 $b = 0, 1$，令 W_b 是对手 $\mathcal{A}$ 在实验 b 中输出 1 的事件。我们将对手 $\mathcal{A}$ 对于 Φ 和 Sim 的优势定义为：

$$\mathrm{niZKadv}[\mathcal{A}, \Phi, Sim] := |\Pr[W_0] - \Pr[W_1]|$$

定义 20.5. 如果存在一个 Φ 的有效模拟器 Sim 使得对于任意有效对手 $\mathcal{A}$，$\mathrm{niZKadv}[\mathcal{A}, \Phi, Sim]$ 的值都可忽略不计，我们就称 Φ **在随机预言机上提供了非交互式零知识**。

我们注意到，在攻击游戏 20.3 的模拟世界中，对于证明查询，对手必须提供一个见证，尽管这个见证不会被传递给模拟器。因此，模拟器只需要为真陈述生成模拟证明即可。

我们接下来表明，只要底层的 Sigma 协议是特殊 HVZK 的，并且有不可预测承诺（见定义 19.7），则 Fiat-Shamir 变换总是能导出一个 niZK。

定理 20.3. 令 $\Pi = (P, V)$ 是关系 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个特殊 *HVZK* 的 *Sigma* 协议，且其具有不可预测承诺，并令 FS-Π 是由 Π 导出的含有哈希函数 H 的 *Fiat-Shamir* 非交互式证明系统，如果 H 被建模为一个随机预言机，那么 FS-Π 是 *niZK*。

特别地，存在一个模拟器 Sim，如果 $\mathcal{A}$ 是一个如攻击游戏 *20.3* 中那样攻击 FS-Π 和 Sim 的对手，如果他最多能够发起 Q_{p} 次合理证明查询和 Q_{ro} 次随机预言机查询，并且如果 Π 有 δ-不可预测承诺，则有：

$$\mathrm{niZKadv}[\mathcal{A}, \mathrm{FS}_\Pi, Sim] \leq Q_{\mathrm{p}}(Q_{\mathrm{p}} + Q_{\mathrm{ro}}) \cdot \delta \tag{20.9}$$

证明简述. 基本思想类似于定理 19.7 中 Schnorr 签名方案的安全性证明。我们的 niZK 模拟器的原理见图 20.2。这里我们假设 Sim_1 是由 Π 的特殊 HVZK 属性保证的模拟器。读者可以自行验证定理 20.3 中的不等式，具体的论证过程与定理 19.7 的证明非常相似。我们不要求模拟器*总是*返回一个合法的证明（但是如果定义 20.5 得到满足的话，这应该以压倒性的概率发生）。 □

20.4 计算性零知识性及其应用

事实证明，对于某些关系，我们需要更宽松的零知识的概念，以便得到更有效的 Sigma 协议。我们下面先用一个例子来建立直观上的感受。

20.4.1 例子：范围证明

如 20.2 节中那样，我们还是使用乘性 ElGamal 加密方案。现在，假设 Bob 的公钥是 $u = g^\alpha \in \mathbb{G}$，私钥是 $\alpha \in \mathbb{Z}_q$。像往常一样，$\mathbb{G}$ 是一个素阶 q 的循环群，生成元为 $g \in \mathbb{G}$。

我们现在进一步推广例 20.3，假如 Alice 现在加密的不再是一个比特 b，而是一个 d 比特的数字 x，因此有 $x \in \{0, \ldots, 2^{d-1}\}$。为了进行加密，Alice 将 x 编码为一个群元素 g^x，然后用 Bob 的公钥 u 对这个群元素进行加密。由此产生的密文形如 (v, e)，其中 $v = g^\beta$，$e = u^\beta g^x$。我们假设 $2^d < q$，那么 x 的编码方案是一个双射。像之前一样，Alice 想让 Charlie 相信 (v, e) 确实来自于 Bob 的公钥加密的一个 d 比特数字，但又不透露任何其他信息。

因此，我们希望有一个关系：

$$\mathcal{R} = \left\{ \Big((\beta, \gamma, x), (u, v, e) \Big) \ : \ v = g^\beta,\ e = u^\beta \cdot g^x,\ x \in \{0, \ldots, 2^{d-1}\} \right\} \tag{20.10}$$

