以太坊平台评估

私有链和联盟链的机会与挑战

作者：Vitalik Buterin

翻译：万向区块链实验室/ChinaLedger 联盟

排版/校对：区块链铅笔（ChainB.com）

目录

以太坊平台的构想最早是在 2013 年 11 月提出来的，当时的目标是创建一个更通用的区块链平台——通过工作量证明机制（或最终转换成权益证明机制）实现了公共经济共识的概念，并将其与状态丰富的图灵完备虚拟机的抽象能力结合起来，从而让应用程序开发者更容易地创建区块链上的应用程序，并能受益于区块链的去中心化和安全的特性，特别是避免了为每一个新的应用程序创建一个新区块链的麻烦。

而之前曾经出现过的区块链协议可以视为是单一功能的工具，就如口袋中的计算器，即使在好一点的情况下也就像瑞士军刀这样的多功能工具而已；以太坊则是区块链中的“智能手机”：它是一个通用的平台，你可以在上面搭建自己需要实现的功能，就如搭建一个“APP”一样，以太坊的用户将会即时接触到它能提供的好处，而不需下载任何新的软件。

虽然这个项目原本是作为“Mastercoin 功能升级提议”的一部分提出来的（目的是为金融合约提供支持），后来项目的关注点扩展到了更广泛的应用领域，包括金融合约，下注，数字代币（资产）发行，去中心化文件存储激励机制，投票，“去中心化自治组织”等。以太坊平台开发的资金是通过在 2014 年 8 月的一场众筹活动筹集回来的。自从那时起，平台上已经出现了超过 100 个应用程序，范围包括金融清算和结算，到保险，数字资产发行，甚至是非金融领域的应用（包括投票和物联网）。

基础设计

有的区块链都有一个“历史”的概念——所有之前的交易、转账及其发生顺序的集合，以及“状态”——决定某个给定交易是否有效以及在交易处理后状态将会如何改变的“当前相关的”数据。区块链协议同时有一个“状态转换规则”的概念：基于之前的状态，以及一个给定的交易，（i）这个交易是有效的吗？还有，(ii)在交易发生后，状态会是什么？

我们可以用比特币[1]给出一个例子。在比特币中，状态是账户余额的集合（如地址 39BaMQCphFXyYAvcoGpeKtnptLJ9v6cdFY 有 522.11790015 个比特币，地址 375zAYokrLtBVv6bY47bf2YdJH1EYsgyNR 有 375 个比特币……）。状态转换功能

取得发送者的地址、目标地址以及一个值，然后询问：（i）这个交易已经由发送者用密码学的方式正确签名了吗？还有（ii）发送者的账户里面有足够的比特币用于本次的发送吗？如果其中一个问题的答案是否定的，那么交易就是无效的，并且不能被包含到一个区块里面。如果一个区块包含了当前状态下的一个无效交易，那么这个区块就会被网络忽略掉[2]。如果两个问题的答案都是正确的，那么交易的价值就会从发送者的余额中抽离出来，并添加到接收者的余额中。

在以太坊中，相关的设计会更复杂一些。状态可以被视为是所有账户的集合，而每一个账户要不就是“外部拥有的账户”（EOA，externally owned account）或一个合约（contract）。如果账户属于 EOA，则状态会简单地存储这个账户的以太币余额（以太坊的内部加密代币，其功能，类似比特币或 XRP）以及一系列用于防止重复支付攻击的序列号。如果账户是一个合约，则状态会存储合约的代码，以及合约的存储空间——一种键值数据库（key-value database）。

在以太坊中一个交易会指定（跟其他的一些所需信息一起，这会在后面会提到）一个目标地址、用于交易的以太币数量，以及一个理论上能存储任何信息的“数据”域（另外，还有一个发送人的地址，不过这在签名里已经指定了，因此这里没有专门说明）。如果一个交易是对 EOA 或一个尚未存在的账户发出的，则它除了简单地作为发送以太币的手段外，并没有其用途。如果一个交易被发送到一个合约里，则会执行合约的代码。这个代码有能力实现：

- 读取交易数据。
- 读取交易中被发送的以太币余额。
- 读取或写入合约自己的存储空间。
- 读取环境变量（如时间戳，区块难度，前一个区块的哈希值）
- 向另一个合约发送一个“内部交易”。

本质上说，你可以将合约视为是存储在以太坊的状态里的一类“虚拟对象”，它可以包含自己内部的永久存储空间，并有能力像外部用户一个拥有对其他合约的操作权限和关系。内部交易是由合约创建的一种交易；就如普通的“外部”交易一样，它也包含了一个明确的发送者、目标地址、以太币数量以及信息数据，

如果一个内部交易被发送到一个合约，则合约的代码就会运行起来；在合约执行结束后，合约的代码将有能力返回 0 或更多字节数的数据，可以让内部交易用于“询问”其他合约去获取特定的信息。一个交易（transaction）可以创建一个新的合约——通过将合约的代码放置在交易的数据中（目标地址不进行设置），或通过 CREATE 这个操作代码（opcode）从合约的内部实现这个功能。

简单地说，在以太坊的公有链上，合约机制是以如下的形式被使用的：

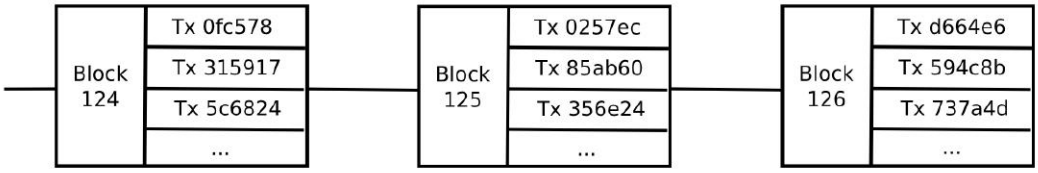
- 作为一个追踪用户发行资产（issuer-backed assets）的数据库；
- 作为一个可控制其他资产的“智能合约”（下面会详细介绍这个概念），并将根据特定的条件将这些资产发送到特定的参与方；这通常会进一步分解成几个子类别，包括（i）金融合约（如差价合约，二元期权，衍生品），（ii）担保交易（实施“无需信任的数字资产原子化互换”），（iii）多方协议，如拍卖，保险合约，鼓励披露特定信息的经济游戏等；
- 作为一个基于区块链的域名系统的注册记录管理；
- 作为一个代表用户或机构的账户，不过拥有复杂的访问权限控制，如多重签名；
- 作为“软件库”，让代码可以写入、发布到区块链上，并由其他的人使用；

“智能合约”的概念通常被简单地定义为“一个直接控制数字资产的电脑程序”[3]，这是特别重要的。合约有它们自己的地址，因此可以像用户一样作为数字资产的持有者了如果一个合约真的“拥有”数字资产，这意味着（i）只有合约自己的执行过程能够向另一方发送资产，还有（ii）每一个参与方在区块链上可以看到并检验这个资产真的在程序的控制下。

例如，你可以在无需信任的情况下实现资产 A 与资产 B 的交易——通过资产 A 的拥有者将资产发送到一个程序里，这个程序的代码定义了“如果我在 24 小时内收到资产 B，则我会将资产 A 发送到发送人的地址里，并将资产 B 发给我的创建者，否则我将会把资产 A 发送回我的创建者”。资产 B 的拥有者可以看到该合约确实控制了资产 A，而且知道一旦他们将资产 B 发送到合约的账户，合约将

会进行一个公平、正确的交易。合约并没有“拥有者”（owner）这个概念；当资产 A 的原始拥有者把该资产发送到合约里，则他们就无法通过操纵这个合约把资产拿回来，而是只能等待交易成功完成并接收所交换到的资产 B，或者 24 小时后，若交易还是没有成功，则他们会自动地拿回资产 A。

区块链



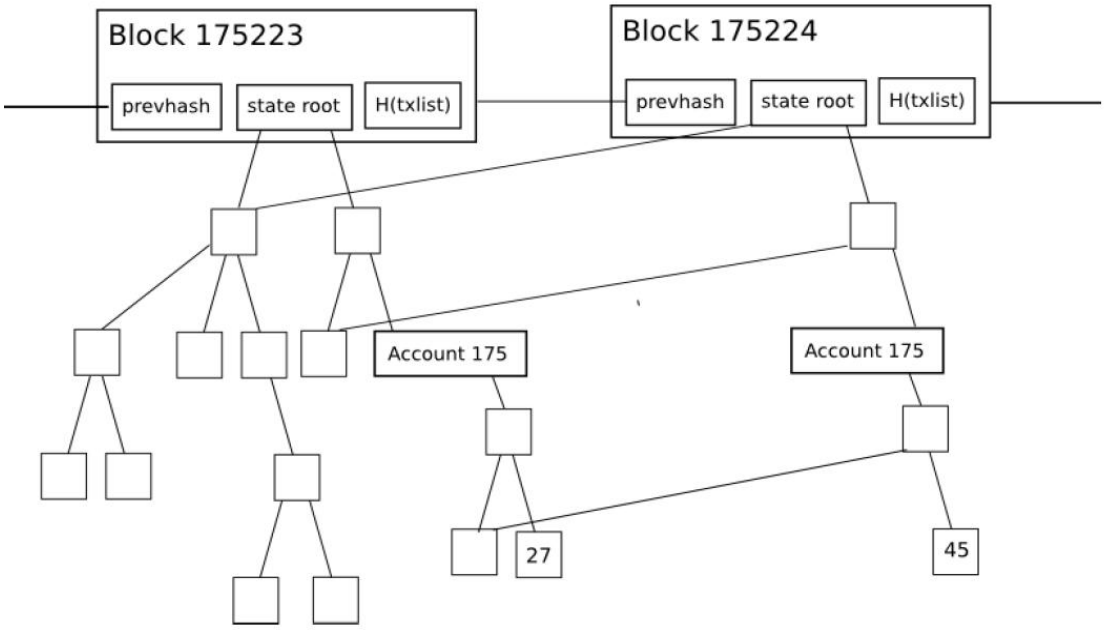
以太坊区块链在较高的层级上有点像比特币的区块链。最主要的也是最重的——区块链由一系列的区块所组成，每一个区块包含指向前一个区块的指针，以及一个经过排序的交易列表。区块是由工作量证明所维护的，“最长的链”（由整体难度决定）定义了一系列经过确认的交易及其执行的顺序。

为了到达以太坊区块链上的“当前状态”，节点可以从“初始状态”开始（这是一个内嵌在每一个以太坊客户端里的、所有人都认可的初始状态），并处理每一笔交易，并按顺序执行由交易处理和代码执行过程所带来的余额/序列号/代码/存储的变化。比特币里面也有同样的过程；虽然以太坊对交易执行过程“状态变化”模型的重视是独特的（在比特币中，交易通常被视为花费来自历史上前一个交易的“输出项”，而不是状态里的某个对象）。但以太坊、比特币和瑞波、狗狗币等协议里，代码在这部分工作中的运行模式大体上是一样的。

以太坊使用了内存需求较高的哈希函数，uncle 激励机制，难度调整算法，gas 限制调整算法等，这些都是以太坊的一些值得注意的不同之处，不过这些都是优化工作，可以说在以太坊的本质中并不是处于中心的地位。

以太坊中另一个值得注意的特性（比特币中没有的）是 Merkle 树的深度使用，让“轻客户端”可以只下载和校验区块的头信息，同时也能在有需要的时候

决定并安全地对区块链状态的任何特定部分进行校验。而在比特币中，一个区块头只储存包含交易的 Merkle 树的根哈希值（root hash），在以太坊中每一个区块投包含了一个“状态根”（state root），实质上是包括了整个当前状态（账户余额，序列号，代码和储存）的密码学哈希树的根哈希值。



因此，轻客户端通常可以只下载区块头（每 17 秒约 500 字节），如果轻客户端需要了解状态中的某个特定数值（如“账户 0x124b6f72 中的余额是多少？”或“账户存储空间中键值为 178233 的记录对应什么数据？”），它就可以请求网络中的任何“完全节点”提供一个“分支”——这是（Merkle）树结构中的数据区块构成的一系列哈希值，指定了直到树根部的特定信息，而轻客户端可以自己验证该分支的完整性。如果该分支是正确的，它就接纳这个答案。“轻客户端”模式对那些使用智能手机和物联网/嵌入式设备的以太坊用户是很有用的，对那些电脑性能较低、网络带宽较差的用户也是很有用的。

以太坊的私有链和公有链

需要注意到，虽然以太坊原来是以公有链的形式存在的，但以太坊的状态转换规则（即协议中处理交易、执行合约代码等的部分）可以从以太坊的公有链共

识算法（如工作量证明）中抽离出来，因此完全可以创建运行以太坊代码的私有链（在一个节点上运行，这个节点由一个公司控制）或联盟链（由一系列预先指定的节点运行）。[4]以太坊技术自身因此可谓是与具体的应用场景不相关的——无论是用于公有链、联盟链或私有链的模式，因此我们的目标是让以太坊的多种实例的互操作性实现最大化——如可以将为以太坊公有链书写的合约和应用程序导入到以太坊的私有链中，反过来也一样。

现在，在私有链的应用场景中已经有几个版本的以太坊在进行开发了，如Hydrachain。私有链版本的以太坊实施方案在理论上会一些可扩展性方面的提升，但若这些特性要应用到公有链版本的以太坊上，还是需要完成不少的工作。

金融领域中的应用

虽然以太坊是一个高度通用化的平台，其用途理论上是很广泛的，但以太坊上的主要应用项目还是具有某些金融特性。这些应用项目可以进一步分解成“纯金融性的”，即只进行有价金融资产管理、创建金融合约、担保交易等；另一种是“半金融性”的，将一些本身并非金融应用的服务与金融、支付用例结合起来。

后一个类别的例子由很多，从“去中心化的 Uber”平台，到专注于机构服务的贸易融资项目（如在 2016 年 1 月赢得上海区块链黑客马拉松大奖的 CargoChain），这些项目都利用了以太坊区块链的数字资产管理和智能合约功能及其在金融应用领域之外的功能，将以太坊用作一个航运记录跟踪、声誉和评价信息，以及为签订的法律合约提供存在性证明的数据库。

在“纯金融性应用”的方面，我们可以看到：

- 基于区块链的金融合约和衍生品处理平台（如 Clearmatics）；
- 在区块链上实施的其他金融工具（如 UBS 的“智能债券”平台）；
- 从黄金（如 Digix Global）到以法币计价的“结算币”这类对现实世界资产进行数字化的应用，为金融交易和简单的主流支付服务；
- 使用区块链实现差价合约（CFD），用智能合约执行，并用无对手风险的加密货币进行抵押品管理和结算工作，以创建映射现实世界资产的“镜像资

