第5章 以 太 坊

以太坊是一个支持智能合约功能的公有链平台。本章将主要介绍以太坊平台，5.1节介绍以太坊区块链的整体结构，5.2节介绍以太坊密钥与地址，5.3节介绍以太坊钱包，5.4节介绍以太坊交易与Gas，5.5节介绍以太坊中的智能合约，5.6节介绍以太坊虚拟机(EVM)，5.7节介绍以太坊共识协议。

5.1 区块链的整体结构

以太坊的区块链结构与比特币类似，但做了一些改进，如图5-1所示。以太坊区块中主要包含上一区块的哈希值、随机数、当前时间戳和叔区块的哈希值。叔区块指的是之前的矿

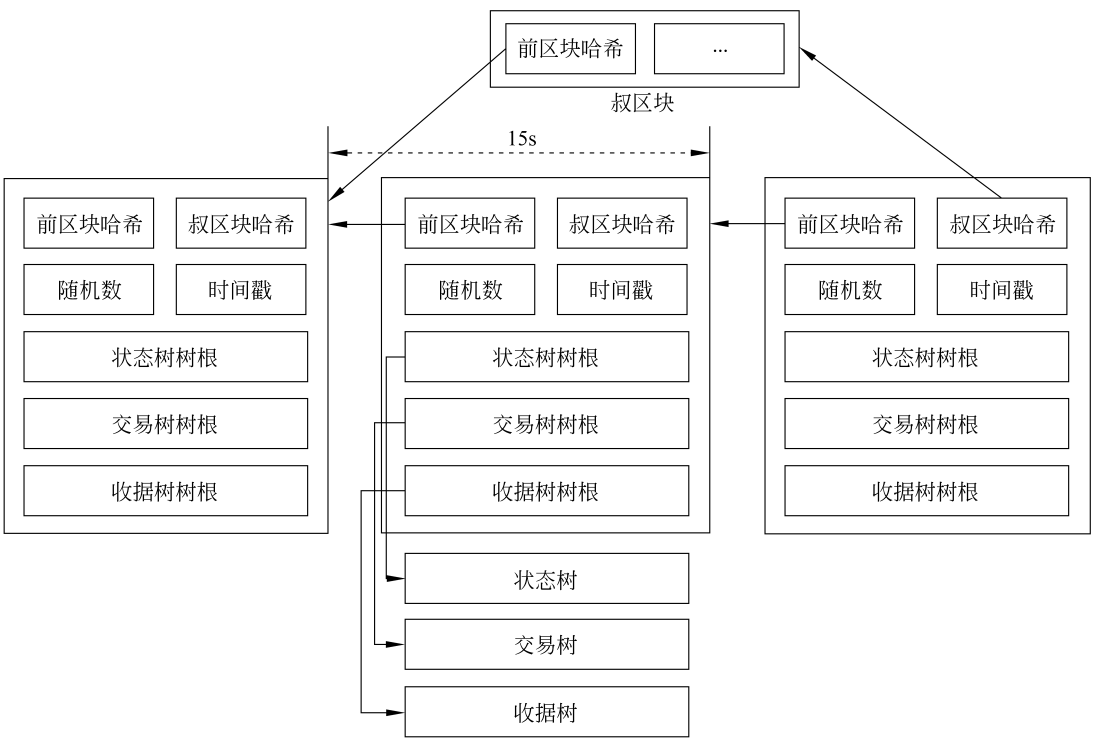

图5-1 以太坊的区块链结构图

工挖到的主链之外的孤块。换句话说，一个区块的叔区块是该区块的父区块的父区块的子区块，同时又不能是自己的父区块。为了提高叔区块的利用率，矿工若引用叔区块，可以获得叔区块引用奖励。与比特币相比，以太坊的区块间隔仅为 15s，较易产生分叉，叔区块引用机制可以看作一种缓解分叉问题的措施。另外，以太坊区块中还包含了 3 棵默克尔 Patricia 树，分别是状态树、交易树和收据树，这 3 棵树的树根存储在区块的头部。