上的 Sigma 协议。这里我们假设 d 是一个固定的公共值。

一个直接的方法就是使用我们在例 20.3 中所使用的 OR 证明构造。也就是说，Alice 可以分别证明 $x = 0$，$x = 1$，$\cdots$，或者 $x = 2^{d-1}$。虽然这个想法可行，但由此产生的 Sigma 协议的通信和计算开销将与 2^d 成正比。事实上我们可以做得更好，我们可以构建一个复杂度与 d 成*线性*关系的 Sigma 协议，而不是成*指数*关系。

下面是具体的方法。Alice 首先用二进制表示 x，即令 $x = \sum_{i=0}^{d} 2^i b_i$，其中 $b_i \in \{0, 1\}$ 对于 $i = 0, \ldots, d-1$ 成立。接下来 Alice 分别加密每一比特。为了得到一个更简单有效的协议，她使用我们在练习 11.8 中讨论的 ElGamal 加密方案的变体。具体地，Alice 会生成一个随机公钥 $(u_0, \ldots, u_{d-1}) \in \mathbb{G}^d$，然后随机选择 $\beta_0 \in \mathbb{Z}_q$ 并计算 $v_0 \leftarrow g^{\beta_0}$；最后，她对 $i = 0, \ldots, d-1$ 计算 $e_i \leftarrow u_i^{\beta_0} g^{b_i}$。因此，$(v_0, e_0, e_1, \ldots, e_{d-1})$ 就是用公钥 $(u_0, \ldots, u_{d-1})$ 加密 $(b_0, \ldots, b_{d-1})$ 的结果。然后 Alice 将 v_0，$(u_0, \ldots, u_{d-1})$ 和 $(e_0, \ldots, e_{d-1})$ 发送给 Charlie，并向他证明 (i) 每个被加密的 b_i 都是一个比特；(ii) $\sum_i 2^i b_i = x$。为了证明 (i)，Alice 会使用一个类似于例 20.5 中所使用的技术，利用 $b_i \in \{0, 1\} \iff b_i^2 = b_i$ 这一事实。

为了证明 (i) 和 (ii)，Alice 和 Charlie 可以使用 19.5.3 小节中介绍的通用线性协议。因此，Alice 向 Charlie 证明存在：

$$\beta, x, \quad \beta_0, \quad b_0, \ldots, b_{d-1}, \quad \tau_0, \ldots, \tau_{d-1}$$

使得：

$$\left.\begin{aligned} &v = g^{\beta}, \;\; e = u^{\beta} g^{x} \\ &v_0 = g^{\beta_0} \\ &e_i = u_i^{\beta_0} g^{b_i}, \;\; v_0^{b_i} = g^{\tau_i}, \;\; e_i^{b_i} = u_i^{\tau_i} g^{b_i} \quad (i = 0, \ldots, d-1) \\ &x = b_0 + 2b_1 + \cdots + 2^{d-1} b_{d-1} \end{aligned}\right\} \tag{20.11}$$

式 20.11 中第一行表明 (v, e) 来自用 u 加密的 g^x。第二行表明每个被加密的 b_i 都是一个比特，使用例 20.5 中所展示的技术的一个变体，其中 $\tau_i = \beta_0 b_i$，第三行表明这些比特就是 x 的二进制表示。

所以，该协议的整体结构如下：

1. Alice 生成 v_0，$(u_0, \ldots, u_{d-1})$ 和 $(e_0, \ldots, e_{d-1})$，并将这些辅助群元素发送给 Charlie；
2. Alice 和 Charlie 共同运行式 20.11 所描述的通用线性 Sigma 协议。

我们首先就可以观察到，通过让 Alice 在通用线性 Sigma 协议的承诺信息之上“捎带”一些辅助群元素，这个协议在整体上仍然具有一般 Sigma 协议的基本结构。

读者可以自行证明该协议能够提供存在健全性。

在这里，我们感兴趣的问题是，这个协议在何种意义上是零知识的？问题在于，尽管通用线性协议是特殊 HVZK 的，但是整个协议并非如此，因为对 x 的比特的加密仍然会向 Charlie 透露 x 的一些信息。直观地说，在 DDH 假设下，这些加密是不应该透露任何信息的。因此该协议尽管仍然是零知识的，但只是*计算上*的零知识。为了进一步增强零知识的概念，我们下面会定义特殊*计算性* HVZK 的概念。

