

[学习](#) [开发者](#) [企业级应用](#) [社区](#)

搜索



语言

[首页](#) / [WHITEPAPER](#)

页面最后更新: 2023年9月19日

以太坊白皮书

这篇介绍性论文最初由[以太坊](#)创始人 Vitalik Buterin 在 2014 年发表，前于 2015 年的项目正式发布时间。值得一提的是，和其他社区驱动的开源软件项目一样，以太坊自发布以来一直不断发展。

虽然已经过数年，但由于本文仍然可提供有用的参考并能够准确表述以太坊及其愿景，我们仍然在维护它。若想了解以太坊的最新进展，以及以太坊协议的更新情况，我们推荐您阅读[本指南](#)。

[寻求此白皮书早期版本或规范版本 \[自 2014 年 12 月起\] 的研究人员和学者应使用此 PDF。](#)

下一代智能合约和去中心化应用平台

中本聪 2009 年开发的比特币常被誉为资金和货币的一次革命性变革，作为数字资产的首个实例，它同时具有以下特点：没有实物或[内在价值](#) 支撑，也没有一个中心化的发行机构或控制者。然而，比特币实验有另一个可以说是更重要的部分，即作为分布式共识工具的底层区块链技术，并且人们的注意力正迅速地开始转移到比特币的这个方面。经常被提到的其他区块链技术应用包括：使用链上数字资产表示自定义货币和金融工具（“[彩色币](#)”）、底层物理设备的所有权（“[智能资产](#)”）、非

在本页面

[下一代智能合约和去中心化应用平台](#)[比特币及现有概念简介](#)[历史](#)[比特币是一个状态转换系统](#)[挖矿](#)[默克尔树](#)[其它的区块链应用](#)[脚本](#)

以太坊

[以太坊帐户](#)[消息和交易](#)[消息](#)[以太坊状态转换函数](#)[代码执行](#)[区块链和挖矿](#)

应用

[代币系统](#)[金融衍生品和价值稳定的货币](#)[身份和信誉系统](#)[去中心化文件存储](#)[去中心化自治组织](#)[更多应用](#)

杂项和关注

改进版 GHOST 协议的实现

同质化资产例如域名（“[域名币](#)”），以及一些更复杂的应用，例如让数字资产由一段实现任意规则的代码（“[智能合约](#)”）甚至由基于区块链的“[去中心化自治组织](#)”(DAO) 直接控制。以太坊打算提供一种内置完全成熟的图灵完备编程语言的区块链，这种语言可用来创建“合约”，而合约可用于编码任意状态转换函数，让用户可以创建上述任何系统以及我们尚未想象到的许多其他内容，只需用几行代码编写出想实现的逻辑即可。

比特币及现有概念简介

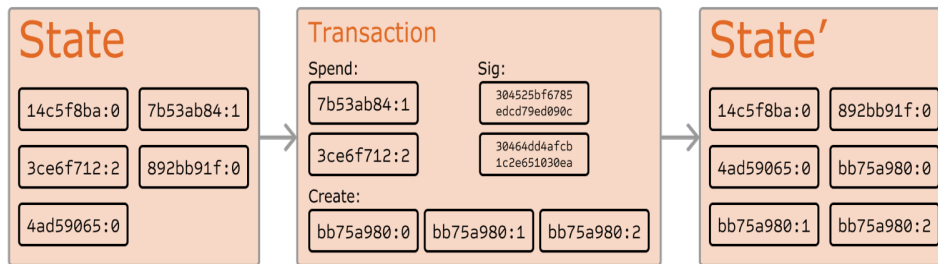
历史

去中心化数字货币的概念以及财产登记等其他应用已经存在了几十年。1980 年代和 1990 年代的匿名电子现金协议主要依赖于称为 Chaumian 盲签名的密码学原语，提供了一种具有高度隐私性的货币，但这些协议基本上未能获得关注，因为它们依赖于中心化中介。1998 年，戴伟 (Wei Dai) 的 [b-money](#) 成为第一个提出通过解决计算难题来创造货币及去中心化共识等想法的协议，但该协议缺乏关于如何实际实施去中心化共识的细节。2005 年，Hal Finney 引入了“[可重复使用的工作量证明](#)”这一概念，该系统将 b-money 的想法与 Adam Back 有计算难度的哈希现金难题相结合来创建加密货币的概念，但由于依赖可信计算作为后端，又一次未能做到完美。2009 年，中本聪将通过公钥密码学管理所有权的成熟原语与用于跟踪货币所有者的共识算法相结合，首次真正意义上实现了一种去中心化货币，被称为“工作量证明”。

工作量证明机制是该领域的一项突破，因为它同时解决了两个问题。首先，它提供了一种简单且比较有效的共识算法，让网络中的节点能够全体对比特币账本状态的一组规范更新达成一致。其次，它提供了一种允许自由进入共识过程的机制，解决了决定谁来影响共识的政治问题，同时防止了女巫攻击。为此，在工作量证明中，将正式的参与壁垒（例如要求在特定清单上注册成为唯一实体）替换成经济壁垒，即共识投票过程中单个节点的权

重与该节点的算力成正比。此后，还提出了另一种称为**权益证明**的方法，节点权重与其货币持有量而非计算资源成正比；针对这两种方法相对优点的讨论不在本文范围内，但应该注意，这两种方法都可以作为加密货币的支柱。

比特币是一个状态转换系统



从技术角度讲，诸如比特币等加密货币账本可视为一种状态转换系统，该系统有一个“状态”，由全部现存比特币的所有权状态和一个“状态转换函数”组成，状态转换函数以状态和交易为输入并输出新状态作为结果。例如，在标准的银行系统中，状态就是一个资产负债表，一笔交易是一个从 A 帐户向 B 帐户转账\$X的请求，状态转换函数将从A帐户中减去\$X，向 B 帐户增加\$X。如果A帐户的余额在第一步中小于\$X，状态转换函数就会返回错误提示。所以，可以如此定义：

```
APPLY(S,TX) -> S' or ERROR
```

上面提到的银行系统中，状态转换函数如下：

```
APPLY({ Alice: $50, Bob: $50 }, "send $20
from Alice to Bob") = { Alice: $30, Bob: $70
}
```

但是：

```
APPLY({ Alice: $50, Bob: $50 }, "send $70  
from Alice to Bob") = ERROR
```

比特币中的“状态”是指所有已铸造但尚未使用的货币（技术上称为“未使用的交易输出”或 UTXO）的集合，每个 UTXO 都有面额和所有者（由一个 20 字节的地址定义，本质上是一个加密公钥 [fn1]（注释编号））。一个交易包括一个或多个输入以及一个或多个输出，每个输入都包含对现有 UTXO 的引用以及所有者地址相关的私钥创建的加密签名；每个输出都包含一个要添加到状态中的新 UTXO。

状态转换函数 $APPLY(S, TX) \rightarrow S'$ 的定义大体如下：

1. 对于 TX 中的每个输入：
 - 如果引用的 UTXO 不在 S 范围内，则返回错误。
 - 如果提供的签名与 UTXO 的所有者不符合，则返回错误。
2. 如果所有输入 UTXO 面值总额小于所有输出 UTXO 面值总额，则返回错误。
3. 在移除所有输入 UTXO 且添加所有输出 UTXO 后，返回 S。

第一步的第一部分防止交易发送者花费不存在的比特币，第二部分防止交易发送者花费其他人的比特币，第二步确保价值守恒。为了用于支付，比特币协议如下。假设 Alice 想给 Bob 发送 11.7 BTC。首先，Alice 将寻找她拥有的一组总数至少为 11.7 BTC 的可用 UTXO。事实上，Alice 不太可能正好有 11.7 BTC；假设她能得到的最小数额是 $6+4+2=12$ 。所以，她可以创建一笔有三个输入和两个输出的交易。第一个输出为 11.7 BTC，所有者是 Bob 的地址，第二个输出为剩下的 0.3 BTC 找零，所有者是 Alice 自己。

挖矿



如果我们拥有可信任的中心化服务机构，状态转换系统可以很容易地实现；可以简单地将上述功能准确编码，使用中心化服务器的硬盘来记录状态。然而，我们想把比特币构建去中心化货币系统，为了确保每个人都同意交易的顺序，我们需要将状态转换系统与一个共识系统结合起来。比特币的去中心化共识进程要求网络中的节点不断尝试将交易打包成“区块”。网络计划大约每十分钟产生一个区块，每个区块包含一个时间戳、一个随机数、一个对上一个区块的引用（即哈希）和上一区块生成以来发生的所有交易列表。随着时间推移就创建出了一个持续增长的区块链，它不断地更新，从而能够代表比特币账本的最新状态。

检查一个区块是否有效的算法，如以下范式所示：

1. 检查该区块引用的上一个区块是否存在且有效。
2. 检查该区块的时间戳是否大于上一个区块 ^[fn2]（注释编号）的时间戳并且在将来 2 小时以内
3. 检查区块上的工作量证明是否有效。
4. 令前一个区块末尾的状态为 $S[0]$ 。
5. 假设 TX 是该区块的交易列表，其中包含 n 个交易。对于 $0 \dots n-1$ 中的所有 i ，如果有任何应用程序返回错误，退出并返回 false，请设置 $S[i+1] = \text{APPLY}(S[i], \text{TX}[i])$ 。
6. 返回 true，并将 $S[n]$ 登记为该区块末尾的状态。