产”，在那些有足够流动性和套利空间的市场中提供对底层资产的相同回报率。

- 基于区块链或基于智能合约的抵押品管理。

有一类非金融领域的区块链应用所提供的服务本来在金融平台上就是很有用的（无论是否基于区块链技术）；或许最佳的例子就是身份验证系统。

为你的应用评估以太坊私有链或公有链的适用性

评估以太坊在这些应用上的适用性涉及两个主要的步骤。第一步是去决定区块链是否真的适合做这个事情。区块链的基本技术优势包括可靠性，安全性，可审计性以及去中心化，这些都是人们了解得比较多的，不过人们考虑得比较少的则是这些特性能在什么场所发挥作用。主流的支付系统需要依靠联盟数据库或公共数据库去实现密码学意义上的可审计性及去中心化吗？证券交易呢？航运或航空票务市场呢？或文件存储、计算呢？商家的积分计划呢？在非金融应用那边，若要对不同的现实世界资产或数字资产（如域名）的所有权进行追踪呢？电子邮件和社交网络应用呢？

这样，我们就面临一个选择了——公有链和私有链哪种更合适？可以这么说，以太坊的优势在公有链或联盟链上是有所不同的。在公有链上，除了能够实现程序化的担保交易和金融合约外，以太坊能提供的一个好处是协同性，正如一位以太坊应用程序开发者最近描述过的那样。以太坊中的合约可以满足不同的功能，而每一种建造在以太坊之上的应用程序在理论上可以利用其它应用程序提供的功能。

例如，如果你希望用区块链去管理公司股份的所有权，那么从安全性和可校验的记录管理角度来看，这个工具确实是很有用的，但另一个好处是让股权众筹变得更简单了：公司可以将股份转移到一个智能合约中，任何人若将 Y 个单位的加密货币 Z 发送到该合约，该合约就会自动往他的账户中返回 X 个单位的股份，而合约中的这些钱若要提取出来，必须经过 5 个董事中的 3 个人的授权。如果公司所在地有相应的监管要求，需要进行某种形式的 KYC（了解你的客户）认证或对投资者的身份进行限制，只要有人在以太坊上设计了一个基于区块链的 KYC

和认证平台，前面提到过的这个股权众筹平台以及其他的股权众筹平台（也可以是广义的金融应用项目）可以立刻与该身份认证系统进行互动，并往合约中添加相应的限制，要求只有经过授权的个人才能达成有效的股权购买行为。[5]

具体来说，我们可以认为协同性是以太坊和所谓的“双层”区块链智能合约系统（如现在已经停止了的 Codius 项目）的主要差别，后者将区块链当成是一个纯粹用于追踪资产所有权的层（甚至是更“傻瓜化”的纯数据层），并要求每一个应用程序通过多重签名“公证人”或让用户独自处理区块链并“解读”其结果的方式单独去处理智能合约；协同性的设计目标是让应用程序就智能合约执行结果正确性的来源达成共识（每一个人都同意这个来源是安全的），因此在协议的层面引入这种机制将鼓励其作为共识算法的一部分从而实现公共利益，是有经济意义的。[6]

在私有链中，论点就有所不同了。私有链通常是用作为特定产业创建结算平台的工具，通过构建一个实质上是唯一的、共同的数据库，让机构之间的交易处理效率赶上机构内的交易处理效率。私有链的理想“用户”实质上是一个在一定程度上已经去中心化的产业，这产业里面没有一个公司拥有超过略高于两位数的市场份额，不幸的是，这种去中心化的状态在当前的技术水平下明显降低了效率：如果一个公司的客户希望与其他公司的另一个客户进行某种互动（如发送付款，执行交易等），就需要一个繁琐和笨重的机构间对账和结算过程，这通常带来了明显的费用以及延迟。使用区块链的话，当前的“半去中心化”的产业“政治架构”照样可以保存下来——并不需要说服所有的参与方进行合并或成为某个超级公司的客户，更不需要说服那些有反垄断政策的监管者同意这种合并。同时，通过技术上的简单变革，得到了大规模的网络效应和高度的互操作性的好处。

另一个潜在的好处是通过“分离关注点”的做法提高金融产业的效率：银行并不需要亲自对每一种类别的金融应用进行创新，而是可以将这个任务交给更灵活的金融企业，这些企业正在往类似软件提供商这种角色的方向发展，它们可以帮助用户在区块链上发送经过密码学签名的交易、执行交易、以及达成涉及有对应背书资产的代币相关的合约，而不需自己对实际的资产进行托管（因此降低了

它们自身的监管负担），而银行还是会保留它们的核心业务——接收存款，以及发放贷款。

在私有链的场景中，协同性就显得没那么重要了，因为每一个私有链更有可能被设计成专门针对某种特定应用。不过，还是有可能实现某种程度上的“私有链间互动”或“公有链与私有链互动”这样的协同模式。现在已经有一些针对跨链资产交易这种特定用例的尝试了，包括 TierNolan 的跨链互换协议，以及 Interledger 最近发布的一些成果；然而，以太坊的策略更具雄心壮志，寻求创造更通用的“搭桥”机制，让从一个区块链读取另一个区块链的内容成为可能（例如，BTCrelay 是比特币和以太坊之间的桥），长期的目标将会是整合一个通用的异步编程语言，让应用程序可以跨越多个区块链开展。若这是一个有较高需求的特性，则计划中的以太坊 2.0 的很多技术可以重点去达成这些目标；若这个功能被认为重要性不高，则以太坊的功能就会受限在智能合约和未来的可证明性上，至于“傻瓜区块链”和“智能区块链”之间的取舍及其效率与复杂性的平衡，这是要由用户去决定的。

当一个公司决定区块链是可行的策略时，它就需要选择具体的平台。以太坊的好处是可编程性，灵活性，协同性，模块性，以及容易使用的哲学思想——毕竟，我们现在和未来五年内都不太可能完全分析清楚开发者们到底需要什么样的特性。如果公司的法务研究工作决定某类特定的应用需要 KYC（了解你的客户）认证、登记限制或其他规则，那么身份认证可以作为一个独立的层进行搭建，并可以书写直接接入这个系统的合约。在一个股份可以被私下进行交易的公司里，你希望股份登记在区块链上，而且希望增加一个限制，即只有经过 51%的现有股东同意才能增加新的股东，这样你还是可以在不改变基础层或系统的其他部分就可以实现这个目的。

即使是对如支付这样简单点的应用来说，刚开始时在以太坊之上搭建这种应用，可以快速地在基础的资产层上整合更高级的应用功能，如金融合约，抵押品管理，无需信任的原子化互换等，这样将来需要这些功能就可以轻易地添加上去。以太坊未来的版本将会继续这种“未来兼容性”，甚至将其扩展到连密码学的层面——用户甚至可以选择用于保护其账号的密码学算法。例如，如果你对量子计

算机（译者注：具有强大运算性能，对一些加密方法带来威胁）感到担忧，而且想快速地升级到 Lamport 签名机制，就可以如愿以偿，无需等待整个区块链的协议的进化。

自然而然地，有些人认为“如果能用一种简单的工具去完成一项任务，就不要用复杂的工具了”，随之而来的还有认为以太坊的通用性或其实施的特定方式带来了特定的效率问题。前一种论点需要就具体的项目进行具体的分析，并与以太坊的特性进行对比；后面的论点将会在接下来的以太坊路线图中进行详细讨论。

以太坊开发路线图概要

以太坊项目当前预期的里程碑目标将会在下面的章节展开讨论：

- Metropolis:Mist 浏览器的发布，预计在 2016 年的夏天或秋天；
- Serenity（“以太坊 1.5”）:发布区块链的 PoS 股权证明(Casper)版本，同时包含以太坊改善提议（EIPs）101 和 105。这计划在 2017 年早期实现。
- WebAssembly 发布（“以太坊 1.75”）:更快的虚拟机。预计在 2017 年到来。
- 以太坊 2.0（尚未命名）：初步可扩展性提升的版本。预期在 2017 年晚期实现。
- 以太坊 3.0（尚未命名）：“没有限制的”可扩展性版本，预计在 2018 年晚期实现。

这些名字在接下来的章节中将会被大量使用。

安全性

（以太坊）作为一个在大范围内复制、共享账本的图灵完备状态机，能让世界上任何能购买 0.3 美分价值以太币的人上传代码，然后网络中的每一个参与者都必须在自己本地的机器上运行这段代码，这确实是会带来一些明显的与安全性相关的忧虑。毕竟，其他的一些平台也提供类似的功能，这包括了 Flash 和 Javascript——它们经常碰到“堆与缓冲区溢出攻击”，沙盒逃逸攻击以及大量

的其他漏洞，让攻击者可以做任何事情，甚至包括控制你的整台计算机。除此之外，还会有拒绝服务攻击（D.D.O.S.）——强迫虚拟机去执行无限循环的代码。

为解决这些挑战，以太坊项目引入了不少的策略。首先，以太坊的虚拟机的架构是高度简化和受限的，之所以要从头搭建一个全新的虚拟机而不是用如Java 这样现有的虚拟机，是为了实现虚拟机的高度安全性。与访问系统资源、直接读取内存或与文件系统互动的操作代码（Op Codes）在 EVM 以太坊虚拟机的设计规格中是不存在的；与此相反，唯一存在的“系统”或“环境”操作代码是与以太坊“状态”里定义的架构进行互动的：虚拟机内存，堆栈，存储空间，代码以及区块链环境信息（如时间戳）。

以太坊虚拟机的大多数实施方案是解释型的，不过有一个由 JIT 编译的虚拟机已经被开发出来了。以太坊虚拟机（以及以太坊协议的其他部分）已经有 6 个实施版本了，也经历了超过 50000 个单元测试，以确保完美的兼容性和确定性的执行结果。[7] Go 语言、C++和 Python 语言的实施版本在我们的安全审计机构 Deja Vu 的帮助下，已经进行了深度的安全性审计。在以太坊发布后的几个月时间里，发现过几个需要修复的安全性漏洞，不过这种事发生的频率已经大幅度降低了：在本文行文前的两个月，以太坊网络一直在正常运行，而没有发现任何明显的安全性漏洞[8]。

无限循环攻击的解决方案是最复杂的。总体上说，任何图灵完备的编程语言在理论上都是会碰到“停机问题”的，即不可能预先确定给定程序在给定的输入值下进行运行是否会出现停机问题。其中一个例子就是考拉兹猜想（Collatz Conjecture）。假设有如下的程序：

```
def collatz(n):
    while n > 1:
        if n % 2:
            n = n * 3 + 1
else:
    n = n / 2
```

该猜想认为，如果你使用任何输入值去运行这个程序，这个程序最终会停止，不过我们并不能证明这一点；若在找到一个反面例子的情况下我们就可以证明

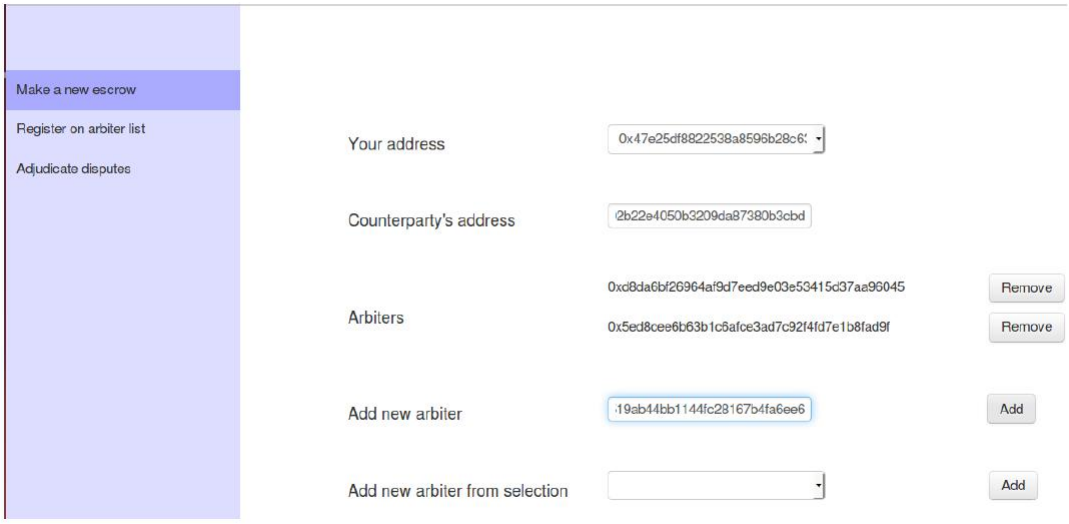
“并不是这样的”，但现在人们已经尝试过 1 万亿以上的输入值了，也没能找到这样的一个反面例子。因此，这带来了一个忧虑——攻击者可以创建一个合约，里面包含一些混淆的无限循环代码，然后往这个合约里面发送一笔交易，从而将系统拖垮。

以太坊解决这个问题的概念是汽油（Gas）：交易的发送者定义他们授权代码运行所需的最大计算步数，然后为此支付相应比例的以太币。在实际中，不同的操作过程需要耗费不同的 Gas 数量，这些耗费的标准不仅是基于执行每一种操作过程所需的运算时间，还包括如全节点储存以及内存耗费等考量因素，所以 Gas 并不只是一个用于统计运算步数的标准，不过我们初步就理解成“Gas=计算步数”吧，这对理解 Gas 的用途基本上足够了。

若在执行交易的过程中“Gas”被耗尽了，如超出了其允许的最大计算步数所需的预算，则交易的执行会被回滚，但交易还是正确的（只是不再有效了），发送者照样要付相应的费用；因此，交易的发送者必须在下面两种策略之间进行取舍：设置一个更高的 Gas 限额，这可能会支付更高的交易费；或设置一个较低的 Gas 限额，这样可能会创建出一个会被回滚的交易，就需要以一个更高的限额重新发送一遍了。信息（译者注：可用于合约间互动的一个特性）本身也可以设置 Gas 的限额，因此可能让一个合约与其他合约进行互动，而无需担心其他合约会无节制地消耗自己的合约里的 Gas“预算”。这个机制已经被密码学学者 Andrew Miller 及其他人审查过了，其结论是这个机制确实能实现逃过停机问题的目的，并以经济学的方式分配运算能力，不过在某些边缘案例中激励机制可能不是一个最优解[9]。

第三个安全性问题在较高的层面展现出来：若我看到某人提供的一个智能合约，上面写着它是一个二元期权合约，需要收取 10 美元的费用，而且若 3 月 30 日的金价超过 1100 美元时则会向你支付 20 美元——我如何知道这是真的呢？如果这是一个合约，用于管理整个市场内的二元期权合约，让用户可以出价、投标、完成交易，我如何知道这个合约没有能让作者拿着所有的钱跑路的后门呢？一个更好的问题是，假设这个合约的作者编程水平有限，他们如何知道自己写出的程序代码没有漏洞，让大家永远都没法将钱从合约中拿走呢？