20.4.2 特殊计算性 HVZK

我们在定义 19.5 中定义了 Sigma 协议的特殊 HVZK 的概念。现在，我们放松定义 19.5，以定义一个较弱的特殊*计算性* HVZK 的概念，简称为特殊 cHVZK。我们的想法是，我们不要求真实定义和模拟定义的分布是*完全相同*的，而只要求它们在*计算上不可区分*。

令 $\Pi = (P, V)$ 是一个 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的 Sigma 协议，其挑战空间为 $\mathcal{C}$。如定义 19.5，Π 的*模拟器*是一个有效概率性算法 Sim，它接受 $(y, c) \in \mathcal{Y} \times \mathcal{C}$ 作为输入，并且总是输出数对 (t, z)，使得 (t, c, z) 是 y 的一个接受对话。

攻击游戏 20.4 (特殊 cHVZK). 令 $\Pi = (P, V)$ 是 $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ 上的一个 Sigma 协议，其挑战空间为 $\mathcal{C}$。如上所述，令 Sim 是 Π 的一个模拟器。对于一个给定的对手 $\mathcal{A}$，我们定义两个实验：实验 0 和实验 1。在这两个实验中，$\mathcal{A}$ 一开始就计算 $(x, y) \in \mathcal{R}$，并将 (x, y) 提交给挑战者。

- 在实验 0 中，挑战者运行 $P(x, y)$ 和 $V(y)$ 之间的协议，并将得到的对话 (t, c, z) 交给 $\mathcal{A}$。
- 在实验 1 中，挑战者计算：

$$c \xleftarrow{R} \mathcal{C}, \;\; (t, z) \xleftarrow{R} Sim(y, c)$$

并将模拟的对话 (t, c, z) 交给 $\mathcal{A}$。

在游戏结束时，$\mathcal{A}$ 计算并输出一个比特 $\hat{b} \in \{0, 1\}$。

对于 $b = 0, 1$，令 W_b 是 $\mathcal{A}$ 在实验 b 中输出 1 的事件。我们定义 $\mathcal{A}$ 对于 Π 和 Sim 的优势为：

$$\mathrm{cHVZKadv}[\mathcal{A}, \Pi, Sim] := |\Pr[W_0] - \Pr[W_1]|$$

定义 20.6. 如果存在一个 Π 的模拟器 Sim，使得对于每个有效对手 $\mathcal{A}$ 来说，$\mathrm{cHVZKadv}[\mathcal{A},\Pi,Sim]$ 的值都可忽略不计，我们就称 Π 是**特殊计算性 *HVZK*** 的，简称为**特殊 *cHZVK*** 的。

许多对特殊 HVZK 的 Sigma 协议成立的结论也对特殊 cHVZK 的 Sigma 协议成立：

- 如果使用一个 cHVZK 的协议代替 HVZK 的协议，定理 19.15 仍然成立，但具体的安全边界变成了：

$$\mathrm{ID2adv}[\mathcal{A},\mathcal{I}] \leq \mathrm{ID1adv}[\mathcal{B},\mathcal{I}] + Q \cdot \mathrm{cHVZKadv}[\mathcal{B}',\Pi,Sim]$$

  其中 Q 是在窃听攻击中获得的交互记录数量的上界。这个 Q 的系数来自于应用一个标准混合论证，它允许我们用 Q 次模拟对话代替 Q 次真实对话。

- 引理 19.17 也可以被改编以适用于 cHVZK 的协议，此时，式 19.20 中的安全边界变成了：

$$\epsilon \leq \frac{r}{N} + \sqrt{r\epsilon'} + Q \cdot \mathrm{cHVZKadv}[\mathcal{B}',\Pi,Sim]$$

  其中 Q 仍是在窃听攻击中获得的交互记录数量的上界。

- 定理 20.3 对于 cHVZK 的 Sigma 协议也仍然成立。同样地，具体的安全上界新增了一项 $Q_{\mathrm{p}} \cdot \mathrm{cHVZKadv}[\mathcal{B},\Pi,Sim_1]$，其中 Q_{p} 是证明查询的次数。