5.2 密钥与地址

以太坊使用非对称加密创建公私密钥对，公钥由私钥派生而来，大多数以太坊钱包工具将私钥和公钥作为密钥对存储在一起。公钥和私钥共同表示以太坊账户，公钥提供可公开访问的账户地址，私钥提供对账户的私有控制权。私钥通过作为创建数字签名所需的唯一信息控制账户和签署交易，以使用账户中的资金。另外，数字签名还用于验证合约的所有者或使用者。当一个交易被用户发送到以太坊网络以转移资金或与智能合约交互时，这个交易需要包含一个与该以太坊地址对应的数字签名。任何人都可以通过检查数字签名的正确性验证交易是否有效，这种验证完全不涉及私钥。

以太坊有两种不同类型的账户，即外部账户和合约账户，如表 5-1 所示。外部账户是由公钥生成且由私钥控制的账户，用来发送交易和存储以太币。合约账户是由外部账户通过交易创建的，不受私钥控制，而受合约代码控制。每当合约账户收到一条交易消息时，其合约代码被交易输入的参数调用执行。

表 5-1　外部账户和合约账户对比

账 户 类 型	外 部 账 户	合 约 账 户
地址	Hash(公钥)	Hash(外部账户地址＋交易)
余额	账户余额，表明该账户控制的以太币的数量	
控制权	私钥	合约代码
代码	无	合约代码受交易或其他合约消息调用而执行

5.3 钱包

在以太坊中，钱包具有多个含义。广义地讲，钱包指的是软件应用程序，是以太坊的主要用户界面。钱包负责控制对用户资金的访问，管理密钥和地址，追踪余额以及创建和签署交易。此外，一些以太坊钱包还可以与智能合约执行交互，如 ERC20 令牌。狭义地讲，钱包是指用于存储和管理用户密钥的系统，每个钱包都有一个密钥管理系统，一些钱包甚至仅有密钥管理功能，而无其他额外功能。本节将以太坊钱包定义为以太坊私钥的存储器及私钥的管理系统。对以太坊的一个常见误解是，以太坊钱包中包含以太币或代币。事实上，钱包里只有私钥，而以太币或其他代币被记录在以太坊区块链上。用户可以使用钱包中的私钥创建和签署交易，以控制区块链上的以太币或代币转账。

对于传统银行业的中心化系统，用户账户里的余额只有用户本人和银行能看到，只要银行相信用户想转移资金，用户就可以发起交易；而对于以太坊这样的去中心化区块链平台，所有人都能在不知道账户所有者的前提下看到一个账户里的以太币余额。而且，用户若想转移资金，必须对交易数字签名，让所有人相信该用户要转移资金。此外，如果用户不再想使用当前的钱包，用户可以将以太币从当前的钱包转移到另一个钱包。

设计者设计钱包时要考虑的一个关键问题是易用性和隐私保护之间的权衡。最简单的以太坊钱包只包含一个私钥和一个地址。然而，这样的钱包设计方案是用户的隐私噩梦，因为任何人都可以轻松追踪和关联属于同一用户的所有交易。用户为每个交易使用一个新密钥是保护隐私的最佳方法，但这样做，钱包会非常不方便管理。设计者很难在易用性和隐私保护之间找到最佳的平衡，这也是良好的钱包设计至关重要的原因。

以太坊钱包主要有两种类型，区别在于钱包中的密钥是否相互关联。第一种类型是不确定性钱包，其中每个密钥都是由不同的随机数独立生成的。这些密钥之间彼此不相关。第二种类型的钱包是确定性钱包，其中所有密钥都派生于一个主密钥，这个主密钥称为种子。确定性钱包中的所有密钥都是相互关联的，只要用户保存好种子密钥，则可以重新生成所有密钥。为了使确定性钱包能防范数据丢失事故，种子密钥通常被编码为单词列表，用户可以记录下这个单词列表，以便在发生事故时找回钱包。这些单词列表被称为钱包的助记词。当然，如果用户的记忆码被其他人获得，则其他人也可以重新创建该用户的钱包，从而控制该用户的以太币和智能合约。因此，用户要非常小心地保管记忆码。