上述的两个问题——程序员作恶和程序员出错——是独立的问题，而解决问题也有所不同，但两者之间还是有共同点的。以太坊里面临的挑战对应两类解决方案，分别是低科技的和高科技的解决方案。高科技解决方案在理论上能提供更好的希望，不过会带来整合的难度，以及更依赖于复杂的工具。而针对作恶问题的低科技解决方案有两个层面。第一，在以太坊里，我们明确以下两者的区别，即应用程序的核心（纯粹由智能合约的集合构成）以及界面（HTML 和 Javascript 代码通过读取区块链及发送交易的方式与核心通讯）。



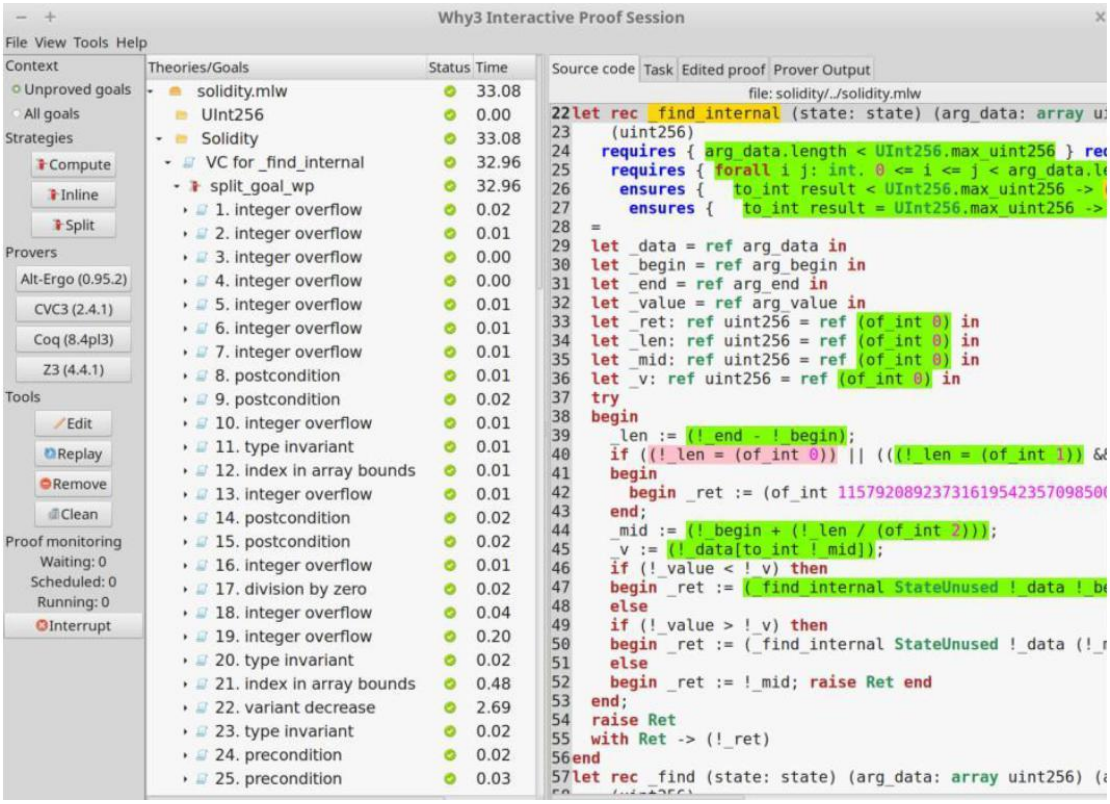
这是一个为担保交易 Dapp 应用而设的简单、美观的用户界面。不过你怎么知道点击“移除”按钮时并不是将你的全部钱发送给开发者呢？

我们的目标是让核心变得可信，若要实现这点，核心就必须尽量精简，并且经过深入的审计和审查；而用户界面可能包括大量的代码，可以无需这么信任；我们的一个中期的设计目标是让以太坊用户界面去保护用户，以免他们受到来自作恶的交易界面的损害，如通过强制性的弹出一个“你是否想将带有这些数据的一个交易发送到这个合约里？”对话框（或其他形式的通知方法）。我们希望会呈现出一些专业事务所的市场空间，就如今天律师事务所创建标准化法律合约那样，这些专业事务所会为不同的用例创建标准化的合约，并让这些合约接受来自第三方审计人员的深入检查。

审计和标准化可以减少代码错误带来的损害，不过在错误的特例里，已经有人进行了数十年的研究，引入了一种强类型要求的编程语言，专门用于避免错误；这类语言让你可以更丰富地指定每一种数据的含义，并自动防止数据被以明显错误的方式组合起来（如时间戳加上了一个货币价值，或由区块哈希值分割的地址）。

更高级的解决方案集合依赖于名为形式化验证（Formal Verification）的技术。简单地说，形式化验证就是使用计算机程序自动地在数学的层面上证明关于其他计算机程序的语句。其中一个简单的例子就是证明一个排序算法的正确性：给定一段代码，它以一个数组作为输入项，并提供一个数组作为输出项，你可以插入一个“定理”，如 for x ϵ I: x ϵ O; j > i => O[j] >= O[i]（若用英语表达，则是“证明了每一个在输入数组里的值都在输出数组里面，所以输出数组肯定是输入数组的某种排列方式，并证明了输出是以递增的顺序出现”）。形式化验证者将会处理这个代码，构造出肯定或否定这个定理的数学证明，或放弃处理（在考拉兹猜想的例子里有可能会这样）。Aesthetic Integration 公司的 Imandra 产品甚至可以自动找出错误定理的反面例子。

现在，以太坊的高级编程语言 Solidity（编译到以太坊的 bytecode）的主导开发者 Christian Reitwiessner 正在将形式化证明引擎 Why3 整合到 Solidity 里面，让用户可以在 Solidity 程序里面插入与某种数学论点有关的证据，并在编译的时候进行验证；还有一个项目在将 lmandra 整合到以太坊虚拟机的代码里。



通过 why3 引擎实现 Solidity 的形式化证明

形式化证明是一项强大的技术；不过，它无法解决这样的一个问题：有一些事情，可能连我们自己都不知道应该要去证明。人类对公平和正确性的定义往往是非常复杂的，而由此带来的复杂影响往往是难以检测的。市场的订单本（译者注：Order Book，即撮合交易引擎常见的买卖单系统）通常需要有一些基础的正确性规则(你要不就以所出的价格得到你需要的东西，要不系统就把钱返还给你)，顺序的不变性（为了防止 front-running，即所谓的提前交易或抢单），确保每一个买家和卖家都能与最佳的对手单匹配上，以及其他等等；不幸的是，我们并不能简单地将所有的情况都列举出来，也无法确保我们没有任何遗漏的事项。现在看来，对我们来说选择什么东西进行验证会是继续是一项艺术（难题），不过形式化证明技术肯定会降低攻击者们进行攻击的自由度，让对代码进行审计的人员降低负担。

关键建议

金融机构应该考虑使用以太坊的公有链或私有链版本，或者使用一个基于EVM（以太坊虚拟机引擎）的系统，毕竟这个虚拟机是以完美的确定性和安全性的出发点去设计和测试的，这些两个目标对分布式账本的用例是很有用的。（译者注：基于 EVM 的系统，并不一定指以太坊，而是指 EVM 这个虚拟机引擎的设计能够即使是与其他的系统结合起来，例如非区块链技术的系统，也能发挥强大的功能）。具体来说，除了 EVM 之外，当前并没有其他主流的虚拟机引擎包含了 Gas 计数器这个概念，而这个机制对以下的目标是必须的：(i)避免无限循环攻击，(ii)避免在某些边缘案例里的具备不确定性的行为。在一个私有链的实例里面，并不一定要用以太币去支付 Gas（或者其他密码学代币）；你可以直接给用户分配“Gas 资源“，有点像亚马逊云服务器给客服分配“CPU 时间”一样，你也可以简单地要求所有的交易都包含一个给定的最大限制（如每个交易的 gas 上限是 100 万），（译者注：这样即使有人发动无限循环攻击，他也只能最多执行 100 万 Gas 允许执行的计算步数，无法导致系统长期的死循环）。

不过，除了虚拟机的安全性外我们还有其他需要考虑的事情：在以太坊平台上，用户若要与一个智能合约进行互动，必须确定他们面对的程序是以他们想象中的执行方式去运行（译者注：例如，一个旨在进行遗产分配的智能合约，用户必须确定当自己的财产划拨进这个智能合约后，这个智能合约在日后真的会自动地按照设定的规则去分配遗产）。区块链是一个提供了密码学担保的平台，具有一定的便利性，毕竟代码是存储在区块链上的，所有与合约相关的参与者都能查看此代码，用户也能确定代码真的会被执行，不过若要确保不存在任何形式的作恶行为和意外的漏洞，这依然具有挑战性（可参考 Underhanded Contest，这是一个所谓的“卑鄙 C 程序大赛”，专门编写一些“说是一套，做是另一套”的程序，里面的一些例子具有相当的代表性）。我们推荐用户去探索形式化验证技术，在他们与代码互动的过程中进行自动化的验证，在金融行业的用例里面，涉及的经济利益较大，这样的技术可能是很有用的。

效率

效率问题是人们对以太坊平台的主要忧虑之一。具体来说，一些人认为以太坊平台太通用了，因此可能会成为“什么都能做，但什么都不精通”的例子。由比特股团队（BitShares team）及其他人提出来的一类观点就是虚拟机内执行的代码必然会比原生的代码慢，相比之下，那种支持很多“预编译”操作代码的平台显然会更高效。Gideon Greenspan（MultiChain 平台所属公司的 CEO）也提出了与并发性有关的论点，不过平行化和计算效率是不同的概念，所以将会在下文与可扩展性相关的段落叙述这个问题。第三个担忧是以太坊区块链里进行的一些操作指令，特别是 Merkle 哈希过程，带来了不必要的缓慢，因此可以将它移除以实现更高的效率（而且，在私有链的场合，可能不太需要这个了）。

EVM 以太坊虚拟机

或许可以说，虚拟机的问题是目前最复杂的。区块链最早的脚本语言——比特币脚本，是一个模仿 Forth 语言搭建的基于堆栈的编程语言，它的基础算法带有无大小限制的整数，字符串操作，以及一系列的高级密码学原语。其执行引擎的效率还是比较低的，显然不适合在脚本的层面创建任何形式的复杂应用。不过，因为 UTXO 模型固有的“无状态”的特性，在比特币之上搭建高级应用一直是不太可能的，因此在比特币之上的脚本执行效率一直没有受到太多的关注（这个情况在最近出现了变化：在一篇近期的博客文章里面，比特币的核心开发团队反对区块大小从 1MB 扩展到 2 MB，认为这样的话一个体积最大的交易可能需要十分钟才能校验完毕）。

以太坊原来的编程语言有意做得像比特币脚本那样，除了带有一个状态丰富的执行环境，而不是比特币脚本那种无状态的。对 256 位整数的限制被添加进去了（比特币脚本里还是禁止了这个限制），是为了防止整数乘法成为一种拒绝服务攻击的载体。它也曾经考虑过加入基于整数大小的动态 Gas 消耗机制，不过基于效率的考虑最终还是剔除了。256 位被视为一个最佳的尺寸上限，因为它执行和计算起来还是相对较快的，而且也具有足够的资源去处理椭圆曲线加密算法

（使用 256 位数）和哈希（大多数主流的哈希算法会提供一个 256 位的输出值）。它里面带有专为哈希和椭圆曲线操作的操作代码（opcodes）。

那时候的用意是为了支持简单的金融脚本、if-then 条件语句以及其他的应用程序，这些场景里 256 位的虚拟机可能会存在一定的效率问题，但相对于在交易中校验椭圆曲线签名涉及的耗费，这些效率问题还是可以忽略不计的（这有点像比特币的脚本）。在大多数的案例中，以下的描述是成立的：对一个签名进行校验需要处理超过一千个模块化的乘法运算操作，而运行实际的虚拟机代码只需要简单地处理一些条件语句及变量的变化。因此，那种认为“只”校验签名和运行“完整”的程序之间存在巨大成本差别的看法，在很大程度上是错误的：不管虚拟机的性能如何，时间瓶颈在于密码学校验，而不是在于处理一些 if-then 条件语句。

足够的极限

不过，在一些场景中还是存在非常明显的整体开销的。第一个是在涉及较多的状态改变的操作过程中。当前，每一个状态的改变是以对 Merkle 树进行修改的形式实现的，这会触发 5-20 次反序列化操作，哈希和重新序列号操作，以及数据库的更新。就如之前描述的那样，Merkle 树若从轻客户端的友好性来说是协议的重要组成部分，若要在跨链运作中处理“收据”的话就更为重要了；不过，Merkle 树相关操作对效率的影响还是非常明显的——基准测试显示，在某些合约中，Merkle 树的更新会耗费很多的时间，若要验证签名的话花费的时间就更多了。

若要缓和这个影响，有两个途径。第一个解决方案是比特股团队推荐的，即完全去除 Merkle 树，简单地在 leveldb 数据库里存储状态。这对状态更新的效率大约会提升 5-20 倍左右，不过这是会对轻客户端的友好性带来影响的。对很多私有链的应用来说， 这或许是一个合适的做法。第二个方法，也是以太坊团队最可能在 1.1 版本的 Serenity 释出时跟进的办法，是一个折中的策略：目前 Merkle 树里允许 32 字节的键值对（key/value pairs），在将来会允许无限制

大小的值，让应用程序开发者可以在一个地方存储数据架构并对其进行频繁更新，这样就能够减少实际上更新 Merkle 树的次数。[10]

在以太坊虚拟机内使用高级的数学和密码学算法可能是第二个弱点。目前，在以太坊虚拟机内进行开发的可选方案有环签名，zk-SNARKs 零知识证明协议，RSA 式公钥加密算法，奇异值分解（singular value decomposition），甚至是一些对内存需求较高的哈希函数，如 scrypt。若要引入这些功能，以太坊虚拟机的性能在实际使用中可能较低：

- 用原生 Python 语言进行的椭圆曲线签名校验需要 0.017 秒；在 Python 版本的以太坊虚拟机中需要 0.57 秒
- 用原生 Python 语言进行的五把钥匙的环签名校验需要 0.119 秒；在 Python 版本的以太坊虚拟机里需要 3.68 秒
- g 用 Python 实施对内存需求较高的 scrypt 哈希函数的运算，会小于一秒；在 Python 版本的以太坊虚拟机里这个操作的过程实在是太长了，甚至需要分解到 118 个区块里才能执行这些交易。

这些情况是由两个方面的原因所导致的。第一，现有的虚拟机方案已经经历了几十年的开发，有不少优化的 JIT（just-in-time，即时技术）方案，而以太坊只有相对简单的 JIT 方案。若要让以太坊赶上现有的虚拟机的性能，可能需要几年的时间。其次，256 位整数的需求让虚拟机的速度大幅度降低，毕竟其底层的机器使用的是 64 位的架构，而这些算法的大部分代码只处理那些小于 64 位的整数，并不需要在每一个加法和乘法运算中使用整个 256 位的运算器及由此带来的整体开销。