本质上，区块中的每笔交易都必须提供一个有效的状态转换，从交易执行前的规范状态转换到某个新状态。注意，状态并未编码到区块。它纯粹只是校验节点记住的抽象概念，只能被任意区块从创世状态开始，按顺序加上每一个区块的每一笔交易，（安全地）计算出当前状态。另外，需要注意矿工将交易收录进区

块的顺序。如果一个区块中有 A、B 两笔交易，B 花费的是 A 创建的 UTXO，如果 A 在 B 之前，这个区块是有效的，否则，这个区块无效。

“工作量证明”是出现在上表而其他系统没有的验证条件。具体验证方法为，对每个区块进行两次 SHA256 哈希处理，得到一个 256 位的数值，该数值必须小于不断动态调整的目标数值，本文写作时目标数值大约是 2^{187} 。工作量证明的目的是使创建区块有算力困难，从而阻止女巫攻击者恶意重新生成区块链。因为 SHA256 是完全不可预测的伪随机函数，创建有效区块的唯一方法就是简单地不断试错，不断地增加随机数的数值，查看新的哈希数是否小于目标值。

当前的目标数值是 $\sim 2^{187}$ ，网络必须平均尝试 $\sim 2^{69}$ 次才能生成有效的区块。一般而言，比特币网络每隔 2016 个区块重新设定目标数值，从而保证网络中的节点平均每十分钟生成一个区块。为了对矿工的计算工作进行激励，每一个成功生成区块的矿工有权在区块中包含一笔凭空发给他们自己 12.5 BTC 的交易。另外，如果交易的输入额大于输出，差额部分就作为“交易费”付给矿工。顺便提一下，这也是比特币发行的唯一机制，创世状态中并没有比特币。

为了更好地理解挖矿的目的，让我们分析比特币网络出现恶意攻击者时会发生什么。因为比特币的密码学基础是非常安全的，所以攻击者会选择攻击没有被密码学直接保护的部分：交易顺序。攻击者的策略非常简单：

1. 向商家发送 100 个比特币以换取某种产品（最好是快速交付的数字商品）
2. 等待商品交付
3. 创建另一笔交易，将这 100 个比特币发送给自己
4. 试图让网络相信他对自己的交易是先发生的。

一旦步骤 (1) 发生，几分钟后矿工将这笔交易收录到区块中，假设是编号为 270000 的区块。大约一小时后，此区块后面将会

有五个区块，每个区块间接地指向这笔交易，从而确认这笔交易。这时卖家收到货款，并向买家发货。因为我们假设这是数字商品，交付将瞬间完成。现在，攻击者创建另一笔交易，将相同的 100BTC 发送到自己的帐户。如果攻击者只是单纯地向全网广播这一消息，该笔交易不会被处理；矿工将运行状态转换函数 $\text{APPLY}(S, \text{TX})$ ，发现这笔交易要花费已经不在状态中的 UTXO。所以，攻击者会对区块链进行分叉，将第 269 个区块作为父区块重新生成第 270 个区块，在此区块中用新交易取代旧的。因为区块数据是不同的，这要求重新进行工作量证明。另外，攻击者的新版 270 区块有不同的哈希，原来的 271 到 275 的区块不指向它，所以原链和攻击者的新链是完全分离的。规定，在发生区块链分叉时，最长链被认为是诚实的区块链，合法的矿工将会沿着原有的 275 区块挖矿，只有攻击者一人在新的 270 区块后挖矿。攻击者为了使其区块链最长，他需要拥有比除了他以外的全网更多的算力来追赶（即“51%攻击”）。

默克尔树



左：仅提供默克尔树上的少量节点已经足够给出分支的合法证明。

右：对默克尔树任意部分进行改变的尝试最终都会导致链上某处

不一致。

比特币一个重要的可扩展特性是：它的区块存储在多层次数据结构中。一个区块的哈希实际上只是区块头的哈希，区块头是一段约 200 字节的数据，包含时间戳、随机数、上个区块的哈希和默克尔树根的哈希，而默克尔树是一个存储了该区块所有交易的数据结构。默克尔树是一种二叉树，由一组叶节点、一组中间节点和一个根节点构成。最下面是大量包含基础数据的叶节点，每个中间节点是其两个子节点的哈希，顶部的根节点也是其两个子节点的哈希。默克尔树的目的是允许区块数据可以零散地传送：节点可以从一个源下载区块头，从其它源下载相关树的一小部分，而依然能够确认所有的数据都是正确的。之所以如此是因为哈希向上传播：如果一个恶意用户尝试替换一个伪造的交易到树的底部，此改动将导致树的上层节点的改动，以及更上层节点的改动，最终导致根节点的改动以及区块哈希的改动，这样协议就会将其记录为一个完全不同的区块（几乎可以肯定是带着无效的工作量证明）。

默克尔树协议可以说是比特币长期持续性的基础。比特币网络中的一个全节点——存储和处理所有区块全部数据的节点，在 2014 年 4 月需要占用 15GB 的磁盘空间，而且还以每个月超过 1GB 的速度增长。目前，对台式计算机来说尚可接受，但是手机已经负载不了如此巨大的数据了，未来只有商业机构和爱好者才会充当完整节点。简化支付确认协议（SPV）允许另一种节点存在，这样的节点被称为“轻节点”，它下载区块头，使用区块头确认工作量证明，然后只下载与其交易相关的默克尔树分支。这使得轻节点只要下载整个区块链的一小部分，就可以安全地确定任何一笔比特币交易的状态和帐户的当前余额。

其它的区块链应用

将区块链思想应用到其它领域的想法早就出现了。2005 年，Nick Szabo 提出了“[利用所有者权限确保财产权](#)”这一概念，该文件描述了“复制数据库技术的新进展”将如何允许基于区块链的系统存储谁拥有哪些土地的登记表，创建了一个包括宅基地、违法占有和佐治亚州土地税等概念的复杂框架。然而，不幸的

是在那时还没有实用的复制数据库系统，所以这个协议没有被付诸实践。不过，自 2009 年比特币的去中心化共识开发成功以来，大量区块链的其它应用开始快速出现。

- **域名币** - 创建于 2010 年，[域名币](#) 描述成去中心化的名称注册数据库最为恰当。在 Tor、比特币和比特信等去中心化协议中，需要某种方式来识别帐户，以便其他人可以与帐户交互，但在所有现有解决方案中，唯一可用的标识符是伪随机哈希，如 1LW79wp5ZBqaHW1jL5TCiBCrhQYtHagUWy。理想情况下，人们希望能够拥有名称的帐户，比如 “george”。但是，问题在于如果一个人可以创建名为 “george” 的帐户，那么其他人也可以按相同流程为自己注册 “george” 来冒充。唯一的解决方案是 “成果优先原则” 范式，即第一个注册者成功后第二个注册者将失败，这个问题非常适合比特币共识协议。域名币是应用这种想法的最早、最成功的名称注册系统实现。
- **彩色币** - [彩色币](#) 的作用是充当一种协议，让人们在比特币区块链上创建自己的数字货币，或者在货币只有一个单位的这种重要但琐碎情况下，创建数字代币。在彩色币协议中，通过公开为特定的比特币 UTXO 分配一种颜色来 “发行” 新货币，并且该协议以递归方式将其他 UTXO 的颜色定义为与创建它们的交易所花费的输入的颜色相同（一些特殊规则适用于混合颜色输入的情况）。这样，用户可以维护仅包含特定颜色 UTXO 的钱包，像发送普通比特币一样发送它们，并通过区块链回溯以确定他们收到的任何 UTXO 的颜色。
- **元币** - 元币是想要拥有一个基于比特币的协议，使用比特币交易来存储元币交易，但具有不同的状态转换函数 `APPLY'`。因为元币协议无法阻止无效元币交易出现在比特币区块链中，所以增加了一条规则，如果 `APPLY'(S, TX)` 返回错误，该协议默认为 `APPLY'(S, TX) = S`。这为创建任意加密货币协议提供了一种简单的机制，可能有无法在比特币内部实现的高级功能，但开发成本非常低，因为比特币协议已经处理了挖矿和网络的复杂性。元币已被用于实现某些类别的金融合约、名称注册和去中心化交易所。

因此，一般而言，建立共识协议有两种方法：建立一个独立网络

或把协议建立在比特币网络上。前一种方法在域名币这样的应用中相当成功，但是该方法的实施非常困难，每个应用都要创建独立的区块链，建立并测试所有必须的状态转换函数和网络代码。另外，我们预测去中心化共识技术应用将会服从幂律分布，大多数的应用太小不足以保证自身的安全，我们还注意到大量的去中心化应用，尤其是去中心化自治组织，需要进行应用之间的交互。

