**第二节 区块链基础技术的介绍**

**一、分布式账本**

随着最前沿的科技成果应用，分布式账本技术崭露头角，并在许多领域中得到推广。从最简单账本到复式账本，再到数字化账本，以及目前正在探索的分布式账本[3]，账本科技的每次突破都会引起不同领域里程碑式的发展，同时也对我们生活的各个方面不断进行改变。举个简单例子：数学常识“1+1=2”，将“1+1=2”比喻成“记账内容”，每个人都认可“1+1=2”这个数学共识，那所有人类中的每个人就是“分布式账本”。在已知的数学逻辑下，如果有人想要指鹿为马说“1+1=3”，那就需要更改全人类的记忆才行，难度相当大。

（一）基本介绍

分布式账本（Distributed ledger）的数据库分布于对等网络的节点(设备)上，当中每个节点都复制及存储与账本完全相同的副本并独立更新。分布式账本的主要优点是不存在中央权威[4]。通常情况下，当一个账本出现更新，每个节点都将执行一笔新交易，然后所有节点以共识机制投票决定哪一个副本是正确的。一旦达成共识，所有其他节点都会按照正确副本的数据进行更新。在区块链系统中，将数据区块按照时间顺序相连组成逻辑上的链，有着持续增长并且排列整齐的记录[5-6]。每个区块都包含一个时间戳和一个与前一区块的链接，因此可以将区块链看成一个不断增长的账本。账本可以完全公开，例如比特币系统和以太坊系统，也可以在联盟内公开，例如Hyperledger Fabric, corda, FISCO BCOS等。

（二） 基本特点与分类

1.去中心化

去中心化意味着不依赖于中央处理节点，没有中心化的应用和管理部分。数据库中的数据可以通过多个站点、不同地理位置或者多个机构组成的网络进行分享[7]。在一个网络里的网络成员都可以获得一个唯一、真实的账本副本。账本里的任何改动都会在在所有的副本中被反映出来，反应时间会在几分钟甚至是几秒内。在这个账本里存储的资产可以是金融、法律定义上的、实体的或是电子的资产。

2.共识机制

根据网络中达成共识的规则，账本中的记录可以由一个、一些或者是所有参与者共同进行维护[4]。网络中的参与者根据共识原则来制约和协商对账本中的记录的维护。没有中心化的第三方仲裁机构的参与。

3.信息不可更改

分布式账本中的每条记录都有唯一的时间戳和唯一的数字签名，这使得账本成为网络中所有交易的可审查记录。如果有任何的人员想要修改数据，一般需要根据共识机制与其余人员达成一致才能够完成，不然是无法进行修改的。

随着科技的不断进步与发展，分布式账本的需求日益提高。自从电子计算机问世以来，数字化账本就因其高效便捷的特点成为了主要应用的记账方式。数字化账本不但可以提高大规模记账的效率，而且可以避免人工书写的错误，使得账本的规模、记账处理的速度、账本的复杂度，都有了天翻地覆的提升。数字化账本虽然不容易出错，但其仍是中心化的形式。这就意味着账本掌握在个体手中，一旦出现数据丢失则无法找回，同时涉及到多个交易方的情况下，需要分别维护各自的账本，容易出现不一致、对账困难的情况。由此可以很自然的想到，可以借助分布式系统的思想来实现分布式账本：由交易多方一起来共同维护同一个分布式账本，打通交易在各个阶段的来龙去脉，凭借分布式技术，进一步提高记账的安全和可靠性[8]。

根据区块链系统的构建目标，账本会呈现出不同的形态，从账本的所有权（包括查看权与写入权）来看，在比特币等公有链系统中，所有用户都对账本具有查看权，在符合共识算法的前提下，特定的节点对其具有写入权。而在Fabric等系统中，符合共识机制的条件下，仅有某些通道中的特定节点对该通道的账本具有所有权。