在短期内，我们的解决方案一直是“预先编译的合约”。本质上说，如果应用过程中对某种特定的密码学功能有较高的需求（如 SHA3，SHA256，椭圆曲线签名校验等），我们就给每一种功能分配一个地址（当前我们用较低的地址，如 1，2，3 等），并增加一个协议上的规则，即进入这个地址的一个交易（或内部交易）消耗低额 Gas（有相应的标准），并输出一个与输入值相应的结果。这个功能并不是整合在以太坊虚拟机里；相反，它是以原生代码的形式整合到每一个

以太坊的客户端，让其运行成本更低及拥有原生代码的速度。这实质上是一个“急救绷带式的修复方案”，专门为特定的用例解决需求。不过，它显然不是最优的：这意味着对某类智能合约来说——那些带有稀奇古怪的、对运算资源需求较多的密码学算法的智能合约，其“无需许可的创新”可能就会受限于这个设计，它们无法像普通的智能合约那样享受任意的灵活性——普通的合约可以在不对协议层面进行更改的情况下书写任意的新功能。

WebAssembly ——“EVM 2.0”

基于这个原因，我们的一个研究者 Martin Becze，正在探索用 WebAssembly 作为实现更快的虚拟机的途径，以提供一个长期的可行解决方案。这个项目有两个步骤。第一步是书写一个 JIT 编译器，目标是将 EVM 的代码编译成 WebAssembly 的代码，这样能实现更快的 EVM 实施方案。第二步是使用 WebAssembly 去创建一个“以太坊虚拟机 2.0”，其对 WebAssembly 增加的功能只有一个，就是一个代码翻译器，可用于（i）插入 gas 计数操作，和（ii）禁止如下的一些操作代码（opcode）——那些被认为不太明确的、没有确定性的、或访问以太坊虚拟机本来就不应该访问的信息的。（如线程，浮点运算）。

基于几种原因，WebAssembly 被视为是一个理想的方案，这与 WebAssembly 和以太坊共享着相似的设计目标是有关联的。

- WebAssembly 致力于为小型的应用程序提供非常小的代码体积。一些更传统的技术方案（如由 C++生成的字节码）及其默认的工具往往会生成很大的代码尺寸——即使是那些微小的应用程序。（举例：参考这个开发者生成一个小型的 Linux 可执行文件的尝试）。这对今天的大体积硬盘来说或许不是个问题，但对一个区块链环境来说，可能会包含由几百万的用户上传的代码，这就带来了一定的负担了。
- WebAssembly 的设计目标是运行来自不明身份的人提供的未经信任的代码。
- WebAssembly 已经有多个 JIT 的实施方案，也有让它们都具备兼容性的目标。

- WebAssembly 若要用在我们所需的应用场所，其主要的不足之处会是缺乏 gas 计数的机制，和潜在的非确定性问题，而代码翻译器就是解决这些问题的。在理想中，用户可以使用（与 WebAssembly 编程所需的）相同的高级编程语言和开发工具去书写代码，而代码将会在区块链上执行——以一种跟 WebAssembly 环境里一样的方式去执行，只是代码翻译器层将会跟踪计算步数，并禁止非确定性的操作。初步的测试表明，gas 的计数在总体上只增加了 5%. 从初步的测试来看，WebAssembly 自身在不同的场合会有媲美于原生 C++代码的 25-90%效率——可以说对我们的需求而言，基本上足够拉近解释型语言与原生执行语言之间的差距了。

WebAssembly 的实施目前尚处于初始的阶段；已经有一个将 WebAssembly 代码转换成增加 gas 计数的概念验证模型在运行了，不过若要推出最终产品的话还需要较多的工作。生下来的工作包括测试 WebAssembly 的多个实施方案，或许会增加一个 Python 的实施方案以增加其完整性，并将其整合到客户端里；如果 WebAssembly 计划继续下去的话，将有可能被添加到 Serenity 版本与可扩展的以太坊 2.0 版本（后面的章节会提到）之间的一个版本里，如在 2017 年期间的某个时间。同时，整体上说，这些以太坊协议的改善方案将有可能先会应用到私有链上，因为在公有链上修改错误是更难的，这是一个安全性上的考量因素。

需要注意的是，这些解决方案旨在解决初步的效率问题。那些反对无限制的状态变更运算、可并行性的主要技术论点是很重要的；虽然在低吞吐量的情况下，运算到底是在一台或多台计算机上进行的问题并不是特别重要，但在高吞吐量的情况下，就如我们将会看到的那样，并行化是解决可扩展性的关键——甚至比效率更重要。

关键建议

以太坊虚拟机的效率在目前要比其他主流的虚拟机都低。这主要是因为在早期作出的“灵活性高于性能”的设计目标。对很多简单的应用来说，这并不是一个问题，毕竟相对于校验密码学签名的整体运算开销来说，处理一些加法、乘法

和 if-then 条件语句的整体运算开销可以忽略不计。不过，目前还是不适合将一些很复杂的运算（如定制化的密码学）直接整合到以太坊虚拟机里面。

在中期，以太坊的更新版本将有可能使用以下两种手段的组合：代码的预先处理过程，以及将 WebAssembly 作为创建更快的以太坊虚拟机的方案——其代码执行效率可以接近原生的代码。

在此之前，Serenity 的版本释出将会允许几种额外的、专用的密码学操作（以原生代码的速度运行），我们推荐那些想使用对运算资源需求较高算法的开发者先使用以太坊的私有链去扩展以太坊的“预先编译合约”的机制，在原生代码的层面通过预先编译的合约直接加入这些高级的操作——如果他们实在不想再等了。那些正在使用私有链，想获得更高度的可扩展性的用户，则可以考虑（i）立刻整合一些 Serenty 版本计划中的修改方案，以及（ii）若不考虑轻客户端的友好性的话，可以直接移除 Merkle 树。

可扩展性

可扩展性是每一种区块链技术都在面临的根本性挑战。若用传统的眼光去看，去中心化协议一种备受称赞，原因是它们拥有更高的可扩展性，让用户和公司以低成本、高效的方式分发文件，也能让应用程序去管理数百万用户之间的通讯，因此对非技术性的用户来说，他们对区块链的印象可能是，这种“点对点货币”（现在是“点对点智能合约”）应该有同样的好处。

然而，事实并不是这样。区块链的去中心化应用有着自身的独特特性——它的数据并不是以“在不同的参与方之间分割、存储数据”，而是“复制”的：网络中的每一个节点处理每一个交易，并维护着整个状态（译者注：可以理解为总账本）。这带来的结果是，区块链得到了去中心化技术的一些好处——特别是在容错和中立性方面，以及让每一个用户都能亲自校验区块链上的每一个交易而得到的极强的可校验性。但是，这也让它的可扩展性相对于传统系统来说更低了。实际上，不管区块链有多少个节点，它的交易处理性能始终无法超过单一的节点。还有，与 BitTorrent 这样的网络不同的是，BitTorrent 的网络性能会随着节点

的加入而提升，而区块链的性能永远不可能高于每一个节点的性能，还有，随着节点的增加及其带来的节点间通讯延迟，其性能可能还会更低。

公有链上面临的麻烦会更多。公有链上的每一个节点都必须处理整个网络的交易（这还意味着它让每一个人都能加入的设计目标，实质上意味着很多人就是用消费级的笔记本电脑加入网络的），而且我们也不可能期望每一个节点都会将100%的 CPU 运算资源贡献出来。这是有几个原因的。各个节点的 CPU 一直在同时运行其他应用程序，也不是总是在线的。其次，除了设备的限制外，公有链还需要面对经济激励上的限制：如果维护节点的成本显著高于节点的总成本，那么网络就倾向于走向中心化，即两个节点基于经济因素的考量，可能会互相信任并最终只运行一个节点[11]。还有，节点可以简单地停止校验网络上的校验，并直接享受别人提供的校验服务——这是一个明显的“公地悲剧”，若很多矿工同时尝试这样的做法，就可能会带来很大的系统性风险。中心化和不参与校验的均势是必须防止的失败模式，而且必须考虑到安全性上的误差，毕竟当一个系统进入一个不利的均势，就很难改变规则或强制推倒重来了。

基于这些原因，虽然 C++版本的以太坊节点理论上的交易处理性能是超过1000 交易/秒，但以太坊网络的区块 gas 限制带来了一个软性的上限——10 个交易/秒（假设这是简单的交易，那些复杂的交易更接近 1-5 个交易/秒，而当前在网络中实测的结果是接近 6.8 交易/秒）。比特币若单从 CPU 限制的考虑能实现4000 交易/秒的性能，但现在实际上有一个 7 个交易/秒的上限（还只是对小型交易而言），实测中约等于 3 个交易/秒[12]。私有链并没有这些方面的忧虑，因为它们可以要求每一个节点配置高性能的计算机和网络条件，并在现实世界中确保参与者在运行一个真实的节点；因此，基于比特币和以太坊的私有链可以轻松实现 1000 个交易/秒。因此，在短期内，对很多企业用户来说，私有链可能是最适合的、最实际的选项——若他们要将区块链用于主流的用例并达到足够可扩展性的话，不过在大规模使用的场景中可能还是不够的：DTCC 现在每天处理 1 亿次操作（约等于每秒 1200 次），而上海证券交易所的交易系统订单处理峰值可超过 8 万笔/秒，至于那些区块链推广者们认为前途广泛的小微交易应用，需要的交易性能就更高了。

解决方案

为解决可扩展性面临的挑战，我们在探索两个类别的解决方案：sharding（分片）和 state channels（状态通道）。Sharding 技术与数据库分片的技术相似，状态的不同部分会由不同的节点去存储，交易会被引导到不同的节点——取决于这些节点负责哪个分片，因此这些分片可以被同时处理；而状态通道是比特币模式的“闪电网络”的一个广义方案，它寻求将大部分的交易在参与方之间直接进行，而不在区块链上进行，最终只使用区块链作为最终的争议仲裁手段。（现实问题：Dominic William 的 dfinity/pebble 提议使用了这类分片技术吗？）

在前一个方案中，私有链和公有链的分片用例有着不同的考虑因素。在私有链分片的例子里，目标是中规中矩的：让以太坊最大程度地并行化。每一个节点始终要处理每一笔交易，唯一不同的地方是当运算进行并行化了，网络的可扩展性可以通过增加每一个节点的 CPU 的核心数据及内存去扩展。

在公有链的案例中，目标就更有雄心壮志了：创造一个区块链协议，其网络的完全节点（译者注：完全节点即存储完整的总账本的节点）数量即使为 0，也能生存下去——即创造一个网络，其中的每一个节点只处理所有交易的一小部分，并用“轻客户端”技术去访问区块链的其他部分，同时保留了安全性。在这种模式下，交易不仅在不同的 CPU 核心上处理，而且会在不同的计算机上处理，就如公有链加密货币经济学的分析[13]那样，并不需要相信每一个节点。以太坊 2.0 和 3.0 的长期计划，是将协议改造成一个可运行 VISA 支付网络级别的区块链，甚至是其几个级别之上的性能——这还不需要特别的机器，只是由运行在商品级的笔记本电脑所构成的网络上。[14]

为解决这个问题，我们的整体方案是从实施一个并行化计划出发（通过以太坊改善提议 EIP105，这是在 Serenity 版本里安排的），这会提供私有链方案所需的可扩展性特性，以及公有链上的中规中矩的改进，在此之后的以太坊 2.0 版本里将会包含一个密码学与经济学相关设计的计划，以通过分片技术为公有链增加安全可靠的可扩展性。EIP105 及一些为 Sernity 版本而设的特性已经被整合到一个 Python 语言的概念验证中；2016 年的末期可能会推出测试链，2016

年的早期可能会有一个实测的网络，当然了时间上的推迟是有可能的，我们的指导思想是只在产品就绪的时候发布，并不会冲着计划表发布。在这份路线图的前半部分，主要的挑战是开发工作，不过在日后的阶段会涉及与经济激励以及点对点网络架构相关的研究任务。

私有链里的并行化

总体上说，如果计算任务可以被完全的并行化，私有链的可扩展性就会是无限的：你总是可以往每一个节点里添加更多的 CPU 核心数量。若事实不是这样，则最大的理论速率就受限于安达尔定理(Amdahl’s law)：你能给系统进行提速的极限取决于那些不能进行并行化的部分的倒数。如果你可以进行 99%的并行化，那么你就可以提速到 100 倍，但如果你只能实现 95%的并行化，那么你就只能提速到 20 倍（译者注：在本例中，若有 99%的并行化，则有 1%是不能并行化的，那么 1%的倒数即 100，所以得出可提速的极限是 100 倍，95%的例子同理）。我们可以给一个更精确的描述：假设 CPU 时钟的速度是固定的，处理一批交易所需的理论最小时间跟交易执行的依赖关系图里最长链条的长度是成比例的;即最长链条里面的交易在理论上会有相互的影响，因此必须按顺序执行。

多重签名合约（multisig-oracle）的智能合约执行模式的最大功能或许是——每一个合约都有自己的一系列“公证人”，单独地就该合约的执行结果进行投票，而不考虑其他合约的事务。彼此之间没有协同工作，这就意味着没有彼此间的依赖性，因此寻找将底层的资产管理层或数据层进行并行化的方案就变得相对简单了[15]。以太坊里的循环与协同的模式带来了这样的挑战：理论上，并不存在这样的一个限制，去明确“状态”中哪些数据是可以被合约执行过程所依赖的，因此将计算任务按照序列的方式去运行可能就是我们的现实方案了。[16]

因此，若我们想通过 sharding 技术实现可扩展性，我们就需要对以太坊的执行模型作出一些妥协，让其能进行并行化，来自传统并行计算理论的经验就很有用了。我们不再将整个状态构造为单一的机器，而是将其作为分布式存储机，在里面交易被限制为只处理状态的一部分，这样它们就可以进行并行处理了。若要实现分片间的通讯功能，我们需要添加消息传递机制；这里面最主要的挑战就

是要确保它具有确定性。我们正采用的一个手段，它可以被用于公有链和私有链的场景中，那就是一个“收条”的范式：交易的执行过程可以改变局部分片的状态，不过它同时会生成“收条”，被存储于一种共享的存储空间里面，以后可以通过其他分片的交易执行过程被审阅（但不能被修改）。

若要观察一下这种技术如何在现实中应用，可以在较高的层次上分析一种数字代币例子。用户 A 持有 500 单元的 X 代币；这个信息是存储在分片 M 里面的。用户 B 持有 100 单元的代币 X；这个信息是存储在分片 N 里面的。假设用户 A 想从分片 M 中发送 100 单元的代币 X 到分片 N 里。在一个同步的模式里，所有的分片是被全部节点存储的，这实现起来会很简单；就如以太坊代码的一个例子——这个例子经常在幻灯片和我们开发者的 T 恤背后可以看到，代码会很像这样：