另一方面，基于比特币的方法存在缺点，它没有继承比特币简化确认支付（SPV）的特性。比特币可以实现简化确认支付，因为比特币可以用区块链深度代表有效性；某种程度上，当一笔交易的祖先们距离现在足够远时，就可以安全地认为它们是合法状态的一部分。与之相反，基于比特币区块链的元币协议不能强迫区块链剔除违反元币协议的交易。因此，完全安全的元币协议的简化支付确认需要后向扫描所有的区块，直到比特币区块链的初始点，以确认某一交易是否有效。目前，所有基于比特币的元币协议的“轻”实施都依赖可信任的服务器提供数据，这对主要目的之一是消除信任需要的加密货币而言，可能是一个相当次优的结果。

脚本

即使不对比特币协议进行扩展，它也能在一定程度上实现“智能合约”。比特币的 UTXO 并非只能被公钥拥有，也可以被用基于堆栈的编程语言所编写的更加复杂的脚本所拥有。在这一模式下，花费这样的 UTXO，必须提供满足脚本的数据。事实上，甚至基本的公钥所有权机制也是通过脚本实现的：脚本将椭圆曲线签名作为输入，验证该交易和拥有此 UTXO 的地址，如果验证成功则返回 1，否则返回 0。其它更复杂的脚本用于各种不同的应用情况。例如，人们可以创建要求集齐三个私钥签名中的两个才能进行交易确认的脚本（多重签名），对公司帐户、安全储蓄帐户和某些商业代理来说，这种脚本是非常有用的。脚本也能用来支付解决计算问题的奖励，人们甚至可以创建这样的脚本“如果你能够提供你已经发送一定数额的狗币给我的简化确认支付证明，该比特币 UTXO 就是你的了”，本质上，比特币系统允许不同的加密货币进行去中心化交易。

然而，比特币系统的脚本语言存在一些严重的限制：

- **缺乏图灵完备性** - 也就是说，虽然比特币脚本语言支持一个很大的计算子集，但它基本上不支持所有计算。缺少的主要类别是循环。这样做是为了避免交易验证期间出现无限循环；理论上，对脚本程序员来说循环是一个可以克服的障碍，因为任何循环都可以通过简单地使用 if 语句多次重复执行底层代码来模拟，但这确实会导致脚本的空间效率非常低下。例如，实现另一种椭圆曲线签名算法可能需要 256 次重复的乘法，而每次都需要单独写在代码里。
- **价值盲** - UTXO 脚本无法对可提取金额进行精细控制。例如，预言机合约的一个强有力的用例是对冲合约，其中 A 和 B 存入价值 \$1000 的比特币，30 天后脚本将价值 \$1000 的比特币发送给 A，其余的发送给 B。这需要预言机来确定 1 个比特币的美元价值，但即便如此，与现有完全集中化的解决方案相比，这在信任和基础设施要求方面仍是一个巨大的进步。然而，由于 UTXO 要么是全部要么是零，要实现这一目标，只能使用非常低效的破解方法，即持有许多不同面额的 UTXO（例如，面额为 2^k 的 UTXO，每个 k 值都可以达到 30）并让预言机选择发送给 A 和发送给 B 的 UTXO。
- **缺少状态** - UTXO 可以是已使用或未使用；用于保存任何其他内部状态的多阶段合约或脚本是没有机会出现的。这使得多阶段期权合约、去中心化交易报价或两阶段加密承诺协议（这是安全计算赏金所必需的）难以创建。这也意味着 UTXO 只能用于构建简单的一次性合约，而不是去中心化组织等更复杂的“有状态”合约，使得元协议难以实现。二进制状态加之价值盲也意味着另一个重要应用 — 提款限制 — 是不可能实现的。
- **区块链盲** - UTXO 看不到区块链的数据，例如随机数、时间戳和上一个区块的哈希。由于该脚本语言无法通过随机性来创造可能的价值，它在博彩和其他几个类别的应用受到了严重限制。

至此，我们已经考察了在加密货币上建立高级应用的三种方法：建立一个新的区块链、在比特币区块链上使用脚本、在比特币区块链上建立元币协议。建立新区块链的方法可以自由地实现任

意的特性，但要付出开发时间、引导工作和安全性的代价。使用脚本的方法容易实施和标准化，但是它的功能有限。元币协议尽管非常容易实现，但是存在扩展性差的缺陷。在以太坊系统中，我们打算建立一个替代框架，使得开发更便捷、轻客户端性能更强大，同时允许应用程序共享经济环境和区块链安全性。

以太坊

以太坊的目的是创建一个用于建立去中心化应用的替代协议，我们认为提供一套不同的折衷方案对大量去中心化应用非常有用，尤其是那些强调快速开发、小型和不常用应用的安全性，以及应用间高效交互能力的程序。以太坊通过构建本质上是最终的抽象基础层来实现这一点：一种内置图灵完备编程语言的区块链，允许任何人编写智能合约和去中心化应用，并在其中设立他们自由定义的所有权规则、交易方式和状态转换函数。域名币的主体框架只需要两行代码就可以实现，诸如货币和信誉系统等其它协议只需要不到二十行代码就可以实现。智能合约，即包含价值、只有在满足特定条件时才能解锁的加密“盒子”，也可以在平台上构建，并且因为图灵完备性、价值知晓（value-awareness）、区块链知晓（blockchain-awareness）和多状态所增加的力量，而比比特币脚本所能提供的智能合约强大得多。

以太坊帐户

在以太坊中，状态由称为“帐户”的对象组成，而每个帐户都有一个 20 字节的地址，状态转换是指帐户之间价值和信息的直接转移。一个以太坊帐户包含四个字段：

- **nonce**，用于确保每笔交易只能处理一次的计数器
- 帐户当前的**以太币余额**
- 帐户的**合约代码**（若有）
- 帐户的**存储**（默认为空）

以太坊是以太坊内部的主要加密燃料，用于支付交易费。通常有两类帐户：由私钥控制的**外部帐户**以及由其合约代码控制的**合约帐户**。外部帐户没有代码，持有者可以通过创建和签署交易从外部帐户发送消息；在合约帐户中，每次合约帐户收到消息时，其代码都会激活，允许该帐户读取和写入内部存储，继而发送其他消息或创建合约。

注意，以太坊中的“合约”不应被视为要“履行”或“遵守”的东西；相反，合约更像是存在于以太坊执行环境中的“自治代理”。当被交易或消息“触发”时，合约总是执行特定的代码段，并直接控制自己的以太坊余额和键/值存储，以跟踪永久变量。

消息和交易

在以太坊中，术语“交易”用来指代已签名的数据包，数据包存储着将从外部帐户发送的消息。交易包含：

- 消息接收者
- 用于识别发送者身份的签名
- 从发送者转账到接收者的以太坊金额
- 一个可选数据字段
- STARTGAS 值，表示允许交易运行的最大计算步骤数
- GASPRICE 值，表示发送者每个计算步骤支付的费用

前三个是任何加密货币都有的标准字段。默认情况下，数据字段没有函数，但虚拟机有一个操作码，合约可以使用该操作码访问数据；以这样的用例为例：如果一个合约作为区块链上的域名注册服务，那么它可能希望将传送给它的数据解释为包含两个“字段”，第一个字段是要注册的域名，第二个字段将域名注册到 IP 地址。合约将从消息数据中读取这些值，并将其适当地存储。

STARTGAS 及 GASPRICE 字段对于以太坊的反拒绝服务模型至

关重要。为了防止代码中出现无意或恶意的无限循环或其他计算浪费，要求每笔交易对代码可以执行的计算步骤设置一个限制。计算的基本单位是燃料；通常，一个计算步骤消耗 1 份燃料，但某些操作会消耗更多燃料，因为它们在计算上更加昂贵或者增加了必须存储到状态中的数据量。交易数据中的每个字节还需支付的费用为 5 份燃料。收费系统的意图是要求攻击者相应支付他们消耗的每一种资源，包括计算、带宽和存储；因此，任何导致网络消耗更多这些资源的交易，都必须支付大致与增加量成比例的燃料费用。

消息

合约能够向其他合约发送“消息”。消息是从未序列化的虚拟对象，只存在于以太坊执行环境中。消息包含：

- 消息发送者（隐含的）
- 消息接收者
- 随消息一起转账的以太币金额
- 一个可选数据字段
- STARTGAS 值

本质上消息类似于交易，只是消息是由合约而非外部参与者产生的。当前正在运行代码的合约执行 CALL 操作码时会产生一条消息，该操作码就是用于产生并执行消息。像交易一样，信息导致接收者帐户运行其代码。因此，合约之间可以建立关系，方式完全与外部参与者之间建立关系相同。

请注意，为交易或合约分配的燃料配额适用于该交易和所有子执行消耗的总燃料量。例如，如果外部参与者 A 向 B 发送一笔配额为 1000 份燃料的交易，B 在向 C 发送消息需要消耗 600 份燃料，而 C 在内部执行需要消耗 300 份燃料才能返回结果，那么 B 再发送 100 份燃料就会消耗完燃料。