5.4 交易与 Gas

交易是由外部账户产生的包含签名的消息，在以太坊网络上传输，并被记录在以太坊区块链上。在以太坊中，交易还有另一种含义，那就是能触发状态变化或触发智能合约在以太坊虚拟机(Ethereum Virtual Machine，EVM)中执行的唯一事物。读者可以把以太坊看作一个全局状态机，则交易可以改变该状态机的状态。智能合约不能自动触发，相应地，以太坊不能自动运行，一切改变都源自交易。接下来，5.4.1 节介绍以太坊交易结构，5.4.2 节介绍以太坊交易计数 Nonce，5.4.3 节介绍以太坊中的 Gas，5.4.4 节介绍以太坊交易传播机制。

5.4.1 交易结构

交易以序列化的形式在以太坊网络上传输，接收到交易后客户端和应用程序使用自己的内部数据结构将其存储在内存中，可能还会使用交易中不存在的元数据对交易加以修饰。网络序列化是交易的唯一标准形式。一个以太坊交易是一个序列化的二进制消息，交易格式如图 5-2 所示。

其中，From 和 To 分别指交易的发送地址和接收地址。Value 指的是要发送到目的地址的以太币数量。Data 指的是交易可能包含的智能合约代码。Gas 是用来衡量一笔交易所消耗的计算资源的基本单位。以太坊节点执行一笔交易所需的计算步骤越多、越复杂，则交易消耗 Gas 越多。Gas Limit 指的是交易发起人愿意为该交易支付的最大 Gas 数量，需要发送者在广播交易时设置。Gas Price 指的是交易发起人愿意为每单位 Gas 支付的以太

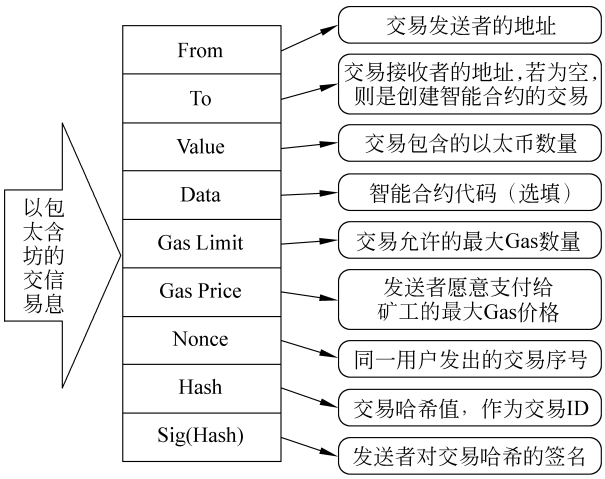

图 5-2　以太坊交易格式

币数量，用户创建交易时可以自己设定任意大小的 Gas 价格。Nonce 指的是同一用户发出的交易序号，用于防止重放攻击。Hash 指的是交易的哈希值，作为交易的 ID。Sig(Hash) 指的是广播交易的外部账户对交易的 ECDSA 数字签名。

5.4.2　交易计数 Nonce

这里的 Nonce 和比特币中用来寻找合适哈希的 Nonce 功能不同，以太坊中使用 Nonce 字段作为交易的唯一标识。每发起一笔交易，Nonce 就加 1。因此，对于外部账户而言，Nonce 代表发送该交易的账户总共产生的交易数量；而对于合约账户而言，Nonce 代表由该账户创建的合约数量。下面两个案例可以帮助读者理解 Nonce 的含义及作用。

案例一：Alice 希望产生两笔交易，分别需要支付 6 个以太币(记为交易 A)和 8 个以太币(记为交易 B)。Alice 首先对交易 A 签名和广播，再对交易 B 签名和广播。遗憾的是，Alice 的账户只包含 10 个以太币。因此，以太坊网络中的其他用户不能同时接收这两个交易。读者可能认为，因为 Alice 先发送的是交易 A，所以交易 A 会通过验证，交易 B 会被网络中的节点拒绝。然而，在以太坊的去中心化网络中，节点可能按任何一种顺序接收这两个交易。因此，可以肯定的是，网络中一些节点先接收到交易 A，而另一些节点先接收到交易 B。如果没有 Nonce，节点就会随机选择接收一个交易，拒绝另一个交易。然而，交易是包含 Nonce 的，Alice 先产生的交易 A 将具有 1 个 Nonce(如 3)，而后产生的交易 B 具有下一个 Nonce(如 4)。因此，一个节点即使先收到交易 B，也不会立刻处理，而是等到该节点处理完 Nonce 值为 3 的交易 A 后，再处理交易 B。