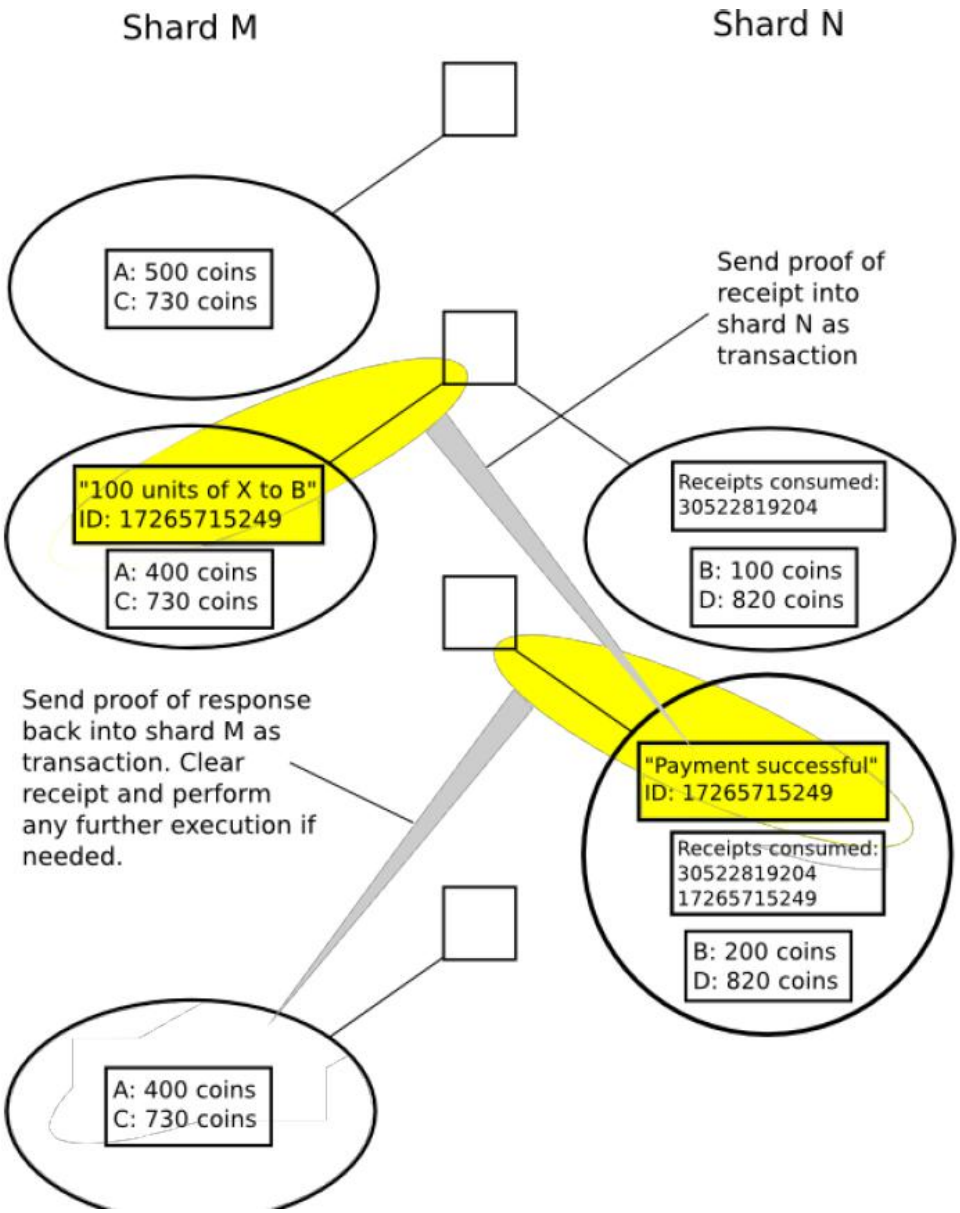
```
def transfer(to, value):
    if self.balances[msg.sender] >= value:
        self.balances[msg.sender] -= value
        self.balances[to] += value
```

在一个异步的模式中，我们就需要将这些过程分成几个阶段：

1、 在分片 M 里，把 A 的余额减少 100，并生成一个收条，称“B 应该接收到 100 单元的 X 代币。这个收条的 ID 号是 17265715249。“

2、 在分片 N 里会以一个交易的形式引入该收条被创建的证据（在私有链的例子会是指向共享存储模块的一个指针，而在公有链的例子里则会是 Merkle 的证据）。这个合约会对以下细节进行校验：（i）该证据是正确的，而且（ii）未曾消费过具有相同 ID 号的收条。这个合约给 B 的余额增加了 100，并在存储区里标记上该收条及其 ID 号已经被消耗的事实。

3、 在更高级的应用中，若分片 M 在运行一个智能合约，它要在付款完成后执行一些操作，这样分片 N 中的记录就能用作一个收据，提供给分片 M 并证明付款已经完成，这样分片 M 就可以继续它要进行的操作。



若被消耗过的收条记录不断增加，其解决方案可以是给收条设置一个超时参数（例如几个星期），并将那些太旧的被消耗收条列表清除掉。

收条的模式是很方便的，因为它对应了两种开发者们很熟悉的编程概念：异步编程和 UTXOs（比特币里面的“未花费的交易输出”的概念）。在比特币里面，交易运行的方式是消耗一堆由之前的交易创造出来的 UXTOs，然后生成一个或多个新的 UTXOs，这些新生成的 UTXOs 也能被将来的交易所消耗掉。每一个 UTXO 可以视为是一种“币”：它有一个面额，有一个持有者，而一个有效交易的两个主要规则是：（i）该交易必须包含它消耗的每一个 UTXO 的持有者的签名，而且（ii）UTXOs 被消耗的总面额必须大于或等于该交易产生的 UTXO 面额的总量。

虽然 UTXO 的模式在某种程度上是很笨拙的，它需要复杂的钱包代码去决定用户消耗 UTXO 的顺序，以优化交易费，并让那些没有仔细实施该技术的钱包暴露到拒绝服务攻击之中，不过它在某种程度上也提高了对用户的隐私保护水平（因为用户的资金并不能立刻与其他人的资金联系起来），也提供了一定程度的可并行性，也让在其内扩展一些密码学技术成为可能（这会在“隐私保护”的章节进行讨论）。收条是由交易执行过程所产生的对象，可以被将来交易的执行所“消耗”，这与 UTXOs 有相似之处；因此，它可以被理解成用图灵完备智能合约实现的广义、概况化的 UTXOs 模式。

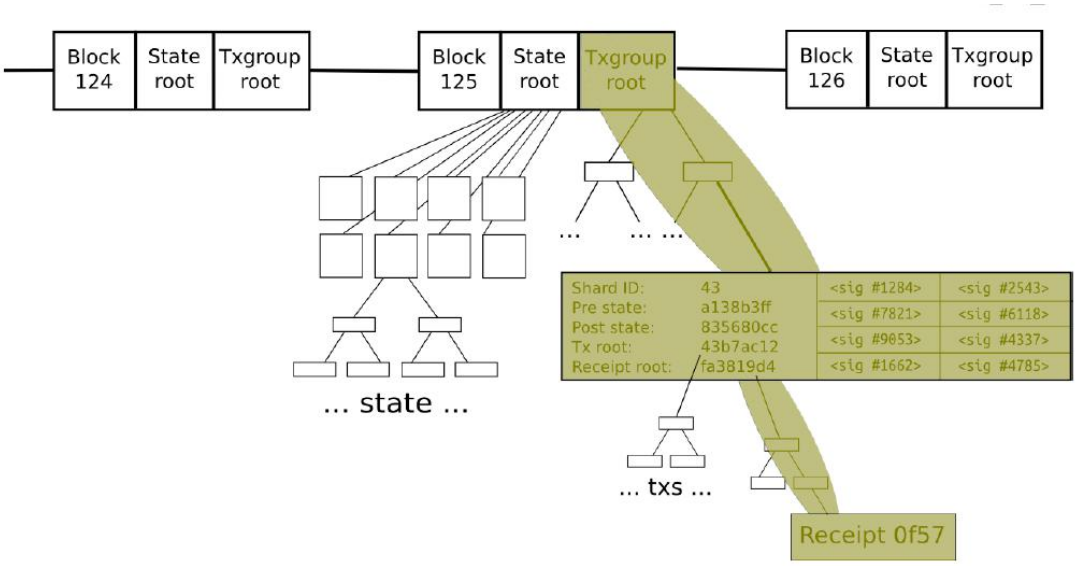
收条模式与异步编程的联系是挺明显的。若分片 M 里的函数 A 用异步的方式调用分片 N 里的函数 B，这可以通过如下的方式实现：用代码执行函数 A，并生成一个收条（内含提供给函数 B 的参数），之后执行函数 B 的交易可以消耗这个收条。若需要一个回调函数，则当调用函数 B 的时候，函数 A 可以在状态中留下一个对象，并声明它在等待来自函数 B 的回复，这个对象迟些就会被由函数 B 生成的收条所消耗，并得到一个回复。

需要注意的是，在私有链的场景中，一旦交易被处理后，收条就可以即时在所有的分片中进行处理，这个过程会持续下去，直到收条生成和执行的完全链条被执行完成。由于收条的异步特性，这个过程还是可以被高度的并行化的：在每一个分片中的交易可以并行处理，进入每一个分片的收条也可以并行处理，如此类推；因此，只要在一个分片内没有太多的交易在同时进行，并且收条-回调的链条的长度有一定的控制，那么就能完全解决并行化的问题。

从私有链到公有链

在公有链的场景中，我们会面对两个额外的问题。第一，在不同的分片内处理交易执行的节点是在地理上被分割的，在不同的计算机上运行；这就导致了处理收条的工作并不是即时的，一个复杂的异步合约执行可能需要超过一分钟才能执行完毕。其次，这些计算机并没有相互信任的关系。因此，一个在分片 M 内处理交易的节点并不能简单地通知分片 N 内处理交易的节点并声称“我已经创建了

一个收条了”；而是要证明给它们看。幸好，Merkle 证明可以是一个理想的解决方案。



在以太坊 2.0 的公有链中，设计目标是构造一个系统，其中的状态集合是分布在在网络中的不同节点上的，每一个节点只持有状态的一小部分“分片”。所有的节点都会跟踪一个“header 链”，里面包含了状态的 Merkle 树根的哈希值。因此，可以很轻松地整合一个收条方案：当分片 M 创造了一个收条后，一个 Merkle 分支（如一系列用于证明该收条存在的哈希值链条）可以作为一个进入分片 N 的交易进行传递，然后若需要一个回调，则可以创建另一个收条，并返回另一个 Merkle 分支。

下一个问题是决定由谁去校验每一个分片。当然了，分片技术的目的就是要舍弃“每个人处理一切事情”的范式，并在很多节点间分割校验的责任。原生的工作量证明机制若要安全实现这个技术会比较困难：比特币整合的工作量证明机制是完全匿名的（连假名机制都没有）[17]共识机制，若某个分片只由总算力中的一小部分去维持其安全性，那么攻击者可以调用他们的算力专门去攻击这个分片，这样就有机会用不到 1%的全网算力去攻击整个区块链。

其实，这个状况在以下的一些解决方案中会发生变化：股权证明机制，如 puzzle towers 这样的改进版工作量证明机制，以及在私有链里使用的拜占庭容错共识算法[18]：在共识处理过程的参与者们会有某种身份，即使是类似地址的密

码学化名身份，因此我们可以通过随机采样方案解决“定向攻击”的问题，即从整个验证节点池里面随机选择某些节点的集合去处理任意给定的交易集合，这样攻击者就几乎无法定向攻击任何特定的交易或特定的分片。还有，恶意和渎职行为是很容易被打击和管制的；就如 Vlad Zamfir 所说的那样，“若股权证明机制里某个节点挖出了一个无效的区块，其后果就如同挖矿工厂被烧掉一样”。[19]

因此，公有链的场景中，整体的思路是首先将交易收集到一个各自独立的“交易组”里面，然后让每一个交易组选择一个随机的验证节点去进行验证。“header chain”自身不再校验全部的”交易组”，而只是校验某种校验凭证——这个凭证会随着交易组提供，它包含来自特定集合的、具有足够数量的大多数验证节点的签名。

将分片技术应用到区块链应用中

现在还有两个问题。第一，状态应该怎么进行分片？在公有链的场景中，这可以进一步分解成“应用程序间的分片”，目标是将状态以某种形式进行分片，从而让多个在大多数情况下独立的应用程序以并行的方式处理；还有“应用程序内的分片”，即应用程序被分割到多个分片中。第一个问题或许比较简单，毕竟应用程序间的互动往往是低频率的，因此最简单的解决方案可以是通过地址空间进行分片。在私有链的场景中，区块链往往只被用于一种应用，因此“应用程序内的分片”在这种场合会更有优势。其次，我们如何能设计出高级的编程语言，去利用区块链的应用程序内分片的潜能，并将其优势最大限度地呈现给开发者们？

从分片的角度来看，面临着如下的挑战：我们如何创建一个分片方案，同时可以提供应用程序内和应用程序外的分片功能，而不增加太多的复杂性？当前，我们解决这个问题的主要提议是 Serenity EIP105 提议，其主要思想如下：首先，地址空间被拆分为 65536 个分片，只有在同一个分片内才能进行同步的交易执行；其他的则只能以异步的方式执行。其次，一个具有给定片段的代码可以以相同的地址存在在多个分片中，让程序可以同时在多个分片之间存在。合约或许可以采

用在传统的并行计算场景里常见的分区技术（如 SQL 查询），去决定如何存储它们的数据：通过某些特定钥匙的特性属性进行分区，随机向分区里分配新的对象，或许一些更智能的负载均衡算法等。在 Solidity 里面，可行的方案可以是定义一个 shardedMap 数据结构，在编译的时候定义一个将钥匙映射到分片 ID 的方案。我们预期高级语言的开发者探索以下技术的组合：SQL 风格的分片方案，异步编程概念——如回调和 promises，以及其他的技术，从而让开发者更容易定制一个可扩展的区块链环境。

对特定的区块链应用来说，最简单的分片方式是存在证明（使用区块链去展示特定的数据片段在特定的时间存在）。这种用例并不需要担心双重支付的问题，而且已经被 Factom（公证通）和其他项目以一种可扩展的方式在使用。“不存在证明”，即证明一个“存在证明信息”还没有被发布到区块链上（如证书撤回），就稍微要难一点了，因为你无法简单地使用 Factom 方案里的“只发布 Merkle 树根”的手段去解决它，不过还是可以最大限度地并行化的。[20]