以太坊状态转换函数



以太坊状态转换函数 $\text{APPLY}(S, \text{TX}) \rightarrow S'$ 可如下定义：

1. 检查交易格式是否正确（即具有正确数量的值）、签名是否有效以及 Nonce 值是否与发送者帐户中的 Nonce 值匹配。若否，则返回错误。
2. 通过 $\text{STARTGAS} * \text{GASPRICE}$ 计算出交易费，并从签名中确定发送地址。从发送者的帐户余额中减去费用，并增加发送者的 nonce 值。如果帐户余额不足，则返回错误。
3. 初始化 $\text{GAS} = \text{STARTGAS}$ ，并根据交易中的字节数量为每个字节扣除相应数量的燃料。
4. 将交易数值从发送者帐户转移至接收帐户。如果接收帐户尚不存在，则创建此帐户。如果接收帐户是合约，运行该合约的代码，直到代码运行结束或燃料耗尽。
5. 如果由于发送者资金不足或者代码运行耗尽了燃料，而导致转账失败，则回滚除支付费用之外的所有状态变化，并将费用支付给矿工帐户。
6. 否则，将所有剩余燃料的费用退还发送者，并把为所消耗燃料而支付的费用发送给矿工。

例如，假设合约的代码如下：

```
if !self.storage[calldataload(0)]:  
    self.storage[calldataload(0)] =  
    calldataload(32)
```

注意，合约代码实际上是用低级以太坊虚拟机代码编写的；为了清晰起见，此示例是用我们的一种高级语言 Serpent 编写的，它可以编译为以太坊虚拟机代码。假设合约的存储一开始是空的，发送了一个价值为 10 个以太币的交易，消耗 2000 份燃料，燃料价格为 0.001 个以太币，并且数据包含 64 个字节，字节 0-31 代表数字 2，字节 32-63 代表字符串 CHARLIE。在这种情况下，状态转换函数的执行过程如下：

1. 检查交易是否有效、格式是否正确。
2. 检查交易发送者是否至少有 $2000 * 0.001 = 2$ 个以太币。若有，则从发送者帐户中扣除 2 个以太币。
3. 初始化燃料 = 2000 份，假设交易长度为 170 个字节，每字节费用 5 份燃料，减去 850 份燃料，剩下 1150 份燃料。
4. 从发送者帐户再减去 10 个以太币并增加到合约帐户。
5. 运行代码。在本例中，运行比较简单：代码检查是否使用合约的索引 2 处的存储，若未使用，则通知；若使用，代码将索引 2 处的存储设置为值 CHARLIE。假设该运行花费了 187 份燃料，所以余下的燃料数量是 $1150 - 187 = 963$ 份燃料。
6. 向发送者帐户增加 $963 * 0.001 = 0.963$ 个以太币，同时返回产生的状态。

如果交易的接收一端没有合约，那么总交易费就等于提供的 GASPRICE 乘以交易的字节长度，并且和随交易发送的数据无关。

注意，消息在回滚方面与交易相同：如果消息执行耗尽燃料，那么该消息的执行以及该执行触发的所有其他执行都会回滚，但父执行不需要回滚。这意味着合约调用另一份合约是“安全的”，就

好像 A 使用 G 份燃料调用 B，那么可以保证 A 的执行最多损耗 G 份燃料。最后请注意，有一个创建合约的操作码 CREATE；它的执行机制通常类似于 CALL，不同之处在于执行的输出决定了新创建合约的代码。

代码执行

以太坊合约中的代码用一种基于堆栈的低级字节码语言编写，被称为“以太坊虚拟机代码”或“EVM 代码”。该代码由一系列字节组成，每个字节代表一种操作。通常，代码执行是一个无限循环，即重复执行当前程序计数器（从零开始）处的操作，然后将程序计数器增加一，直到代码执行完毕或出现错误，或者检测到 STOP 或 RETURN 指令。操作可以访问三种数据存储空间：

- **堆栈**，一种后进先出容器，值可以在其中入栈和出栈
- **内存**，一种可无限扩展的字节数组
- **合约的长期存储**，一个键/值存储。与堆栈和内存会在计算结束后重置不同，存储将长期持续存在。

代码可以访问传入消息的值、发送者信息和数据，可以访问区块头数据，而且代码还可以返回数据字节数组作为输出。

以太坊虚拟机码的正式执行模型简单得令人吃惊。当以太坊虚拟机运行时，其完整计算状态可以由元组 (block_state, transaction, message, code, memory, stack, pc, gas) 来定义，其中 block_state 是包含所有帐户的全局状态并包括余额和存储。在每一轮执行开始时，可以通过调用 code 的第 pc 个字节（或者如果 pc >= len(code)，则调用 0）来找到当前指令，并且每条指令在元组影响方式方面都有自己的定义。例如，ADD 将两个项目出栈并将它们的和入栈，将 gas 减少 1 并将 pc 增加 1，SSTORE 将顶部的两个项目出栈并将第二个项目插入到合约存储中第一个项目指定的索引处。尽管有很多通过 JIT 编译来优化以太坊虚拟机执行的方法，但只需几百行代码就可以完成以太坊的基本实现。

区块链和挖矿



以太坊区块链在许多方面与比特币区块链相似，但确实存在一些差异。以太坊和比特币在区块链架构方面的主要区别在于，与比特币不同，以太坊区块包含交易列表和最新状态的副本。此外，其他两个值、区块编号和难度也存储在区块中。以太坊中的基本区块验证算法如下：

1. 检查被引用的前一个区块是否存在并有效。
2. 检查区块的时间戳是否大于被引用的前一个区块的时间戳，并且在将来 15 分钟以内。
3. 检查区块编号、难度、交易根、叔根和燃料限制（各种以太坊特定的低级概念）是否有效。
4. 检查区块上的工作量证明是否有效。
5. 令前一个区块末尾的状态为 $S[0]$ 。
6. 令区块的交易列表为 TX，并包含 n 笔交易。对于 $0 \dots n-1$ 中的所有 i ，设置 $S[i+1] = \text{APPLY}(S[i], \text{TX}[i])$ 。如果任何应用程序返回错误，或者直到此时区块中消耗的总燃料量超过 GASLIMIT，则返回错误。
7. 令 S_{FINAL} 为 $S[n]$ ，但添加支付给矿工的区块奖励。
8. 检查状态 S_{FINAL} 的默克尔树根是否等于区块头中提供的最终状态根。如果等于，则该区块有效；否则该区块无效。

这种方法乍一看效率似乎极低，因为它需要存储每个区块的全部状态，但实际上效率应该与比特币相当。原因是状态存储在树结构中，而且在添加每个区块后只需要更改树的一小部分。因此一般来说，在两个相邻区块之间，树的绝大部分应该是相同的，因此数据可以用指针（即子树的哈希）存储一次和引用两

次。一种称为“帕特里夏树”的特殊类型的树用于实现此目的，它包括对默克尔树概念的修改，允许高效地插入和删除节点，而不仅仅是更改。此外，由于所有状态信息都存在于最后一个区块内，因此无需存储整个区块链历史，如果可以应用于比特币，使用这种策略计算可以节省 5-20 倍空间。

一个常见的问题是合约代码在物理硬件的“哪里”执行。该问题有一个简单的答案：合约代码的执行过程是状态转换函数定义的一部分，而该函数是区块验证算法的一部分，因此如果将交易添加到区块 B 中，由该交易产生的代码执行将在现在和将来由所有节点执行，由此下载并验证区块 B。

应用

通常，以太坊上有三种类型的应用。第一类是金融应用，为用户提供更有效的方式来使用资金管理和签订合约。包括子货币、金融衍生品、对冲合约、储蓄钱包、遗嘱，甚至最终包括某些类别的完整雇佣合约。第二类是半金融应用，它们涉及金钱，但很大一部分功能也与资金无关；一个恰当的示例是针对解决计算难题的自动执行的赏金。最后还有一些应用与金融毫不相关，例如在线投票和去中心化治理。

代币系统

区块链上的代币系统有许多应用，从代表美元或黄金等资产的子货币到公司股票等等，单个代币可以代表智能资产、不可伪造的安全优惠券，甚至可代表作为激励积分系统并与传统价值完全没有联系的代币系统。代币系统在以太坊中非常容易实现，让人吃惊。要理解的重点是，从根本上讲，所有货币或代币系统都是具有这样一种操作的数据库：从 A 中减去 X 个单位并将 X 个单位添加给 B，条件是 (1) A 在交易之前至少有 X 个单位并且 (2) 交易由 A 批准。实现代币系统所需要做的就是将此逻辑实现到合约中。

使用 Serpent 语言实现代币系统的基本代码如下所示：

```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] =
self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] +
value
```

此代码本质上是本文档前面部分详细描述“银行系统”状态转换函数的字面实现。需要额外添加几行代码来规定在最初以及其他一些特殊情况下分配货币单位的初始步骤，理想情况下，应该添加一个函数让其他合约查询地址的余额。但这就足够了。理论上，基于以太坊的代币系统在作为子货币时可能具有另一个重要特征，该特征是基于比特币的链上元货币所缺乏的，那就是直接以该货币支付交易费的能力。实现这一点的方式是：合约会保持一定数量的以太币余额，用来向发送者退还用于支付费用的以太币；合约也会通过收取费用来收集内部货币，并在持续不断的拍卖中转售货币，以此补充以太币余额。因此，用户需要用以太币“激活”他们的帐户，但一旦帐户中有以太币，就可以重复使用，因为合约每次都会向帐户退还资金。