案例二：Bob 有一个包含 100 个以太币的账户，他找到了一个愿意接收以太币付款的商家，想购买一个 2 个以太币的商品。Bob 签署了 1 个交易，将 2 个以太币从 Bob 的账户发送到商家的账户，然后将其广播到以太坊网络。假设交易中没有 Nonce 字段，任何人只要在以太坊网络上看到 Bob 的交易，只要复制和粘贴 Bob 的原始交易并将其重新广播到以太坊网络，就可以一次又一次地重播交易，直到 Bob 所有的以太币都被取出。交易的 Nonce 值确保了即使用户多次将相同数量的以太币发送到相同地址，每个交易也是唯一的。因此，以太坊通过将递增的 Nonce 作为交易的一部分，保证了任何人都不可能复制 Bob 的付款。

总之，Nonce 可以起到对交易排序并防止重放攻击的作用，对以太坊这种基于账户的共识而言是十分重要的。

5.4.3 Gas

在以太坊中，交易将被全球数以千计的计算机处理，开放式的、图灵完备的计算模型需要某种形式的度量，以避免拒绝服务攻击或大量消耗资源的交易。以太坊使用 Gas 控制交易可以使用的资源数。顾名思义，Gas 是驱动以太坊执行交易的燃料，其不等同于以太币，而是一种单独的虚拟货币，有自己对以太币的汇率。

以太坊交易与 Gas 有关的第一个重要字段是 Gas 价格，表示交易发起者愿意为每单位 Gas 支付的以太币数量(以太坊钱包客户端默认的 Gas 价格是 0.000000001ETC/Gas)。价格一般以 Wei/Gas 单位度量(1ETC＝10^{18} Wei)。以太坊钱包可以调整交易的 Gas 价格，Gas 价格越高，交易被确认的速度越快，相反，低优先级的交易可以设定低 Gas 价格。以太坊可接收的最低 Gas 价格是 0，这意味着钱包可以产生没有交易费的交易。这些交易可能永远不会被确认，因为矿工得不到任何报酬，但是以太坊协议中并没有禁止产生 Gas 价格为 0 的交易。读者可以在以太坊区块链上找到一些此类交易成功被包含进区块的实例。

以太坊交易与 Gas 有关的第二个重要字段是 Gas 限制，给出了交易发起者为了完成交易而愿意支付的最大 Gas 数量。如图 5-3 所示，用户创建交易时自己设定任意大小的 Gas 价格后，为交易设置一个合理的最大 Gas 限制，然后广播交易。矿工接收到交易后需要验证交易，记录验证过程中所消耗的 Gas 数量，记为 Gas 消耗。当交易验证成功后，若 Gas 消耗不超过 Gas 限制，则矿工收取 Gas 消耗×Gas 价格的交易费(注意，不是 Gas 价格×Gas 最大限制)，并接收该交易；如果在交易还未验证完毕时，矿工消耗的 Gas 数量已经达到 Gas 限制，则矿工收取 Gas 限制×Gas 价格的交易费，并放弃处理该交易。