然而，我们注意到，定理 19.21（关于见证独立性）对于 cHVZK 并不成立。

范围证明. 读者可以自行证明，我们在 20.4.1 小节中介绍的证明加密数值在区间 $[0,2^d)$ 中的 Sigma 协议是特殊 cHVZK 的。

20.4.3 一种用于非线性关系的无约束通用协议

我们在 20.4.1 小节中所使用的技术可以被推广，以允许我们将形如 $x_i = x_j \cdot x_k$ 的非线性关系添加到通用线性协议所处理的线性方程组中，就像我们在 20.2.1 小节中所做的那样。然而，与 20.2.1 小节不同，我们不需要引入任何辅助方程。我们需要为这种推广所付出的代价是，我们现在只能实现特殊 cHVZK，而不是 HVZK。

同样地，令 $\mathbb{G}$ 是一个由 $g \in \mathbb{G}$ 生成的素阶 q 的循环群，令 ϕ 是一个如式 19.13 的方程，但其中包含形如 $x_i = x_j \cdot x_k$ 的非线性方程。假设证明者和验证者都得到了 ϕ，并且证明者还得到满足 ϕ 的变量 $(x_1,\ldots,x_n)$ 的一组赋值 $(\alpha_1,\ldots,\alpha_n)$。证明者采用如下方法构造一个新的方程 ϕ'。证明者首先随机选择一个 $\beta \in \mathbb{Z}_q$ 并计算 $v \leftarrow g^\beta$，然后将等式 $v = g^y$ 添加到 ϕ 中，其中 y 是一个新的变量。随后，对于 ϕ 中的每个非线性方程 $x_i = x_j \cdot x_k$，证明者随机选择 $u \in \mathbb{G}$ 并计算 $e \leftarrow u^\beta g^{\alpha_j}$，然后将方程：

$$e = u^y g^{x_j}, \quad v^{x_k} = g^t, \quad e^{x_k} = u^t g^{x_i} \tag{20.12}$$

添加到 ϕ 中，其中 t 是另一个新的变量。这就产生了一个新的可由通用线性协议处理的公式 ϕ'。然后证明者将由 v 组成的辅助群元素集合以及每个非线性方程所对应的群元素 u 和 e 发送给验证者。

基于这些辅助群元素，验证者就可以重建公式 ϕ'，这样证明者和验证者都可以对 ϕ' 运行通用线性协议。验证者为变量 y 赋值 β，为每个非线性方程 $x_i = x_j \cdot x_k$ 所产生的变量 t 赋值 $\tau := \beta\alpha_k$。此外，验证者可以在通用线性协议的承诺信息之上“捎带”辅助群元素，从而使产生的协议具有正确的通信模式。

读者可以自行证明，上述转换所产生的 Sigma 协议是特殊 cHVZK 的（在 DDH 假设下，参见练习 11.8），并且能为关系 19.14 提供知识健全性，其中的公式 ϕ 现在可以是上面介绍的非线性形式。

在上述转换过程中，有几个地方还存在改善空间。比如 u 和式 20.12 的第一个方程可以在所有 x_j 是第一个乘数的非线性方程中重复使用。同样地，t 和式 20.12 的第二个方程也可以在所有 x_k 是第二个乘数的非线性方程中重复使用。

范围证明 2. 不难看出，我们之前介绍的范围证明协议可以由上面的转换导出。Alice 想向 Charlie 证明存在：

$$\beta, x, \quad \beta_0, \quad b_0, \ldots, b_{d-1}$$

使得：

$$v = g^{\beta}, \quad e = u^{\beta} g^{x}, \quad x = \sum_{i=0}^{d-1} 2^i b_i, \quad b_i = b_i^2 \quad (i = 0, \ldots, d-1)$$

成立。读者可以自行证明，应用上面介绍的非线性到线性的转换，我们可以直接得到 20.4.1 小节中介绍的协议（在转换中，v_0 和 β_0 扮演 v 和 β 的角色）。

20.5 有效多轮协议

待写。

20.6 简洁非交互式零知识证明 (SNARKs)

待写。

20.7 一个有趣的应用：一切能被证明的事情都可以被零知识证明

待写。

20.8 笔记

要添加的文献引用。

20.9 练习

第二十一章　认证密钥交换

21.1　一个有趣的应用：建立 Tor 信道

第二十二章　安全多方计算

第四部分

附录

第二十三章　基本数论

第二十四章　基本概率论

定理 24.1. 对于所有 $p \in [0,1]$ 和 $\varepsilon < 0.3$，如果 $n = 4 \cdot 1/(p \cdot 2)$，那么 $\Delta > 0.5$

第二十五章　基本复杂性理论

第二十六章　概率性问题