金融衍生品和价值稳定的货币

金融衍生品是“智能合约”最常见的应用，也是通过代码实现的最简单的应用之一。实现金融合约的主要挑战在于，其中大多数合约都需要引用外部价格自动收报机；例如，一个非常理想的应用是对冲以太币（或其他加密货币）相对于美元波动的智能合约，但对冲需要合约知道以太币/美元的价值。要实现这一点，最简单的方法是借助由特定方（例如纳斯达克）维护的“数据馈送”合约，这种合约的设计使得特定方能够根据需要更新合约并提供一个接口，允许其他合约向该合约发送消息并返回包含价格的响应。

鉴于这一关键因素，对冲合约将如下所示：

1. 等待 A 方输入 1000 个以太币。
2. 等待 B 方输入 1000 个以太币。
3. 在存储中记录 1000 个以太币的美元价值（通过查询数据馈送合约计算得出），假设价值是 \$x。
4. 30 天后，允许 A 或 B“重新激活”该合约，以便将价值 \$x 的以太币（通过再次查询数据馈送合约获取新价格并且计算得出）发送给 A，剩余以太币发送给 B。

这种合约在加密货币交易中潜力巨大。加密货币的主要问题之一是它的波动性。尽管许多用户和商家可能希望获得处理加密资产的安全性和便利性，但他们中许多人不希望面临在一天内资金价值损失 23% 的景象。迄今为止，最常见的解决方案是发行人支持的资产；其想法是发行人创建一种子货币，他们有权发行和撤销这种子货币单位，并且发行人可以向给他们（离线）提供一个单位指定基础资产（例如黄金、美元）的任何人提供一个单位的货币。然后，发行人承诺向返还一个单位加密资产的任何人提供一个单位基础资产。这种机制使得任何非加密资产“升级”为加密资产，前提是发行人是可信的。

但实际上，发行人并不总是值得信赖，在某些情况下，银行基础设施过于薄弱或过于不友好，以至于无法提供此类服务。金融衍生品提供了一种替代方案。在这种方案中，不是由单个发行人提供资金来支持资产，而是由一个去中心化的投机者市场承担了这一角色，他们押注加密参考资产（例如以太币）的价格会上涨。与发行人不同，投机者无法在交易中违约，因为对冲合约托管他们的资金。请注意，这种方法不是完全去中心化的，因为仍然需要一个可信来源提供价格自动收报机，但可以说在降低基础设施要求（与成为发行者不同，发布价格馈送不需要许可证并且可能被归类为自由言论）以及减少欺诈的可能性方面，这仍是一次巨大的改进。

身份和信誉系统

最早的替代加密货币[域名币](#) [↗](#) 尝试使用类似比特币的区块链提供一种名称注册系统，通过该系统，用户可以在公共数据库中注册他们的姓名和其他数据。主要用例是 [DNS](#) [↗](#) 系统，它将诸如“bitcoin.org”等域名（在域名币的情况下，“bitcoin.bit”）映射到一个 IP 地址。其它用例包括电子邮件身份验证系统和可能更为先进的信誉系统。下面是一个基础合约，它在以太坊中提供与域名币类似的名称注册系统：

```
def register(name, value):  
    if !self.storage[name]:  
        self.storage[name] = value
```

该合约非常简单；它完全是以太坊网络中的一个数据库，可以向其中添加但不能修改或删除。任何人都可以把名称注册为一个值，该注册将永久保存。更复杂的名称注册合约还包含一个“函数子句”以及一种机制，前者允许其他合约查询它，后者允许名称的“所有者”（即第一个注册者）更改数据或转让所有权。甚至可以在该合约上添加信誉和信任网络功能。

去中心化文件存储

过去几年，大批受欢迎的在线文件存储初创公司不断涌现，其中最著名的是 Dropbox。Dropbox 想让用户可以上传硬盘备份、提供备份存储服务并允许用户访问备份，而用户需按月付费。然而，在这一点上，文件存储市场有时效率相对较低。在粗略了解各种现有解决方案后会发现，主流文件存储的每月价格比整个硬盘驱动器的成本还要高，特别是在被称为“恐怖谷”的 20-200 GB 级别，既没有免费额度也没有企业级折扣。以太坊合约让去中心化文件存储生态系统得以发展，个人用户可以在该系统中将自己的硬盘租出去以获得少量收益，而未使用的空间可用来进一步降低文件存储的成本。

该系统的基础性构件就是我们所谓的“去中心化 Dropbox 合约”。该合约的工作原理如下。首先，用户将需要存储的数据拆

分成几个区块并对每个区块加密以保护隐私，然后以此构建一个默克尔树。然后创建一个含以下规则的合约，对于每 N 个区块，合约将从默克尔树中选择一个随机索引（使用能够被合约代码访问的上一个区块的哈希作为随机性来源），然后给予第一个实体 X 个以太币，以提供具有简化支付确认（例如证明树中特定索引处区块的所有权）的交易。当用户想重新下载他们的文件时，可以使用微支付通道协议（例如每 32 KB 支付 1 个 szabo）收回文件；最节省费用的方法是支付者不到最后不发布交易，而是每 32 KB 之后，用一个更划算的具有相同 nonce 的交易取代原来的交易。

该协议的一个重要特点是，虽然似乎用户相信许多随机节点不会丢失文件，但可以通过以下方法将这种风险降低到接近于零：通过私钥共享将文件拆分成许多部分，并通过监控合约确定每一部分仍在某个节点中。如果合约仍在支付款项，则提供了一个加密证明，证明有人仍在存储该文件。

去中心化自治组织

通常意义上“去中心化自治组织”是指拥有一定数量成员或股东的虚拟实体，他们大概拥有 67% 的大多数股权，有权使用实体的资金并修改其代码。成员集体决定组织的资金分配方式。去中心化自治组织的资金分配方式可以是奖金、薪资或者更奇特的机制等等，比如用内部货币去奖励工作。这在本质上复制了传统公司或者非营利组织的合法手段，但仅使用加密区块链技术进行了加强。目前为止，许多关于去中心化自治组织的讨论都围绕着去中心化自治公司的“资本家”模式，其中有可获得红利的股东和可交易的股份；作为替代方案，有一种可能被称为“去中心化自治社区”的实体将使所有成员在决策时拥有同等权利，并在增减成员时要求 67% 的现有成员多数同意。由于每个人只能拥有一个成员资格，所以需要群体来集体执行。

下面概括了如何用代码实现去中心化自治组织。最简单的设计就是一段自动修改的代码，如果三分之二的成员同意更改，该代码就更改。理论上代码是不可更改的，然而通过把代码片段放入不同的合约并将合约调用的地址存储在可更改的存储中，用户

可以轻易解决这一问题，使代码事实上变得可修改。在这种去中心化自治组织合约的简单实现中，有三种交易类型，可通过交易中提供的数据行区分：

- $[0, i, K, V]$ 在索引 i 处注册提案，以便将存储索引 K 的地址更改为值 V
- $[1, i]$ 注册一张赞成提案 i 的投票
- $[2, i]$ 如果投票有足够票数，则确认提案 i

合约为每一种交易都提供有子句。它将维护所有开放存储更改的记录以及投票支持者的列表。合约还包括所有成员的列表。当任何存储更改获得三分之二成员投票赞成时，一个确认交易将执行这项更改。更复杂的框架可能还有针对发送交易、增减成员等功能的内置投票功能，甚至可以提供[委任式民主](#) [投票委托](#)（即任何人都可以委托另外一个人代表自己投票，而且这种委托关系是可以传递的，如果 A 委托了 B，然后 B 委托了 C，那么 C 将决定 A 的投票）。这种设计将使去中心化自治组织作为一个去中心化社区有机地成长，允许人们最终将筛选成员的任务委派给专家，但与“现有系统”不同，随着时间的推移，当个别社区成员改变他们的阵营时，专家可以很容易地加入或退出。

另一个模型是去中心化公司，其中任何帐户可以拥有零份或多份股份，决策需要持有三分之二多数股份。完整框架将包括资产管理功能，即能够出价购买或出售股份并且能够接受报价（最好是合约里有订单匹配机制）。委托还提供委任制民主形式，普及了“董事会”的概念。

更多应用

1. 储蓄钱包。 假设 Alice 想安全地保管她的资金，但她担心自己的私钥丢失或被破解。她把以太币放到和银行 Bob 签订的一个合约里，如下所示：

- Alice 每天最多可以单独提取 1% 的资金。
- Bob 每天最多可以单独提取 1% 的资金，但 Alice 可以用她的

密钥创建一个交易取消 Bob 的提取权限。

- Alice 和 Bob 一起可以任意提取资金。

通常，每天 1% 的额度对于 Alice 足够了，如果 Alice 想提取更多资金，她可以联系 Bob 寻求帮助。如果 Alice 的密钥被破解，她可以立即找到 Bob，帮她将资金转移到一个新合约里。如果 Alice 丢失了密钥，Bob 最终会取出资金。如果最终发现 Bob 是恶意的，她可以取消他的提取权限。