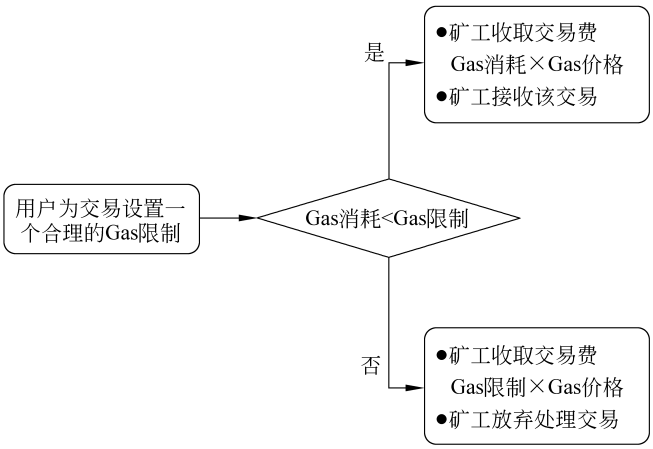

图 5-3　以太坊 Gas 原理

以太坊采用 Gas 机制的主要目的是防止恶意用户发动拒绝服务攻击，如果没有 Gas 限制，以太坊恶意用户可以产生一些需要执行数十亿步的恶意智能合约，验证此类交易将白白消耗极大算力，甚至有限时间内也算不完。Gas 限制使得矿工在验证一个交易之前可以先查看此交易的 Gas 限制，如果该交易的 Gas 限制值过大，超过了验证能力，矿工可以放弃验

证此交易，转而验证那些 Gas 限制值不大的交易。

5.4.4　交易传播机制

以太坊客户端是以太坊点对点网络中的节点，组成了网状网络。节点与节点之间地位平等，没有任何网络节点具有特权。本章将使用术语节点指代连接到点对点网络的以太坊客户端。以太坊的交易传播从原始节点创建一个已经被签名的交易开始。原始节点对交易签名，将交易发送给所有与之直接相连的以太坊节点。每个以太坊节点平均至少与其他 13 个节点保持连接，这些节点被称为它的邻居节点。每个邻居节点接收到交易后立即验证交易。如果验证通过，这些邻居节点存储一个副本，并将此交易传播给所有邻居，但是之前发给自己的那个邻居除外。由此，交易从原始节点向外扩散，在网络中像洪水一样扩散，直到网络中的所有节点都拥有该交易的副本。节点可以过滤传播的消息，即选择自己传播哪些交易，但默认情况下，节点会传播接收到的所有有效交易。

一个以太坊交易通常短短几秒钟内会被传播到全球所有以太坊节点。每个节点接收到一个交易后，不可能识别出原始节点，也就是最先发送该交易的节点。传播一个交易的节点可能是该交易的发起者，也可能是该交易的传递者。追踪者要想追踪交易的起源或干扰交易的传播，必须控制大部分以太坊节点。以太坊中矿工节点收集交易，将交易添加到候选区块中，并使用计算机尝试寻找使得该候选区块有效的工作量证明。有效的交易最终将被包含在一个区块中，并被记录在以太坊区块链上。

5.4.5　多重签名交易

如果读者熟悉比特币的脚本功能，就知道用户可以创建一个多重签名(Multisig)账户，其脚本如图 5-4 所示。上半部分是一笔交易的输出脚本，A、B、C 的公钥代表必须提供使用与这 3 个公钥对应的私钥的数字签名；数字 3、2 代表必须提供 3 个签名中的 2 个才能解锁，才能使用本交易输出的比特币。下半部分是另一笔交易的输入脚本，这笔交易的发起人提供了 A、B 的数字签名，故可以解锁上一笔交易输出的比特币，将这些比特币转移到其他账户。以太坊外部账户产生的交易并不提供多重签名功能。但是，以太坊用户可以通过设计智能合约任意地设定签名限制，以掌控以太币和代币的转移。为了利用这一功能，以太币必须被转移到被编写了支出条件的“钱包合约”中，其中支出条件指的是钱包合约要求其他用户想花费这些以太币必须提供某些信息，比如多重签名或支出限制(或两者的组合)。例如，假设一个用户想使用多重签名保护以太币，他可以将以太币转移到一个多重签名合约中。每当用户想将这些以太币转移到另一个账户时，所有多重签名合约要求提供签名的用户都需要向合约地址发送交易，从而有效地授权合约执行最终的交易。多重签名合约的安全性主要由多重签名合约代码决定。

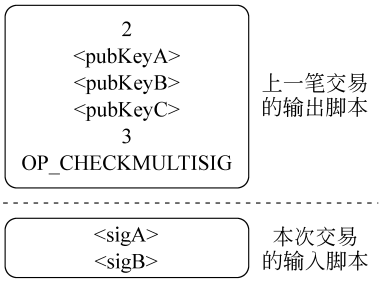

图 5-4　比特币多重签名脚本

5.5 智能合约

密码学家 Nick Szabo 提出了智能合约的概念，将其定义为“一个智能合约是一套以数字形式定义的承诺，包括合约参与方可以在上面执行这些承诺的协议”。其中，承诺指合约参与方同意的权利和义务。在以太坊中，智能合约指代在以太坊虚拟机的上下文中作为以太坊网络协议的一部分而确定性地运行的不可变计算机程序。也就是说，智能合约是一种计算机程序，部署后其代码就无法更改；而且智能合约的执行环境非常有限，仅可以访问自己的状态，调用交易的上下文，以及有关最新区块的一些信息。