在数字资产的用例中可能会更难处理，但还是能通过异步编程实现的，就如上面提到过的那样；其中唯一的效率问题是付款发出和收款人能实际使用之间存在轻微的时间差。双方（或多方）智能合约跟数字资产并没有明显的差别，也面临着同类的限制。诸如金融领域的身份验证这类应用程序可能会涉及数字资产转账系统和身份管理系统之间的互操作过程；在这里异步处理并不是太大的问题。原子化交易实施方案将需要处理一些特殊的案例，即双方尝试同时参与一笔交易；锁定机制和由合约生成退款交易的机制或许是处理这个问题最显然的策略了。那些基于区块链技术并实时处理订单薄系统的市场看上去是最难处理的；比特股开发团队最近得出结论，若不引入多个市场及随之而来的套利空间，是无法应用此技术的。至于类似高频批量竞价机制市场这样的非即时市场或许可以使用并行排序算法。（若这是真的话，这对来自 Overstock 的 TO/Medici 及同类项目有什么影响？）

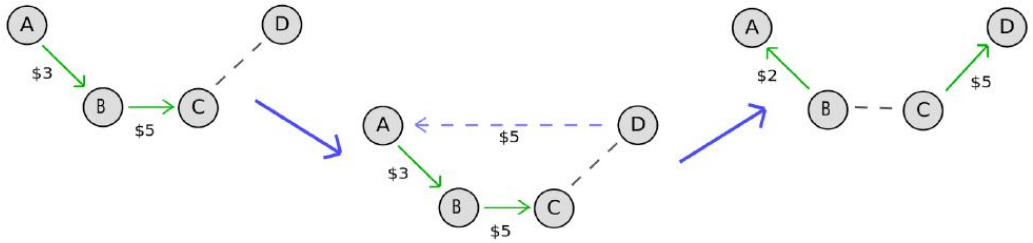
状态通道

States channels（状态通道）是这样的一种策略，它保留底层的区块链协议运作的模式，但改变协议的具体用法，以解决可扩展性的挑战：它将区块链作为处理任何形式交易的主要处理层，而是作为一个结算层——只处理一系列互动所产生的最终交易，并只在出现争议的时候执行复杂的运算操作。

为了更细致地了解这个概念，我们先来考虑一下状态通道用在支付系统的例子。假设 A 有 100 个币，想以较高的频率支付给 B。这样，A 开始将 100 个币发送到智能合约里面，当 A 想执行其对 B 的第一笔付款时（例如 0.1 个币），他创建一个有数字签名的“凭证”（这并不是一个有效的区块链交易），该“凭证”上声明（99.9，0.1，0），在这里面，99.9 表示他本人希望得到合约里的多少币，0.1 表示 B 应该得到多少个币，而 0 是一个序列号。而 B 接着进行会签（counter-sign)。若 A 希望再支付 0.2 个币，他就签发一个新的凭证，声明（99.7，0.3，1），同理 B 又进行会签。现在若 B 想向 A 返还 0.05 个币，她就应该签发一个凭证，声明（99.75，0.25，2），然后 A 进行会签。

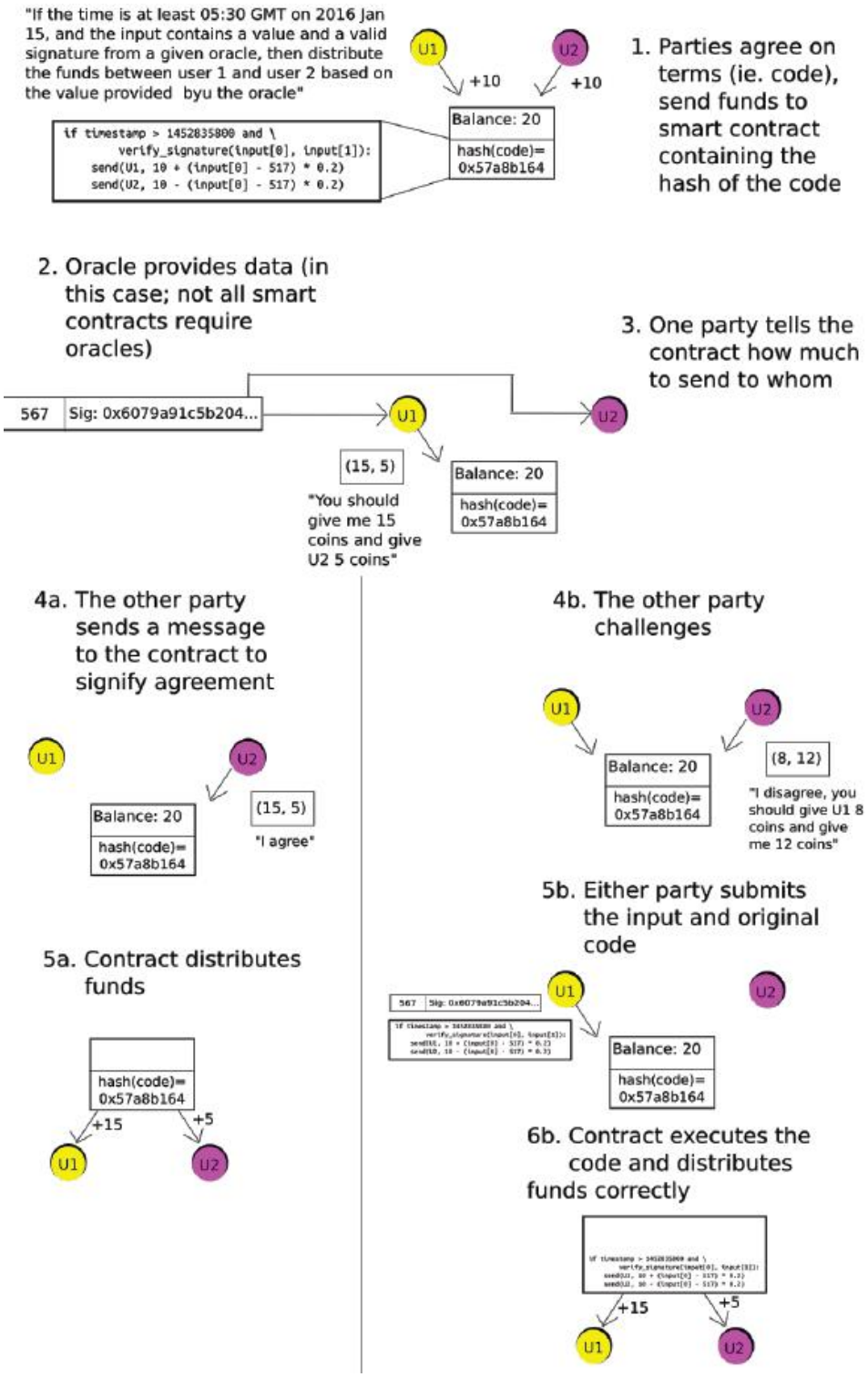
在这个过程中，每一个参与方都可以往这个合约中发送一个交易，从而关闭这个通道，并开始一个结算的过程。这会启动一个时间区间，在这期间 A 或 B 可以提交凭证，具有最大序列号的凭证将会被处理。如果 A 故意提交一个更早序列号的凭证，则 B 可以自己提交具有最大序序列号的凭证。这样，区块链就可以被视为某种“密码学法庭”：作为一种成本较高的仲裁手段，它是一种用于提供安全性的最后手段，因此也降低了造假和欺诈的动机，当然了，如果这个过程运行得很好，那么区块链只会偶尔被用来做最终结算。

这种简单的支付通道机制带来了两种归纳结论。首先，如果你想支持任何人之间的大规模支付活动，而不只是 A 与 B 之间的话，你也不想为每一个人创造“N 的平方”这么多数量的通道，那么你就可以“组合”通道：如果有一个 A 与 B，B 与 C，C 与 D 之间的通道，那么你就可以将 A 到 D 之间的转账变成 3 条通道的更新过程，每条通道更新一次。[21]



其次，可以加入智能合约。如果 A 和 B 想达成一个金融合约，这个合约根据某个公式进行转账，这样他们可以都签发一个凭证（H（C），k），在这里 H（C）是一段代码的哈希值，而 K 是最近的序列号；这实质上是告诉状态结算合约“评估进行了哈希运算的 C 的代码，并让结果告诉你应该向 A 和 B 各发多少钱”。举例来说，C 可以是差价合约这类金融合约；如果状态通道控制着多种资产，它可以是一大类不同类别交易里的一种，如有杠杆的交易抵押品管理协议，期权等。

当到了从通道中取出资产的时候，A 和 B 可以运行 C，由此都知道他们各自应该得到多少钱，区块链就不需要评估这个合约了；若 C 给 A 发送了 75 个币，给 B 发了 25 个币，为了方便起见，他们可以都签发一个新的凭证（75,25，k+1），若任意一方拒绝执行，则另一方可以简单地提交(H(C)，k)以及 C，并让区块链进行运算以及引出同样的结果。



状态通道并不是一个完美的解决方案；具体来说，还不清楚它能怎么适用于大规模的多用户应用场景，它也没有给原来的区块链在可扩展性上带来多少改善（从存储更大状态体积的角度去考虑）——它只增加了实际上的交易吞吐量。不过，它也有不少的好处，除了作为一种可扩展性解决方案外，它也是一个隐私保

护方案，因为区块链在这个过程中看不到中间的支付或合约信息，除了最终的结算和争议解决过程；它也是一个解决延时问题的方案，因为两方之间的通道更新几乎是即时的——比任何区块链上的解决方案都快（无论是公有链还是私有链），甚至可能比中心化的解决方案都快——因为A与B之间的通道更新无需中心化服务器也能实现。[22]

状态通道理论上不需要修改以太坊的协议也能实施，如Raiden这样的团队已经在将状态通道网络整合到以太坊上了；Jeff Coleman和其他人正在研究往以太坊上整合状态通道的最佳方法。因此，这可能是在短期内最安全的可扩展性解决方案了（以及隐私保护方案，在下面的章节会提及）。

关键建议

在短期内，关心可扩展性的机构有两个选项：（i）不管是在公有链还是私有链上，探索将大部分的应用程序逻辑放到状态通道里面的做法，还有（ii）在一个以太坊私有链版本里实施EIP 105计划，并使用异步编程及分片机制去实现并行化。在使用这个方法之前，用户应该先决定他们的应用程序是否能在当前的单线程以太坊实例中运行；性能需求低于2000交易/秒的应用程序并不需要并行化的特性，现在就能在以太坊的私有链上实施了。若不采用状态通道的话，公有链当前的可扩展性并不适用于大部分的金融应用程序，毕竟以太坊的公有链只能安全地支持10-20交易/秒。

隐私保护

以太坊及区块链应用程序面临的另一个主要挑战是隐私保护问题。这个问题很简单：多个节点验证和审计系统中的所有交易，保证了记录的真实性，但也带来了隐私保护上的问题。在一些案例中，很多区块链用户对这点理解不够深入：很多公司刚开始关注区块链，是因为他们听说区块链是一项极佳的信息安全技术，不过要经历一段时间后他们才能理解区块链提供的是信息的真实性，而非隐私性，而他们概念中的“信息安全”指的是隐私保护。

首先，最重要的是要明白私有链并不是一个隐私性的解决方案（除非区块链只有一个节点，这样……它是否还能叫区块链就有一定的争议了）。一个具备 N 个节点的区块链的容错性可以达到 N/3 甚至更高，取决于你选择的网络模型，若从可验真性的角度看有 N/2 的容错性，但从隐私保护的角度看，即使是少量的失误也可能导致很严重的后果。原因很简单：任何拥有该数据的节点可以发布该数据。如果你对信息安全的主要考虑是隐私保护，而且你没兴趣了解高级的密码学或最起码的密码学与经济学相关设计，如状态通道，那么你可能适合用一个中心化的服务器而非区块链去实现。

区块链隐私保护相关的解决方案通常有两种：低科技的解决方案，即简单地依赖密码学哈希运算、签名和密码学与经济学相关设计去减少信息暴露及区块链上信息间关联性被解开的可能性。高科技的解决方案，即使用高级的密码学技术，去提供非常强大、在数学上可证明的隐私保护特性，让区块链可以公开地处理交易和合约，但混淆其内容，这样外人无法进行解密。两种策略都有各自的局限：低科技的方案更容易实施，但其隐私保护程度是“统计学意义上”的。而高科技的方案更强大，但开发起来会更困难，而且可能要依赖于一些并没经过仔细测试的安全性假设。

了解相关的目标

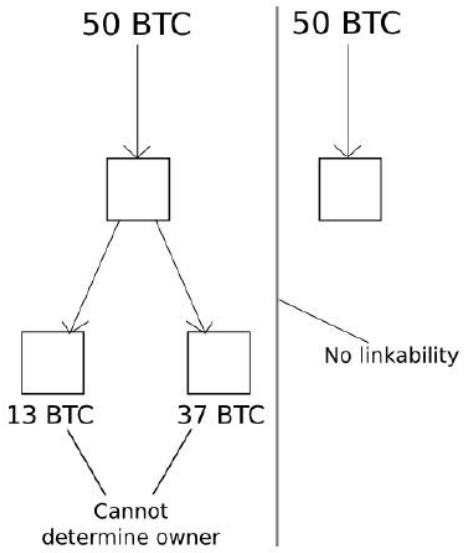
在开始为公有链或私有链研发一个基于密码学的隐私保护方案前，必须先了解具体的需求。在公有链的共同目标就是尽量隐藏信息。理论上“黑盒”区块链是一个理想方案，即使用密码学技术去混淆整个状态及状态变换的规则，因此用户可以发送加密的交易并读取与他们相关的信息，但不能解密其他信息。实际上能实现这项方案的密码学混淆技术理论上是存在的，但当前效率非常低，几乎是不实用的。最近发表的一篇论文里预计“（若要用上这项技术）在同样的 CPU 上执行一个 2 位的乘数需要 1.3*108 年”。不过，有一些稍微弱一点的密码学可以提供更实际的隐私保护方案，可以为特定的数据或元数据提供完全的隐私性保证。这些机制会在接下来的章节介绍。

在私有链，或公有链里的很多类型的金融应用案例中，需求是更具体的：在特定的案例中，监管方需要得到相关的信息。同时，金融机构里面也有了解客户个人信息的需求，最起码是要确定该客户的个人信息包含特定的属性（如确定他们的用户不是美国公民或居民，这种要求对银行和金融初创企业来说是很常见的，因为这可以降低监管风险及合规成本）。其他需求可能还会基于数额，如向相关的金融监管机构汇报超过 10000 美元的大额交易是一个常见的要求。