2. 作物保险。 用户可以轻松地制订金融衍生品合约，但使用的是天气而不是任何价格指数的数据馈送。如果爱荷华州的一位农民购买了一项金融衍生品，该产品基于爱荷华的降雨情况进行反向赔付，那么如果遇到干旱，该农民将自动收到赔付资金，而且如果降雨充沛，他会很开心，因为他的作物收成会很好。通常，这种保险可以扩展到自然灾害保险。


3. 去中心化数据馈送。 对于金融差价合约，实际上有可能通过一种名为“[谢林币](#) [🔗](#)”的协议将数据馈送去中心化。谢林币的基本工作原理如下。N 个相关方都向系统输入给定数据的值（以太币/美元价格），对这些值进行排序，在第 25 和第 75 百分位之间的每个人都会得到一个代币作为奖励。每个人都有动力提供其他人都会提供的答案，而唯一能让众多参与者实际达成一致的值是显而易见的：真相。这样就创建了一种去中心化的协议，它理论上可以提供任何数量的值，包括以太币/美元的价格、柏林的温度、甚至某个硬计算的结果。

4. 智能多重签名托管。 比特币允许多重签名交易合约，例如，提供了给定五个密钥中的三个便可以使用资金。以太坊允许更精细的控制；例如，提供五个密钥中的四个可以使用任意数额的资金，提供五个密钥中的三个可以每天最多使用 10% 的资金，提供五个密钥中的两个可以每天最多使用 0.5% 的资金。此外，以太坊的多重签名是异步的 — 双方可以在不同时间在区块链上注册他们的签名，最后一个签名将自动发送交易。

5. 云计算。 以太坊虚拟机技术还可以用来创建一个可验证的计

算环境，让用户可以要求他人执行计算，然后有选择地索要证明，证实计算在某些随机选定的检查点处正确完成。这就可以创建一个云计算市场，任何用户都可以用他们的台式机、笔记本电脑或专用服务器来参与，并且抽查与保证金双管齐下确保系统是值得信赖的（即节点不能通过欺骗获利）。但是，这样的系统可能并不适合所有任务；例如，需要进行大量进程间通信的任务无法在大型节点云上轻易实现。然而，其他任务则更容易实现并行；例如 SETI@home、folding@home 和遗传算法等项目可以方便地在这类平台上实现。


6. 点对点赌博。 任意数量的点对点赌博协议都可以在以太坊区块链上实现，例如 Frank Stajano 和 Richard Clayton 的 [Cyberdice](#) 。最简单的赌博协议实际上只是一种关于下一个区块哈希的差价合约，并且可以在其基础上创建更高级的协议，创建接近零费用且无法作弊的赌博服务。

7. 预测市场。 如果有预言机或谢林币，预测市场也很容易实现，预测市场与谢林币一起有可能被证明是 [futarchy](#)  的第一个主流应用，作为去中心化组织的治理协议。

8. 链上去中心化市场，基于身份和信誉系统。

杂项和关注

改进版 GHOST 协议的实现

“贪婪最重可观察子树”(GHOST) 协议是由 Yonatan Sompolinsky 和 Aviv Zohar 在 [2013 年 12 月](#)  首次提出的一项创新。提出 GHOST 的动机是，具有快速确认时间的区块链目前由于过时率高而安全性降低 — 因为区块需要一定的时间才能通过网络传播，如果矿工 A 开采了一个区块，然后矿工 B 碰巧在矿工 A 的区块传播到 B 之前开采了另一个区块，那么矿工 B 的区块最终会被作废，不会增加网络安全。此外，还有一个中心化问题：如果矿工 A 是一个拥有 30% 算力的矿池，而 B 拥

有 10% 算力，那么 A 将面临 70% 的时间生产陈腐区块的风险（因为在其他 30% 的时间 A 产生了最后一个区块，所以会立即获得挖矿数据），而 B 将面临 90% 的时间生产陈腐区块的风险。因此，如果区块间隔短到足以使过时率较高，则 A 将仅仅凭借其规模而显着提高效率。结合这两种影响，快速产生区块的区块链很可能造就一个拥有足够高比例网络算力的矿池，从而对挖矿过程拥有事实上的控制。

正如 Sompolinsky 和 Zohar 所描述的，GHOST 通过在计算哪条链“最长”时包含陈腐区块来解决第一个问题 - 网络安全降低；也就是说，在计算哪个区块具有最大的总工作量证明支持它时，不仅区块的父块和更远的祖先，而且该区块祖先（在以太坊行话中称为“叔块”）的陈腐子代也都被添加到计算中。为了解决第二个问题 - 中心化偏差，我们跳出了 Sompolinsky 和 Zohar 描述的协议范畴，并且还陈腐区块提供区块奖励：陈腐区块获得其基础奖励的 87.5%，而包含陈腐区块的侄块获得剩余的 12.5%。不过，交易费不奖励给叔块。

以太坊实现了一个简化版的 GHOST 协议，它仅仅深入七个层级。具体而言，它的定义如下：

- 一个区块必须指定一个父块，并且必须指定零个或多个叔块
- 包含在区块 B 中的叔块必须具有以下属性：
 - 它必须是区块 B 的第 k 代祖先的直系子代，其中 $2 \leq k \leq 7$ 。
 - 它不能是 B 的祖先
 - 叔块必须是有效的区块头，但不需要是之前验证过的甚至是有效的区块
 - 叔块必须不同于前面区块中包含的所有叔块，并且不同于同一区块中包含的所有其他叔块（非双重包含）
- 对于区块 B 中的每个叔块 U，区块 B 的矿工获得额外 3.125% 的铸币奖励，而叔块 U 的矿工获得 93.75% 的标准铸币奖励。

这种限制版的 GHOST 协议，最多只能包含 7 代叔块，采用它有两个原因。首先，无限制 GHOST 协议让计算给定区块的哪些叔块有效时过于复杂。其次，无限制 GHOST 协议采用了以太坊中使用的补偿，取消了促使矿工在主链而不是公共攻击者的链上挖矿的激励措施。

费用

由于发布到区块链中的每笔交易都会给网络带来下载和验证成本，因此需要一些监管机制（通常涉及交易费）以防滥用。比特币中使用的默认方法是收取完全自愿性质的费用，依靠矿工充当守门人并设置动态最低费用。这种方法在比特币社区中非常受欢迎，特别是因为它是“基于市场的”，允许由矿工和交易发送者之间的供需决定价格。然而，这种思路的问题在于，交易处理并不符合市场规律。尽管将交易处理解释为矿工向发送者提供的服务直观上很有吸引力，但实际上矿工收录的每笔交易都需要由网络中的每个节点处理，因此绝大部分交易处理成本由第三方承担，而不是由决定是否收录交易的矿工承担。因此，公地悲剧的问题很可能发生。

然而结果却是，基于市场机制中的这个缺陷，在给出一个不准确的特定简化假设时，会神奇地自我抵消。论证如下。假设：

1. 交易导致 k 个操作，将提供奖励 kR 给收录它的任何矿工，其中 R 由发送者设置， k 和 R 事先（大体上）对矿工可见。
2. 操作在任何节点的处理成本均为 C （即所有节点效率相同）
3. 有 N 个挖矿节点，每个节点的处理能力完全相同（即为总处理能力的 $1/N$ ）
4. 没有不挖矿的完整节点。

如果预期奖励大于成本，矿工将愿意处理交易。因此，预期奖励是 kR/N ，因为矿工有 $1/N$ 几率处理下一个区块，而矿工的处理成本仅仅是 kC 。所以，当 $kR/N > kC$ 或者 $R > NC$ 时，矿工将会收录交易。请注意， R 是发送者提供的每个操作的费用，因此是发送者从交易中获得的收益的下限， NC 是整个网络共同

处理一个操作的成本。因此，矿工有动力仅收录那些总实际收益超过成本的交易。

然而，现实中这些假设会存在几个重要偏差：

1. 与其他验证节点相比，矿工处理交易的成本确实更高，因为额外的验证时间会延迟区块传播，因而增加区块变陈腐的几率。
2. 确实存在不挖矿的完整节点。
3. 实际中挖矿能力的分配最终可能极端不平等。
4. 热衷于破坏网络的投机者、政敌和疯子确实存在，他们可以巧妙地设置合约，使得他们的成本远低于其他验证节点支付的成本。

(1) 让矿工趋向于收录更少的交易，并且 (2) 增加 NC；因此，这两种作用会相互抵消一部分。[如何抵消?](#) (3) 和 (4) 是主要问题，为了解决它们，我们简单地制订了一个浮动上限：没有区块能够包含比 `BLK_LIMIT_FACTOR` 乘以长期指数移动平均值更多的操作数。具体如下：

```
blk.oplimit = floor(  
    (blk.parent.oplimit * (EMAFCTOR - 1) +  
    floor(parent.opcount *  
    BLK_LIMIT_FACTOR)) /  
    EMA_FACTOR  
)
```