5.5.1 节介绍智能合约的生命周期，5.5.2 节介绍构建智能合约所使用的几种高级编程语言，5.5.3 节介绍智能合约的安全性以及可能出现的安全漏洞。

5.5.1 生命周期

智能合约使用高级语言编写，例如 Solidity。但是，为了在以太坊中正常运行，必须将它们编译为在以太坊虚拟机中运行的低级字节码。编译完成后，它们将使用特殊的合约创建交易并部署在以太坊平台上，该交易被发送到特殊合约创建地址（即 0x0）。每个合约都由以太坊地址唯一标识，该地址是作为原始账户和随机数的函数从合约创建交易中获得的。合约的以太坊地址可以作为收件人在交易中使用，将资金发送给合约或者调用合约的函数。需要注意的是，合约创建者在协议级别没有任何特殊权限，不会收到合约账户的私钥。

以太坊合约实际上处于休眠状态，直到交易触发执行，并作为合约调用链的一部分。合约代码虽然不能更改，但合约本身可以删除。执行名为 SELFDESTRUCT 的 EVM 操作码就可以从区块链中删除合约，这种方式不会删除合约的交易历史，因为区块链是不可篡改的。

5.5.2 智能合约的构建

智能合约的高级编程语言包括以下几种。

（1）LLL。一种函数式（声明式）编程语言。这是以太坊采用的第一种高级语言，但现在很少使用。

（2）Serpent。一种过程式（命令式）编程语言，语法类似于 Python，也可用来编写函数式（声明式）代码，但是使用较少。

（3）Solidity。过程式（命令式）编程语言，其语法类似于 JavaScript、C++ 或 Java，是以太坊智能合约最流行、最常用的语言。

（4）Bamboo。一种新开发的语言，具有显式的状态转换，没有迭代流（循环），旨在减少副作用，增加可审核性，很少使用。

（5）Vyper。一种针对以太坊虚拟机的实验性、面向合约的编程语言，具有类似于 Python 的语法。

Vyper 与 Solidity 的区别主要体现在以下 4 方面。

（1）**修饰符**。Solidity 语言使用修饰符执行检查以及在调用函数的上下文中改变智能

合约的环境。而 Vyper 取消了修饰符，提高了可审核性和可读性。

（2）**类继承**。允许程序员通过从现有软件库获取预先存在的功能、属性和行为来使用预编的代码功能。Solidity 支持多重继承和多态性，但 Vyper 中取消了继承，它认为继承会使代码复杂化，从而产生安全性问题。

（3）**内联汇编**。允许 Solidity 通过直接访问 EVM 指令来执行操作，但可读性造成的损失太大，因此 Vyper 不支持内联汇编。

（4）**函数重载**。允许程序员编写多个同名函数，具有相同名称且带有不同参数的多个函数定义可能会造成混淆，因此 Vyper 不支持函数重载。

目前而言，Vyper 由于生态较小、功能受限以及疏于维护等原因，其使用范围不如 Solidity 广泛，故目前最主流的智能合约开发语言仍是 Solidity。

5.5.3　智能合约的安全性

编写智能合约时必须考虑安全性。智能合约代码错误会导致高昂的代价且容易被攻击者利用。与其他程序一样，智能合约将完全执行所写的内容，而不管程序员的意图。另外，所有智能合约都是公开的，任何用户都可以通过创建交易与合约进行交互。因此，遵循最佳实践并使用经过良好测试的设计模式至关重要。任何漏洞一旦被利用，都将造成无法恢复的损失。本节将介绍安全性的最佳实践，以及智能合约可能出现的漏洞。