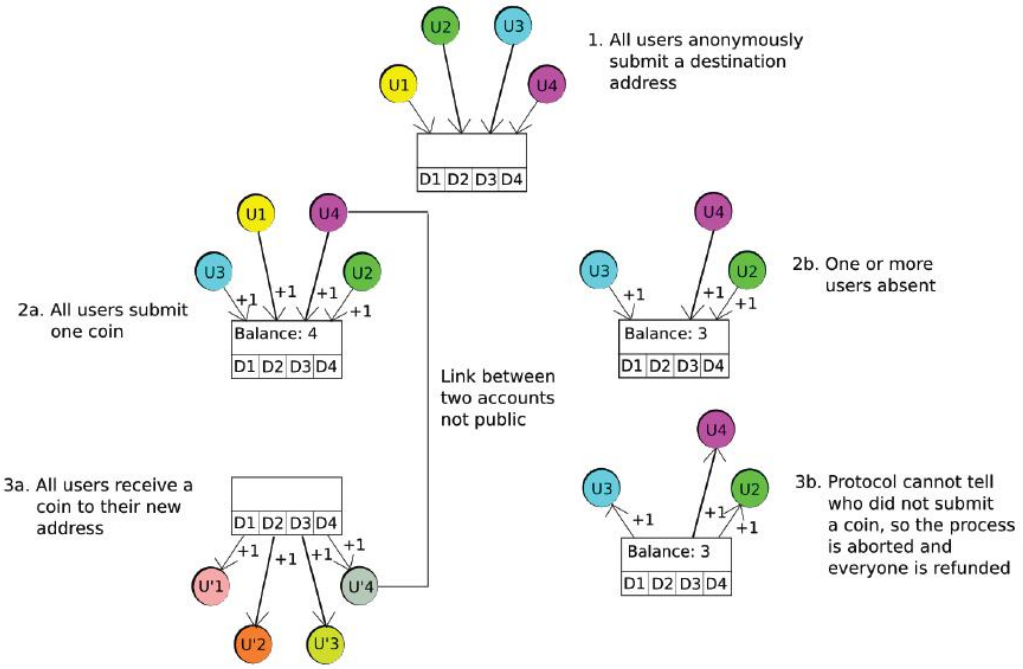
处理这些要求的“基础案例”可以是简单地在区块链上发布与每一个用户账户相关的身份信息，让应用程序在于账户互动前先检查这些信息。不过，这显然是透露得太多了；DTCC 最近发布的一篇文章提醒，“与身份相关的信息并不适合存储在一个去中心化的账本里，除非该项技术走向成熟并证明它能抵御攻击”。因此，这带来了一个挑战——是否有一种方法既能满足监管者与其他信息上的需求，又能在同时避免透露过多的信息？

低科技的解决方案

在低科技的范围里，过去 10 年间有不少令人瞩目的解决方案被设计出来了，这可以用于提高公有链支付系统的隐私保护中（还有更通用的资产转让）用户案例。最简单的策略是为每一个操作设置一个一次性的账号，这样单个用户的活动无法彼此联系在一起，然后引入一种所谓“避免合并付款”的机制，极大地降低用户活动的可关联性。[23]



CoinJoin 是一种更高级的策略，它创建了一种区块链上的协议，N 个参与方在区块链上合并和重新分割他们的资产，这样外人在查阅区块链数据时无法得知输入项与输出项之间的对应关系。



对智能合约用例来说（更准确地说是双方或多方金融合约），就如前面可扩展性章节里讨论过的那样，状态通道是目前最佳的低科技的隐私保护方案：除非发生纷争，或合约的一方失踪，其他人无需知道任何合约的细节，甚至无需知道该合约存在过。

若特定的机构需要了解账户被谁拥有，有几个方法可以满足其需求。最简单方法或许是要求每个账户得到特定的 “KYC 授权机构”的授权；这个授权机构就能得知每个账户后的现实身份，也能通过避免合并付款交易链和 CoinJoin 实例去跟踪资金的流动，而其他人就看到具有“半匿名”特性的交易输出链条。识别信息可以通过 Merkle 树发布在区块链上：KYC 授权机构可以签发 Merkle 树的根值，若某个用户希望向一个应用程序证明特定的信息（如姓名，年龄，ID 号码，公民身份状态），那么他们可以向程序提供 Merkle 树的分支信息，而无需透露其他的信息。

实施这种方案面临的最大挑战是法律上的，而不是技术上的；就如前面章节提到过的那样，金融应用案例通常需要执行它们自己的 KYC 政策，而不能简单地依赖第三方提供的服务。不过，这种挑战可能与特定的案例是高度相关的，而很多应用案例中，可能直接就可以成为软件提供商，从而避免了这些方面的忧虑，并将资产的托管权交给那些需要执行复杂验证工作的一系列机构去进行。

高科技的解决方案

从高科技的角度看，有三种最佳的技术可以用于以太坊平台上，这就是环签名，ZK-SNARKs 零知识证明以及秘钥共享技术。

环签名技术是一种特殊类型的密码学签名，它能够证明签名者拥有一个对应特定集合公钥的私钥，而不会暴露出签名者的身份（该私钥对应哪个公钥）。另外，还有一个更高级的版本，即可链接的环签名，它有一个额外的特性：如果你用同样的私钥签名两次，这个事实就能被检测到——但不会暴露出其它的信息。总体上说，环签名技术可以用于证明签名者属于某个集合里面的成员之一：例如，某种特定服务的授权用户构成了一个集合，一个人可以用这项技术证明自己属于这个集合里面的一员，但不需要暴露出自己的身份。可链接的环签名技术在区块链场景中可能是最有用的，因为可链接的特性增加了抵抗双重支付的关键能力：在保留完整的隐私性的前提下，单个参与者不能进行双重支付——如果他们这样做了，就会被查出来。

这项技术的一个天然的应用，就是为区块链资产管理而设的简化隐私保护计划。例如，开发者可以整合一个简化版的 CoinJoin 方案，在这个方案中，用户只需要往智能合约里发送一个单元的某种资产，在满足以下两种情况时可以将这些资产提取到另一个账号中——提供一个可链接的环签名，以证明（i）他们是其中一个存款者，而且（ii）他们之前没有提供过一个签名，也没有进行过提款。[24]

这项技术若用于现有的身份管理平台之上，就是作为一个安全的、带有隐私保护机制的身份唯一性（一一对应）认证方案。例如，若监管者允许金融平台在无需校验身份信息的情况下开放小额账户（如 200 美元以下）的存款和转账权限，

但对超出这个数额的账号会要求提供身份验证信息。在现在，电话号码通常可以作为一种身份唯一性认证手段，不过它的缺点是（i）它不能保护隐私，而且（ii）电话号码与身份之间并不是一一对应的。有一个潜在的解决方案（假设存在某种密码学数字身份管理方案[25]）就是让用户使用可链接的环签名去证明（i）他们是一名授权用户，而且（ii）他们还没有开设一个账户，这样的话他们就被允许开设一个带有 199 美元限额的账号。用户只有在提供更多身份信息的情况下才能将账户升级到更高的限额。隐私性的特性并不需要是绝对的；具体来说会根据限额的不同设定相应的界限，低于该界限时只需提供少量的信息，而高于该界限时需要提供更多的信息，而可链接的环签名技术可以用于保护涉及更高等级信息的隐私权。

加法同态加密是可链接环签名技术的天然搭档。加法同态加密技术背后的概念如下：你可以为原始值进行加密，若你将 x 和 y 这两个值加密生成 e(x)和 e(y)，第三方可以在不需要知道 x、y 的值或任何隐私信息的情况下，计算出 e(x+y)的值。这个技术与密码学概念“范围证明”（Range Proofs）结合起来，被 Greg Maxwell 和其他人扩展成为机密交易（confidential transactions）方案，这个方案能对区块链上的所有的交易值和账户余额进行加密，用户只能看到自己账户的余额以及进入自己账户的交易相关值，同时能够保证账目的正确性（如交易结果是零和的，一个人的收入必然对应另一个人的支出，任何人不能凭空创造出新的货币，最重要的是任何账户的余额都不会是负数）；令人惊讶的是，从数学的角度看，最后一个要求给开发类似方案的工作带来了很多困难。[26]

这样的一个机密保护方案在理论上可以与状态通道技术一起，创造一个具有高度隐私保护性能的智能合约平台：用户可以进入 State Channels，其互动过程不会被其他用户看到，最终就一个新的集合的余额达成共识，与此同时旧的余额和新的余额是被完全加密的，而公众能够校验的结论是（i）余额的改变是零和的结果，而且（ii）所有的新余额必须是非负数的。可链接的环签名技术可以用于保护身份，加法同态加密技术和范围证明技术可以保护余额的机密性，而 State Channels 可以在带来该信息改变的互动过程中保护其安全性和隐私性。

若有某些中心化的参与方提出监管方面的需求，这个特性可以整合到用户允许使用的智能合约集合里面。[27]

ZK-SNARKs 是一个非常强大的密码学原语，其定义如下：ZKSNARKs 是一个密码学的证明，里面有一个私有的输入项 I，这个输入项只为证明者所知，另有一些公开的程序 P，以及公开的值 O，可得 P(I)=O，而不需要透露出输入项 I 的值。要了解这个原语可实现的功能，可以参考一下身份验证这种特定的场景。通过 ZK-SNARKs，你可以提供诸如“这个密码学钥匙的持有者拥有一个由 X 机构发放的数字 ID 证书，至少有 19 岁，是 Y 国的公民，并有一个驾照”这种信息的密码学证明，而不需要透露出有关你身份的信息。[28]

ZK-SNARKs 可用于资产转让业务场景的隐私保护方案中，实质上是隐藏了交易之间的关联，也能为智能合约创造更好的隐私保护性能。一个叫 Hawk 的项目引入了一种“私有合约”的概念，它的可校验性并不依赖于某些个人或机构，而需要一个参与方确保其隐私性，在这种模式中的“管理者”的任务是负责更新加密后的智能合约的状态，并证明这些更新是有效的。这个管理者会看到所有的数据，不过即使这个管理者失职，其他的参与者也可以继续进行互动（这个过程可能会导致隐私信息的泄露，但这只针对该特定互动过程而言）。管理者这个角色可以简单地由互动过程中的其中一个参与方担任，或者可以是一个中心化的机构；在受监管的金融产业场景里，最理想的情况是由中心化机构担任管理者这个角色，毕竟这种场景里的开发者本来就需要了解系统中发生的各种情况。[29]

另一个独立的问题是：如果 KYC（了解你的顾客）信息必须针对每一个账户进行独立的保存，谁来保存它呢？若要保证隐私性的话，单靠私有链或许无法实现这个目标，这其中的原因已经在这章的开头讲述过了。不过，另一种密码学技术可以为你带来更多的进展，这就是秘钥共享技术。总体上说，秘钥共享的概念跟访问数据所用的多重签名技术有类似之处。一份数据可以在 N 个参与方之间进行共享，而其中的 M 个参与方可以共同恢复该数据，不过若 M-1 个参与方就无法看到任何有用的信息，只能看到加密后的随机信息（M 和 N 的值可以进行任意设置，如 2/3，6/12，9/9 等）。理论上说，开发者可以将这项技术与 ZK-SNARKs 技术结合起来，实现类似 David Chaum 的加密信息提案的金融隐私信息保护方案：

总体上说要求全面保护隐私信息及实现匿名性，不过要求他们将于交易相关的细节信息（发送者和接受者，数值等）在 N 个主钥匙持有者之间进行秘密共享。ZK-SNARKs 可用于确保数据是有效的，若不包含这样的有效元数据或证据的交易就无法被接纳到区块链中。

ZK-SNARKs 是一项很强大的技术，不过可能会在安全性上有一些未经证明的问题，可能还有效率方面的担忧：当前，在一台普通的计算机上创建一个这样的密码学证明大约需要超过 90 秒的时间。因此，对很多应用程序来说，一些隐私保护性能稍弱但更高效的密码学技术或许会更好用，如环签名和加法同态加密技术。

以太坊团队在整合这些方案的过程中有两个角色。首先，这些技术已经有不少可以建造在以太坊上了；类似 Raiden 这样的项目在以太坊上搭建 State Channels 网络的尝试已有数月的时间，而可链接的环签名方案也已经被搭建（以一个早期的测试版本的形式）。我们有意与密码学专家和其他开发者一起，将上述的一些方案整合到以太坊里面，尽量达到最大程度的可用性。其次，虚拟机的效率已经成为了整合这些技术的主要瓶颈，这会成为本项目未来发展的一个重要焦点。因此，在以太坊的公有链上若要最大程度地将这些技术整合进去并进行实践，其时间表与 Serenty 版本的释出以及（可能发生的）WebAssembly 技术推进会有紧密的关联。当然了，私有链可以通过提前整合这些特性的方式获得这些得益，也可以通过简单地添加一些预先编译合约的形式去实现。

关键的建议

在可扩展性的问题上，区块链有不少的解决方案，但隐私性的问题就不是这样了。我们已经作出私有链无法彻底解决这类问题的结论了；因为私有链虽然在可校验性上具有 33%或 50%的容错程度，但在关于隐私性的问题上，它可能不如一台中心化的服务器；在中心化的方案上，你或许只能入侵一台服务器；但在区块链上，节点非常多，若任何一台节点的信息被泄露，都会揭示出所有的信息。

看来，私有链并不是一个解决隐私保护问题的合理方案，在此之外，有一些针对特定应用场景的技术，让你可以为特定的用例创建高度的隐私性，这些用例

具体会包括数字资产转移，身份校验，多方智能合约以及数据存储。这些解决方案需要使用高级的密码学技术，如环签名，加法同态加密，零知识证明以及秘钥共享技术。隐私保护方案可以进行高度的定制，如只有在特定的情况下才能向特定的参与方展示特定的信息（交易相关的参与方，金融机构，监管者等）。这样的系统若能得到深入的应用，就有可能在高度地保护用户隐私权的同时满足监管者及其他信息索取方的需求。

以太坊项目的路线图有计划整合这些技术，并让开发者们在以太坊上建立带有隐私保护特性的应用程序变得更简单。在 State Channels 的例子中，已经有不少项目在整合这项技术了。不过，私有链上的以太坊用户可以通过自己的实践，根据自己的进度选择整合上述列表中的任意技术，并以预先编译的合约（Precompiled Contracts）形式去实施。

纯净性

或许在讨论这些技术能如何用于以太坊时，最关键的一点是：以太坊的主要理念是创建一个具有高度通用性的、高度“开发者友好”特性的平台。与 Monero 这种平台不一样的是，Monero 尝试将“可链接的环签名”作为底层代币管理方案的必须部分，而以太坊将自身看做是一个计算平台，对在其之上搭建的应用并没有特定的要求。因此，以太坊并没有计划实现一个“匿名的代币”，也没有计划走向另一个极端——即不会在协议的层面整合一个现实世界的身份管理系统，这一点对公有链、私有链版本的协议来说都是一样的。

以太坊的主导理念是将技术建立在不同的层面。“第一层”是以太坊的平台自身，包括以太坊的虚拟机（Ethereum Virtual Machine），执行环境以及区块链共识算法。在这之上，开发者们可以将这些隐私保护方案搭建作为“第二层”。在现在，以太坊的协议或许并不是完全“纯净”的：在协议的层面之上，包含了具有余额和序列号的账户的概念，这个部分被 secp256k1 数字签名算法保护着，作为交易执行的“入口”。不过，我们计划通过 Serenity EIP 101 改善方案计划去改变这个现状，将这些特性剥离出来：所有交易都会被校验——只要它们的格式是正确的，而所有的交易看起来都会来自“入口”账户 0xffffffff.....