`BLK_LIMIT_FACTOR` 和 `EMA_FACTOR` 是常量，暂时设置为 65536 和 1.5，但可能会在进一步分析后更改。

还有一个因素会抑制比特币中的大区块大小：大区块将需要更长时间来传播，因此变陈腐的概率更高。在以太坊中，燃料消耗量高的区块也可能需要更长的传播时间，因为它们的物理大小更

大，而且因为它们需要更长时间来处理交易状态转换以进行验证。这种延迟抑制因素在比特币中是一个重要的考虑因素，但在以太坊中由于 GHOST 协议而较少考虑；因此，依靠受监管的区块限制可提供更稳定的基线。

计算和图灵完备

重要的一点是，以太坊虚拟机是图灵完备的；这意味着以太坊虚拟机代码可以对任何设想可执行的计算进行编码，包括无限循环。以太坊虚拟机代码以两种方式实现循环。首先，使用一个 JUMP 指令，允许程序跳回至代码中的前一个位置，还使用一个 JUMPI 指令进行条件跳转，允许诸如 `while x < 27: x = x * 2` 之类的语句。其次，合约可以调用其他合约，有可能通过递归进行循环。这很自然地导致了一个问题：恶意用户能够通过迫使矿工和完整节点进入无限循环而不得不关机吗？这个问题的出现源于计算机科学中的一个难题，称为停机问题：在一般情况下，没有办法知道一个特定的程序是否会停止运行。

正如状态转换部分所述，我们的解决方案要求交易设置一个允许执行的最大计算步骤数，如果超过执行时间，计算就会被回滚，但仍要支付费用。消息的工作原理相同。为显示我们解决方案背后的动机，请看下面的示例：

- 攻击者创建一个运行无限循环的合约，然后向矿工发送激活该循环的交易。矿工将处理该交易，运行无限循环直到燃料耗尽。即使执行耗尽了燃料并中途停止，交易仍然有效，矿工仍然向攻击者索取每个计算步骤的费用。
- 攻击者创建一个非常长的无限循环，目的是迫使矿工持续计算很长时间，以至于计算结束时，将有更多区块产生出来，这样矿工就不可能通过收录该交易来索取费用。然而，攻击者需要为 STARTGAS 提交一个值，限制执行可以进行的计算步骤数，因此矿工将提前知道该计算将进行相当多的步骤数。
- 攻击者看到一个合约，其中的代码形式为 `send(A,contract.storage[A]);`

`contract.storage[A] = 0`，然后发送一个交易，但燃料只够运行第一步而不足以运行第二步（即进行提款但不让余额减少）。合约作者无需担心防卫此类攻击，因为如果执行中途停止，更改会被回滚。

- 金融合约使用九个专有数据馈送的中位数，以便最大限度降低风险。攻击者接管其中一个数据馈送，该数据馈送设计为可通过去中心化自治组织部分描述的变量-地址-调用机制修改，并将其转换为运行无限循环，从而强制任何从金融合约索取资金的尝试都因燃料耗尽而中止。然而，金融合约可以为消息设置一个燃料限制，防止这个问题发生。

图灵完备的替代方案是图灵不完备，其中 `JUMP` 和 `JUMPI` 不存在，并且在任何给定时间每个合约只允许有一个副本存在于调用堆栈内。在这样的系统里，上述收费系统和关于我们解决方案效果的不确定性可能都是不需要的，因为执行一个合约的成本将被它的大小决定。此外，图灵不完备甚至不是一个很大的限制；在我们内部构想的所有合约示例中，到目前为止只有一个需要循环，甚至那个循环也可以通过将一行代码重复 26 次来消除。考虑到图灵完备带来的严重影响和有限的益处，为什么不简单地使用一种图灵不完备语言呢？然而，在现实中，图灵不完备还远远不能有效地解决问题。要想知道原因，请思考以下合约：

```
C0: call(C1); call(C1);
C1: call(C2); call(C2);
C2: call(C3); call(C3);
...
C49: call(C50); call(C50);
C50: (run one step of a program and record
the change in storage)
```

现在，向 A 发送一笔交易。这样，在 51 笔交易中，我们有一个合约需要进行多达 2^{50} 个计算步骤。矿工可以尝试提前检测这种逻辑炸弹，方法是为每个合约维护一个值，指定合约可以进行的最大计算步骤数，然后对递归调用其他合约的合约进行计算，

但是这需要矿工禁止创建其他合约的合约（因为上面 26 个合约的创建和执行可以很容易地汇集到一个单独合约内）。另一个问题是，消息的地址字段是一个变量，所以在一般情况下，甚至不可能提前知道某个合约将调用哪些其他合约。于是，最终我们有了一个惊人的结论：图灵完备的管理惊人地容易，而在缺乏同样的控制时图灵不完备的管理惊人地困难，那为什么不直接让协议图灵完备呢？

货币和发行

以太坊网络包括自己的内置货币以太币，以太币扮演双重角色：提供一个主要流动资金层，实现各种数字资产之间的高效交易；更重要的是，提供一种支付交易费的机制。为了方便起见并避免将来出现争议（参考比特币当前的 mBTC、uBTC、satoshi 争论），不同面值的名称将提前设置如下：

- 1: wei
- 10^{12} : Szabo
- 10^{15} : finney
- 10^{18} : ETH

这应该被视为“美元”和“美分”或“BTC”和“satoshi”概念的扩展版本。在不久的将来，我们期望“ETH”用于普通交易，“finney”用于微型交易，“szabo”和“wei”可以在围绕费用和协议实现的技术讨论中使用；其余的面额可能会在以后变得有用，但目前不应包含在客户端中。

发行模型如下：

- 以太币将以货币销售的形式发行，价格为一个比特币可购买 1000-2000 个以太币，这种机制旨在为以太坊组织筹资和支付开发费用，且已被其他平台（如 Mastercoin 和 NXT）成功应用。早期的购买者将从较大的折扣中获益。发售所得的比特币将全部用来支付开发者的薪资和奖金，并用来投资以太坊和加密货币生态系统中的各种营利和非营利项目。

- 0.099 倍的发售总量（60102216 个以太币）将分配给以太坊组织，以补偿早期贡献者，并用以太币计价的方式支付创世块诞生前的开销。
- 0.099 倍的发售总量将作为长期储备金保留。
- 发售后，将永久性地每年为矿工分配 0.26 倍的发售总量。

分组	启动时	一年后	5 年后
货币单位	1.198X	1.458X	2.498X
购买者	83.5%	68.6%	40.0%
已支用的预售准备金	8.26%	6.79%	3.96%
已使用的售后准备金	8.26%	6.79%	3.96%
矿工	0%	17.8%	52.0%

长期供应增长率（百分比）



尽管采用了线性发行方式，然而和比特币一样，以太币的长期供

应增长率也趋于零。

上述模型提供了两个主要选项：(1) 捐赠池的存在和规模，以及 (2) 永久增长的线性供应的存在，而比特币采用了限制供应的方法。捐赠池存在的理由如下。如果捐赠池不存在，并且线性发行量减少到总发售量的 0.217 倍以实现相同的通货膨胀率，那么以太坊总量将减少 16.5%，而每个单位的价值将增加 19.8%。因此为了均衡，将会多发 19.8% 的以太坊，所以每个单位的价值将再次与以前完全一样。之后，该组织还将拥有 1.198 倍的比特币，可以考虑将其分成两部分：原有的比特币和增加的 0.198 倍比特币。因此，这种情况完全等同于捐赠，但有一个重要区别：该组织仅持有比特币，因而没有动力支持以太坊单位的价值。

永久性线性供应增长模型降低了有些人认为比特币财富过度集中的风险，并为生活在当前和未来的人提供了获取货币单位的公平机会，同时又保留了让人获取并持有以太坊的强效激励措施，因为长期来看，用百分比表示的“供应增长率”将趋于零。我们还推测，由于加密货币总是会因为不小心、死亡等原因而丢失，而加密货币的损失可以被模拟为每年总供应量的百分比，因此流通中的货币总供应量实际上最终会稳定在一个等于每年发行量除以损失率的数值上（例如，在损失率为 1% 时，一旦供应量达到 26 倍，那么每年将有 0.26 倍被开采，0.26 倍丢失，形成一个平衡点）。

注意，未来以太坊可能过渡到权益证明模型以确保安全，将每年发行量降低到 0 至 0.05 倍之间。如果以太坊组织失去资助或出于任何其他原因而消失，我们将开放一个“社区合约”：任何人都有权创建未来的以太坊候选版本，唯一的条件是太币数量必须最多为 $60102216 * (1.198 + 0.26 * n)$ 个，其中 n 是创世块产生后的年数。创建者可以自由地通过众筹或其他方式，分配权益证明驱动的供应增加与最大允许供应增加之间的部分或全部差额，以支付开发费用。不符合社区合约的候选版本升级可能被合理地分叉为兼容版本。