1. 安全性的最佳实践

防御性编程是一种特别适合智能合约的编程风格，具体包含以下几方面。

（1）**简约性**。代码越简单，程序出现错误或无法预料的效果的可能性越小。当第一次参与智能合约编程时，开发人员往往试图编写大量代码。相反，智能合约的编程应该追求更少的代码行、更低的复杂性和更少的功能。如果有项目方声称他们的智能合约生成了数千行代码，那么该项目的安全性就该被质疑了。

（2）**重复利用性**。编写智能合约时，尽量使用已有的库或合约。在编程人员自己的代码中，应该遵循避免重复代码原则。如果看到任何代码片段不止重复一次，则应该考虑是否可以将其编写为函数或库并重复使用。另外，已经广泛使用和测试的代码可能比编写的任何新代码更安全。安全风险通常大于改进价值。

（3）**代码质量**。正如前面提到的，智能合约一经部署，其代码就无法更改；唯一的方法是删除合约，重新建立。因此，合约中每个漏洞都可能导致巨额损失。编程人员应该严格遵循工程和软件开发方法来对待智能合约。

（4）**可读性**。智能合约是公开的，每个以太坊用户都可以读取其中的字节码。考虑到这一点，智能合约代码应当清晰易懂。因此，建议使用协作和开源方法在公共场合开发，可以利用开发人员社区的集体智慧，并从开源开发的最高共同点中受益。

（5）**测试覆盖率**。智能合约在公共环境中运行，任何人都可以选择任何输入来运行合约。应尽可能测试所有参数，以确保合约在预期范围内并且在允许执行代码之前正确格式化。

2. 智能合约漏洞

本节介绍以太坊智能合约常见的安全漏洞，包括重入漏洞、算术溢出漏洞、异常以太等。

智能合约漏洞概览如表 5-2 所示。

表 5-2　智能合约漏洞概览

漏 洞 名 称	漏 洞 简 介
重入漏洞	攻击者劫持外部调用,强制合约执行更多的代码
算术溢出漏洞	攻击者滥用算术溢出并创建恶意逻辑流
异常以太	攻击者不执行任何代码就能操控以太币
默认可见性	用户错误使用可见性说明导致漏洞
随机数误区	矿工控制交易区块信息的变量
外部合约引用	攻击者利用外部消息调用以掩盖恶意意图
短参数漏洞	攻击者向智能合约传递短编码
未检查返回值	外部调用失败时不检查返回值,但期望交易恢复
提前交易	攻击者利用提前交易创建 GasPrice 更高的交易
审查攻击	攻击者通过操控合约审查逻辑,阻止来自某些地址的交易正常执行
拒绝服务	攻击者把以太币永远锁在合约中
区块时间戳操控	用户错误地使用区块时间戳
构造函数失控	合约名称变化时构造函数名称不变,导致合约攻击
存储指针未初始化	未初始化的本地存储变量导致漏洞
浮点和精度	浮点在 Solidity 中不是整数类型,进而导致漏洞

(1) **重入漏洞**。智能合约的一个特点是能外部调用其他外部合约的代码,而外部调用可能被攻击者劫持,攻击者可以强制合约执行更多的代码,包括回调自身。一个典型的案例是 DAO 攻击,这次攻击就利用了重入攻击。攻击者可以在 fallback()函数里包含恶意代码的外部地址中构建合约,当合约将以太发送到此地址时,它将调用恶意代码。避免重入漏洞的方法有 3 种。第一种是在向外部合约发送以太时尽可能使用内置 transfer()函数。第二种技术是确保所有改变状态变量的逻辑在以太被发送(或任何外部调用)之前生效。第三种技术是引入互斥锁,即添加一个状态变量,在代码执行期间锁定合约,防止重入调用。

(2) **算术溢出漏洞**。算术溢出包括算术上溢和算术下溢。当操作需要固定大小的变量来存储超出变量数据类型范围的数字(或数据)时会发生上溢或下溢,即如果将一个数字添加到最大值上,则将从 0 开始向上计数;如果从零减去一个数字,则将从最大数字开始向下计数。这些数字陷阱允许攻击者滥用代码并创建预料之外的逻辑流。目前,防止溢出漏洞的传统技术是使用数学库代替标准数学运算符的加法、减法和乘法(除法除外,因为它不会导致上溢或下溢,EVM 会在除以 0 时执行回滚交易)。