用户的账户将会成为一种“传递合约”——即以太坊上的一个智能合约，可以从入口接收到信息，并仅在该信息包含合法的密码学签名以及序列号时将其传递出去。

这个方案的所带来的影响，是以太坊当前带有序列号的账户模式并不会再拥有系统的特权了；相反的是，任何涉及到 UTXOs、可链接的环签名（linkable ring signatures）、机密交易（confidential transactions）或者其他技术的账户管理方案可以被整合到这个平台上，感觉上就像是“原生”的账户管理方案一样——与现时所用的账号和序列号构成的账户管理方案并没有太大的区别。虽然 EVM（以太坊虚拟机）还没有升级到 WebAssembly 技术（或其他类似的替代技术，如果我们最终要采用那些方案的话），但里面还是有加入一些特定的密码学操作指令的空间，这样就可以避免虚拟机的整体资源耗费，并以合理的效率去处理这些指令，但这些操作指令还是被高度封装过的，我们的最终目标是让所有的功能都在 WebAssembly 的代码里面执行。到那个时候，以太坊的私有链或公有链的实践案例或许会带有一套复杂的第二层工具，让用户可以创建由密码学保护的账号，管理他们的身份，选择性的向特定的参与方展示或证明自己身份相关的信息，或执行其他类似的操作。这样，平台的底层结构就会得到极大地简化，效率也会上一个台阶，而且开发者们依然会拥有高度的自主权，这样他们就能在平台的底层之上搭建满足不同产业和应用要求的工具。

结论

以太坊提供了一个具备高度通用性的平台，让用户可以为各种类型的用例创建应用程序，从而省去了自己搭建区块链所用的时间和耗费。这个平台的愿景是要搭建一个“世界计算机”：它要创建一个系统，对用户来说，这个系统看起来就像是一台计算机一样，同时拥有了区块链技术所带来的安全性、可审计性和去中心化的好处。这个项目的路线图还包括了一些将来的计划，旨在解决一些区块链技术存在的问题，如可扩展性、隐私保护等，从而为多种应用场景提供极致的友好体验，并最大程度上保留和完善当前的“对开发者友好的用户体验”。

在一些场景里面，以太坊或许是不适用的，这些场景包括：（i）在一些能通过搭建专用平台而受益的场景中，在这些场景里，搭建专用的平台能够针对某些特殊的目标进行优化，其提升的可扩展性及效率明显超过了进行专门定制化操作所需的额外成本，或（ii）区块链完全不适用的场所。在以下场景中，以太坊是有意义的：（i）在需要一个具有长期、高度可审计性需求的平台中，以及在需要快速加入新功能的平台中——甚至在没有区块链节点运作者的协作时也能增加新的功能，（ii）在需要创建一个大型的应用程序协作生态时，或（iii）在用户需要使用自动化执行、可编程的“智能合约”的场景中。

至于在以太坊的公有链、私有链还是联盟链的选择中，最终是取决于开发者和应用场景的需求。目前，那些没有得到机构支持的开发者大部分选择了公有链，因为它的准入门槛很低，而且不需要请求别人去采用他们的应用程序。那些早就有几百万用户的金融机构并没有这方面的忧虑，因此这些机构应该将这些选项都考虑进来。在短期内的“安全”的策略就是先尝试一下具备 5-25 个节点（每个机构运营一个或多个）的联盟链，这是因为——(i)联盟链在短期内具有可扩展性的优势，(ii)一个“受控”的系统所带来的低风险性有利于说服法律部门和监管者，而且（iii）他们也能在公有链前使用上如 EIP101 和 105（这两个都是以太坊改善提议的编号）的特性；在长期来说，私有链、联盟链和公有链的选择会取决于特定的应用场景，特别是要考虑在性能和互操作性能之间的取舍。

对那些重视隐私性的用户来说，私有链并不是一种灵丹妙药；在协议的层面上，任何多节点的区块链的隐私性都可能不如一个单服务器的解决方案。因此，我们应该进一步探索 State Channels 状态通道，ZK-SNARKs 零知识证明，环签名等密码学方案，因为它们能够使用区块链去存储金融数据并进行相关的计算，同时保留为用户提供高度的（但并不是绝对性的）隐私保护。以太坊的路线图已经有相关的计划，力求为这类隐私保护技术提供最友好的支持，将其整合到以太坊的协议之上。

附录：引用资料

1 As explained in a later section, Bitcoin actually uses a concept of UTXOs and not balances; however, the description of the protocol using balances is simpler and suffices to illustrate the concept.

2 More precisely, miners are supposed to reject blocks that contain invalid transactions. In practice, miners may earn higher profits by not bothering to validate blocks, instead free-riding off of the presumed validation of others. If enough miners do this, such behavior can lead to dangerously long forks of invalid blocks, as took place in June 2015. The "Verifier's Dilemma" paper from the National University of Singapore discusses the economic incentives behind non-validating miners in an Ethereum context in much more detail; the general conclusion is that if block verification is sufficiently computationally intensive then the system could flip to the non-validating equilibrium, and this gives yet another reason why public blockchains have constraints on their throughput that place their maximum safe capacity far below the theoretical processing power of a single CPU.

3 The question of who actually legally owns assets while they are under the control of a smart contract is complex; common law naturally did not evolve with the understanding that the underlying owners of assets could be computer programs executed on a consensus ledger. See Robert Sams's footnote 32 from Consensus-as-a-service for more details.

4 For a more in-depth explanation of private and public blockchains, see https://blog.ethereum.org/2015/08/07/on-public-andprivate-blockchains/

5 See miiCard and Tradle for examples of companies that are trying to create blockchain-based KYC platforms. Note that a current major problem with such schemes is that companies are often required to all conduct KYC checks themselves, and cannot simply piggyback off of external services; hence, achieving the full gains of blockchain-based KYC will require regulatory support.

6 For an alternate view specific to a private blockchain context, see Gideon Greenspan's blog posts and IBTimes article.

7 Note that here lies a major philosophical difference between Ethereum and many other protocols, where there is no standard independently defined protocol specification, and in fact a policy that the protocol is the implementation is often explicitly adopted. If there is a bug in the implementation then that bug will often simply become part of the protocol. Perhaps the best known example is OP_CHECKMULTISIG, which was accidentally implemented in a way that consumes the top value on the stack with no effect before doing anything else; this is now solidified as a protocol rule.

8 A few security bugs have been found in the months immediately after launch, see these three blog posts alerting users to update; however, the frequency has been rapidly decreasing, and of course Bitcoin itself has not been without problems.

9 Outside of the Least Authority analysis, the "Demystifying Incentives in the Consensus Computer" paper from the National University of Singapore provides additional insights regarding the limitations of incentive-compatibility in Ethereum

10 A third mitigation route is to parallelize at least the hashing portion of the work in Merkle tree updates (theoretically not too difficult to do particularly if tree split/merge operations are implemented), and a fourth is to look for processors with hardwareaccelerated Keccak implementations (or switch to a hash function that does have hardware acceleration in the processors that you intend to use)

11 There are also other forms of quasi-centralization; for example, most miners now pass blocks to each other using the Bitcoin Relay Network instead of the P2P network used by regular nodes. Increasing block sizes also increases incentives for these kinds of behaviors to emerge. 12 There do exist various proposals to increase the de-facto Bitcoin block size, including the Bitcoin Classic 2MB hard fork, and the Segregated Witness proposal, which essentially (alongside some transaction malleability fixes) includes an accounting change such that bytes from transaction signatures are now only partially counted toward the block size, thereby increasing the de-facto block size to 1.5-3 MB depending on the types of transactions that are sent. However, these only increase the maximum throughput by a factor of two, several orders of magnitude below what is required to achieve the throughput necessary for mainstream applications.

12 There do exist various proposals to increase the de-facto Bitcoin block size, including the Bitcoin Classic 2MB hard fork, and the Segregated Witness proposal, which essentially (alongside some transaction malleability fixes) includes an accounting change such that bytes from transaction signatures are now only partially counted toward the block size, thereby increasing the de-facto block size to 1.5-3 MB depending on the types of transactions that are sent. However, these only increase the maximum throughput by a factor of two, several orders of magnitude below what is required to achieve the throughput necessary for mainstream applications.

13 See Vlad Zamfir's DEVCon 1 presentation and other writings for a more detailed description of what publicblockchain "cryptoeconomics" entails.

14 The research on how this can actually be accomplished has been carried out by several independent groups that have converged on roughly similar solutions; see Chain Fibers Redux and Dominic Williams' Dfinity project; the work by Maidsafe with its concept of "close groups" is similar. No system has been implemented in practice yet, and so of course many technical challenges in implementation remain.

15 In general, "the multisig-oracle model of smart contract execution" consists of a design where N parties put their assets into a contract where a majority of a given set of M "notaries" have full control over the assets; at that point, the M notaries execute the code, maintain any internal state and move the funds as required. See Codius and CIYAM Automated Transactions for examples of attempts to implement this in practice.

16 Note that the challenge of coming up with worst-case examples for contract executions that are unparallelizable is exactly equivalent to the problem of creating sequential memory-hard hash functions; perhaps somewhat ironically, the cache generation process in Ethereum's mining algorithm Ethash is one example

17 In general, cryptographers view the concept of "identity" as encompassing any solution to problems of the form "prove that action A and action B were made by the same entity". "Anonymous" comes from the Greek "no name"; essentially, no actions can be correlated with each other. "Pseudonymous" ("false name") means that there is an identifier, but it is not connected to the identifier that that entity uses in other contexts (eg. a legal name). Mining by default is anonymous since blocks do not come with any information; the Puzzle Towers scheme requires miners to specify an address, and in order to earn substantial profits miners need to mine multiple blocks with the same address, hence the address is an identifier, albeit a "pseudonymous" one.

18 This generally includes algorithms such as PBFT; a more detailed overview can be found on the wikipedia page on Byzantine fault tolerance.

19 Proof of stake is a broad area of research; it's also worth noting that there are generally two categories of proof of stake: "first generation" proof of stake algorithms, which try to maximally mimic proof of work, and "second generation" proof of stake algorithms which try to add more rigorous cryptoeconomic incentivization (and disincentivization) measures to either first-generation-style algorithms, traditional Byzantine fault tolerant consensus algorithms, or combinations or variations of the two categories. First-generation algorithms have stake grinding and nothing-at-stake vulnerabilities; for this reason they are generally viewed with suspicion by the academic community, even though newer iterations have removed stake grinding concerns an argument can be made that stake grinding has not been an issue to first-generation PoS blockchains in practice. Second-generation algorithms, including Tendermint and Ethereum's Casper, are robust against these concerns, but have not yet been tested "in the wild".

20 okTurtles' DNSChain uses blockchains in part to achieve easy certificate revocation; see okTurtles' FAQ for a more detailed description. Also, a more specific certificate revocation system using the bitcoin blockchain has been built by Christopher Allen. However, neither system has yet seen substantial usage.

21 See the Lightning Network for an implementation on Bitcoin, and Raiden for an implementation on Ethereum.

22 Another idea that can be philosophically viewed as being in some sense similar to state channels is the notion of "fidelitybonded banking", first conceived by Peter Todd in 2013. Fidelity-bonded banking, as a general category, is a combination of two ideas. First, we have known for a long time that it is possible to design financial services in such a way that every operation carried out by the operator is cryptographically provable, allowing users to audit the service and make sure that it is acting honestly; doing so was a key design goal of the

OpenTransactions project, as well as Greg Maxwell's proof-of-solvency scheme. Second, once these proofs exist, we can create an Ethereum smart contract such that if a valid "proof of fraud" is submitted, the contract takes money out of the bank's deposit and sends the user the amount that they are entitled to. Fidelity-bonded banking hence relies on similar notions to state channels, and it may have applications in democratizing the provision of financial services by allowing service providers to build services that are mathematically provably trust-free, thereby removing the need for the providers to be trustworthy themselves.

23 See https://crypto.stanford.edu/seclab/sem-14-15/pustogarov.html and http://www.forbes.com/sites/andygreenberg/2013/09/05/follow-the-bitcoins-how-we-got-busted-buying-drugs-on-silk-roads-black-market/#64dea53189a8 for examples of bitcoin account de-anonymization using computer-science-theoretic techniques being employed in practice. Sometimes, such techniques are not even required; see the fate of Carl Mark Force after attempting to steal hundreds of thousands of dollars in bitcoin from Silk Road: http://motherboard.vice.com/read/how-a-twotiming-dea-agent-got-busted-for-making-money-off-the-silk-road

24 Monero is an example of a public-blockchain cryptocurrency that provides such functionality built in.

25 The question of how to actually put cryptographic digital identity schemes into people's hands is a complicated one, but governments are moving in that direction; perhaps the most successful example so far is Estonia's e-Residency program, which allows anyone in the world to obtain a smart card that can be used to cryptographically sign documents in a way that can be verified

26 And of course, note that the math that ensures balances are non-negative in CT schemes can also be flipped around and repurposed to enforce maximum balances; this technique could also be used to enforce account balance or transfer limits while preserving privacy

27 If it is desired to restrict the kinds of actions that users can take, then one must think carefully about the set of smart contracts that users are allowed to create and participate in; sending assets into a smart contract that says "anyone who can provide a signature matching public key X can withdraw the assets" is essentially an asset transfer to an arbitrary recipient. Another case worth keeping in mind is that a smart contract that says "anyone who can provide a signature matching public key X can withdraw the assets or change the value of X" - essentially, a tradeable contract that holds some assets - could be used to circumvent non-transferability requirements. Whitelisting the set of smart contract templates that users are allowed to enter into at least initially seems like the safest choice.

28 As usual, this requires either a government authority or an acceptable substitute entity to be willing to create cryptographically signed digital documents that can be interpreted by the zero-knowledge proof scheme 29 An alternative to these schemes is Enigma, which uses secure multiparty computation to achieve "private contracts" with an M-of-N trust model. Secure multiparty computation makes serious efficiency sacrifices (eg. most protocols require a set of

network messages to be exchanged for each multiplication, although if the “circuit” can be processed in parallel then the effect of this in practice is mitigated), but the resulting trust model is quite strong particularly in a private blockchain context. In a public blockchain context, there is no way to prove that the consensus participants did not collude to leak information, hence honesty cannot be incentivized, whereas in a private blockchain context if such collusion is discovered the response of “just suing the bad actors” is always an option.