挖矿中心化

比特币挖矿算法的原理是，让矿工一次又一次地对区块头稍作修改的版本进行数百万次 SHA256 计算，直到最终某个节点所产生版本的哈希小于目标值（目前大约为 2^{192} ）。然而，这种挖矿算法容易遭受两种形式的中心化攻击。第一种，挖矿生态系统已经被 ASIC（专用集成电路）所支配，这些计算机芯片专门为特定的比特币挖矿任务而设计，因此效率提高了数千倍。这意味着比特币挖矿不再是一种高度去中心化和平等的事业，需要巨额资本才能有效参与。第二种，大部分比特币矿工事实上不在本地完成区块验证；而是依赖中心化矿池提供区块头。这个问题可以说更糟：截至撰写本文时，排名前三的矿池间接控制了比特币网络中大约 50% 的处理能力，尽管当矿池或联盟试图进行 51% 攻击时，矿工可以转换到其他矿池这一事实缓解了该问题。

以太坊现在的目的是使用一种挖掘算法，要求矿工从状态中获取随机数据，从区块链的最后 N 个区块中计算一些随机选择的交易，并返回结果的哈希值。这有两个重要好处。首先，以太坊合约可以包含任何类型的计算，因此以太坊 ASIC 本质上是用于一般计算的 ASIC，即更好的 CPU。其次，挖矿需要访问整个区块链，这迫使矿工存储整个区块链并至少能够验证每笔交易。这样就消除了对中心化矿池的需求；虽然矿池仍然可以起到平衡奖励分配随机性的合法作用，但没有中心化控制的点对点矿池同样也可以很好地发挥此功能。

该模型未经测试，在将合约执行作为挖矿算法使用时，在避免某些巧妙优化的过程中可能会遇到困难。然而，这种算法有一个值得注意的特点，任何人都可以通过将专用于抑制某些 ASIC 的大量合约引入区块链中，在“井里下毒”。由于存在经济激励，ASIC 制造商会使用这种方法互相攻击。因此，我们正在开发的解决方案最终是一种适应性人为经济解决方案，而不是纯粹的技术解决方案。

可扩展性

可扩展性问题是以太坊常被关注的一个方面。像比特币一样，以太坊也有缺陷，即网络中的每个节点都需要处理每笔交易。

使用比特币，当前区块链的大小约为 15 GB，每小时增长约 1 MB。如果比特币网络像 Visa 一样每秒处理 2000 笔交易，它将每三秒增长 1 MB（每小时 1 GB，每年 8 TB）。以太坊可能也会经历相似甚至更糟的增长模式，因为以太坊区块链之上还有很多应用，不像比特币区块链上只有货币，但以太坊完整节点只需存储状态而不是完整的区块链历史，这一事实让情况得到了改善。

大区块链的问题是中心化风险。如果区块链大小增加到 100 TB，可能的情况是只有极少数大型企业能运行完整节点，而所有普通用户将使用轻 SPV 节点。在这种情况下，可能会出现这样的担忧：完整节点合伙欺诈牟利（例如更改区块奖励，给他们自己比特币等）。轻节点无法立即检测到这一点。当然，可能至少存在一个诚实的完整节点，几个小时之后有关诈骗的信息会通过 Reddit 这样的渠道泄露，但这时已为时过晚：将由普通用户相互组织协作将指定区块列入黑名单，这种大规模的、很可能不切实际的协作在规模上无异于发动一次成功的 51% 攻击。就比特币而言，目前这是一个问题，但 [Peter Todd 建议](#) 对区块链进行修改，以缓解这一问题。

在短期内，以太坊将使用两种其他策略来应对这个问题。首先，因为基于区块链的挖矿算法，至少每个矿工都会被强制成为一个完整节点，为完整节点的数量创建了一个下限。其次，更重要的是，处理完每笔交易后，我们会把一个中间状态树根收录到区块链中。即使区块验证是中心化的，只要存在一个诚实的验证节点，就可以通过验证协议规避中心化问题。如果矿工发布了无效区块，该区块必定是格式错误，或者是状态 $S[n]$ 不正确。由于已知 $S[0]$ 是正确的，因此必然存在第一个不正确的状态 $S[i]$ ，但状态 $S[i-1]$ 是正确的。验证节点将提供索引 i 以及“无效证明”，该证明包括处理 $\text{APPLY}(S[i-1], \text{TX}[i]) \rightarrow S[i]$ 所需的帕特里夏树节点的子集。节点将能够使用这些节点来运行该部分计算，并查看生成的 $S[i]$ 与提供的 $S[i]$ 是否不匹配。

另一种更复杂的攻击涉及恶意矿工发布不完整的区块，因此甚至不存在完整信息，致使无法确定区块是否有效。解决方案是质

询-应答协议：验证节点对目标交易索引发起“质疑”，接受到质疑信息的轻节点会对相应的区块取消信任，直到另外的节点（无论是矿工还是另一个验证者）提供一个帕特里夏树节点子集作为有效性证明。

结论

以太坊协议最初被设想为加密货币的升级版本，通过高度通用的编程语言提供高级功能，如区块链托管、提款限制、金融合约、博彩市场等。以太坊协议不会直接“支持”任何应用，但图灵完备编程语言的存在意味着，理论上可以为任何交易类型或应用创建任意合约。然而，关于以太坊更有趣的方面是，以太坊协议远远超出了货币的范畴。围绕去中心化文件存储、去中心化计算和去中心化预测市场的协议以及许多其他这类概念，有可能大大提高计算行业的效率，并首次通过添加经济层来大力促进其他点对点协议的发展。最后，还有大量与金钱完全无关的应用程序。

以太坊协议实现的任意状态转换函数的概念提供了一个具有独特潜力的平台；而不是一种专门针对数据存储、赌博或金融领域内一系列特定应用的封闭式单用途协议，以太坊在设计上是开放式的，我们相信在今后几年中它非常适合作为大量金融和非金融协议的基础层。

注释与延伸阅读

注释





1. 有经验的读者可能会注意到，事实上比特币地址是椭圆曲线公钥的哈希，而非公钥本身。然而事实上从密码学术语角度把公钥哈希称为公钥完全合理。这是因为比特币密码学可以视为一种定制的数字签名算法。在数字签名算法中，公钥由ECC（椭圆曲线加密算法）公钥的哈希组成，签名由连接了

ECC 签名的 ECC 公钥组成。而验证算法涉及用 ECC 公钥哈希（作为公钥提供）来检查签名中的 ECC 公钥，然后用 ECC 公钥来验证 ECC 签名。

2. 技术上来说，前 11 个区块的中位数。
3. 在内部，2 和 "CHARLIE" 都是数字 ^[fn3]（注释编号），后者采用大端序基数 256 表示。数字可以至少为 0，最大为 $2^{256}-1$ 。

延伸阅读

1. [内在价值](#) 
2. [智能资产](#) 
3. [智能合约](#) 
4. [B-money](#) 
5. [可重复使用的工作量证明](#) 
6. [利用所有者权限确保财产权](#) 
7. [比特币白皮书](#) 
8. [域名币](#) 
9. [佐科三角](#) 
10. [彩色币白皮书](#) 
11. [万事达币白皮书](#) 
12. [去中心化自治公司，比特币杂志](#) 
13. [简化支付确认](#) 
14. [默克尔树](#) 
15. [帕特里夏树](#) 
16. [GHOST 协议](#) 
17. [StorJ 和自治代理，Jeff Garzik](#) 

18. [Mike Hearn 在图灵节上谈论智能资产](#) 
19. [以太坊递归长度前缀编码 \(RLP\)](#) 
20. [以太坊默克尔帕特里夏树](#) 
21. [Peter Todd 论默克尔求和树](#) 

有关本白皮书的历史，请参阅[此维基文章](#) .

和众多社区驱动的开源软件项目一样，以太坊自启动以来一直不断发展。若想了解以太坊的最新进展以及如何更改以太坊协议，我们推荐您阅读[本指南](#)。

本文对您有帮助吗？



是



否

网站最后更新: 2023年12月1日



使用以太坊

查找钱包

获取以太币

去中心化应用 (dapps)

二层网络

运行节点

隐私

质押ETH

质押ETH

学习

学习中心

什么是以太坊？

什么是以太坊 (ETH)？

以太坊钱包

Gas fees

以太坊安全和预防欺诈措施

什么是 Web3？

智能合约

以太坊能源消耗

以太坊路线图

以太坊改进提议

以太坊的历史

以太坊白皮书

以太坊词汇表

以太坊治理

区块链桥

零知识证明

测试中心

开发者

开始体验

以太坊

以太坊

教程

通过编码来学习

设置本地环境

生态系统

社区中心

以太坊基金会

以太坊基金会博客 [↗](#)

生态系统支持方案 [↗](#)

以太坊漏洞悬赏计划

生态系统资助计划

以太坊品牌资产

Devcon [↗](#)

企业级应用

主网以太坊

私人以太坊

企业级应用

关于ethereum.org

关于我们

工作机会

参与贡献

语言支持

隐私政策

使用条款

Cookie 政策

联系我们 [↗](#)