(3) **异常以太**。通常将以太发送给合约时,必须执行 fallback()函数或合约中定义的其他函数。但是,在异常情况下,合约不执行代码却可以操控以太,这会导致以下攻击,即只有执行代码才能操控以太的合约容易受到强制发送以太攻击。针对此漏洞最常用的防御性编程技术是不变检查。此技术定义了一组不变的度量或参数,并检查它们在单个或多个操作

后是否保持不变。特别指出一点,存储在合约中的当前以太币很容易由外部用户操纵。

(4) **默认可见性**。Solidity 中的函数具有可见性说明符,用于决定用户或其他派生合约可否仅在内部或外部调用函数。Solidity 定义了 4 种可见性,包括 external、public、internal、private,默认可见性是 public,即允许用户从外部调用函数。错误使用可见性说明符将导致智能合约产生破坏性漏洞。避免这种漏洞的方法是始终指定合约中所有函数的可见性。

(5) **随机数误区**。区块链上的所有交易都是确定性的状态转换操作,即在以太坊中没有熵或随机性的来源。一些基于赌博的以太坊合约需要不确定性,而区块链是确定性系统,因此不确定性必须来自区块链外部的来源。常见的陷阱是使用未来区块的变量,也就是包含值未知的有关交易块信息的变量,如哈希、时间戳等。而这些变量由矿工控制,并非真正随机的。使用过去或现在的变量可能更具破坏性。另外,使用单个块的变量意味着块中交易的伪随机数都是相同的,因此攻击者可在单个块内执行多次交易。

(6) **外部合约引用**。以太坊的一个好处是能重用代码并与已经部署在网络上的合约交互。因此,通常通过外部调用引用外部合约。这些外部调用可以用一些不明显的方式掩盖恶意意图,因此也成为攻击者可以利用的途径。

(7) **短参数漏洞**。此攻击在与智能合约交互的第三方应用程序上执行。将参数传递给智能合约时,参数将根据应用二进制接口规范编码。编码参数短于预期参数长度时,EVM 将在编码参数的末尾添加 0 以构成预期长度。当第三方应用程序不验证输入时,将会产生问题。为防止这类攻击,一方面,应用程序中的输入参数在发送到区块链之前应执行验证。另一方面,由于填充发生在字符串末尾,因此参数的排序也很重要。

(8) **未检查返回值**。以太坊通常使用 transfer()将以太发送到外部账户,send()也可用于此。call()和 send()通过返回一个布尔值表明调用是否成功。这些函数都设置了警告:如果外部调用失败,执行这些函数的交易将不会恢复;相反,函数将返回 false()。因此产生了一个常见问题:在外部调用失败时开发人员不检查返回值,但期望交易能恢复。所以,尽可能使用 transfer(),而不是 send();若使用 send(),则需检查返回值。另外,建议采用回退(withdraw)模式。

(9) **提前交易**。矿工根据交易的 Gas 价格排序来选择矿池中的交易包含在区块中。这是一个潜在的攻击媒介,即攻击者可以从交易中获取数据并以更高的 Gas 价格创建交易。

(10) **审查攻击**。交易执行者可以筛选发起交易的地址,对来自某些地址的交易采用忽略的策略,从而屏蔽这些地址的用户使用区块链系统。

(11) **拒绝服务**。这类攻击非常广泛,基本攻击形式都是让用户短暂地(在某些情形下则是永久地)不可操作合约,可以把以太永远锁在合约中。为预防此类攻击,合约中不应含有可被外部用户操纵的数据结构。

(12) **区块时间戳操控**。区块时间戳用于生成随机数、锁定一段时间内的资金、各种基于时间变更状态的条件语句。矿工可以调整时间戳,如果在智能合约中错误地使用块时间戳,则相当危险。

(13) **构造函数失控**。构造函数是特殊函数,在初始化合约时经常执行关键的任务。在 Solidity 0.4.22 版本之前,构造函数与包含该函数的合约具有相同的名称。在智能合约开发过程中,合约名称变化时,如果构造函数的名称没有改变,它就变成了一个正常的、可调用的