根据分布式账本的定义，可以简单的设计出一个分布式账本[9]，如图2-4所示，其中A、B、C、D、E…代表参与方，其对应的账本分别为a、b、c、d、e…。从图中可以看出，所有的参与方都可以对其进行更改与维护。如果所有的参与方均可以按照其共同约定进行账本信息的更改与上传，则该帐本具有可信性，各参与方也可以正常进行工作，但如果有参与方违反约定协议，进行恶意操作，随意更改数据，账本将不具有可信性。

A

参与方

账本

B

C

D

E

...

a

b

...

e

d

c

**图2-4 简单分布式账本示意图A、B、C、D、E…代表参与方，a、b、c、d、e…代表账本**

为了防止上述情况发生，需要将图2-4简单分布式账本进行更改，加入验证机制，对发生过的交易进行校验，引入数字摘要，形成一种不可随意篡改的分布式账本，如图2-5所示。当新的交易信息被添加到账本上时，参与者可根据历史帐本信息对新加入的信息进行验证，一旦新写入的信息不符合验证，各方参与者便可以发现，同时可以确定信息位置。虽然此分布式账本解决了交易信息被随意篡改的问题，但是不可扩展的缺陷仍是一个不可避免地严重问题。由于每次验证都需要对所有的信息进行计算，随着帐本中的信息数量的增加，进行验证的成本将不断地增加，因此，这种账本对于数据量大的账本并不适用。

A

参与方

账本

数字摘要

B

C

D

E

...

a

b

...

e

d

c

**图2-5 防篡改分布式账本示意图**

**A、B、C、D、E…代表参与方，a、b、c、d、e…代表账本，代表数字摘要验证区间。**

为了解决大数据量账本的问题，对图2-5的账本进行进一步改进，得到新的账本模型，如图2-6所示。每次进行验证数据的准确性时，保证从头开始到验证开始位置数据的准确性。因此，每次加入新的交易信息时，只需要对部分历史交易信息进行验证即可，这样既解决了信息篡改的问题，同时也解决了数据的扩展问题，能够对大数据量的账本进行操作。这种分布式账本结构即为区块链结构。

A

参与方

账本

数字摘要

B

C

D

E

...

a

b

...

e

d

c

**图2-6区块链分布式账本示意图**

**A、B、C、D、E…代表参与方，a、b、c、d、e…代表账本,代表数字摘要验证区间。**

目前常见的三种分布式账本技术包括Hyperledger Fabric，R3 Corda 和以太坊。国内具代表性的开源社区是由中国区块链技术和产业发展论坛发起建立的分布式应用账本（DAppLedger）开源社区。BCOS是其中重点孵化的开源项目之一。接下来将对以上四种典型开源底层平台进行简要介绍与对比。以太坊（Ethereum）：开源的有智能合约（将在2.5节进行详细介绍）功能的公共区块链平台，数字货币[[1]](#footnote-1)是以太币（Ether），提供分散化的的Ethereum Virtual Machine（EVM）处理点对点的合约，由Vitalik Buterin提出，使用Solidity语言开发智能合约，共识机制采用工作量证明（账本级），交易执行需要消耗Gas[[2]](#footnote-2)，gas用完后会回滚操作，依据Gas价格决定打包优先级，价格越高越早被打包进区块中。

Hyperledger Fabric：由 Linux 基金会发起创建的开源区块链分布式账本，可用于全球供应链管理、金融交易、资产账和去中心化的社交网络等场景，但无意以此来构建一种加密货币，每个交易都会产生一组资产键值对，可重用公司自带的身份管理功能，HSM（硬件安全模块）支持对保护和管理数字密钥（将在2.3节进行详细介绍），数据格式是json，数据库CouchDB支持富格式[[3]](#footnote-3)和富数据[[4]](#footnote-4)查询。

Corda：由分布式账本创业公司R3CEV开发，应用于商用DLT平台，使用Kotlin、Java语言，能够进行并行交易。Corda 舍弃了每一个节点都要验证和记录每一笔交易的账本全网广播模式，仅仅要求每一笔交易的参与方对交易进行验证和记录。这样做的好处主要是解决了分布式账本技术在商业化应用中非常敏感的两个问题：极大地提高了交易的吞吐能力；避开了共享账本能否保证交易数据私密的争议。同时也带来了问题，即如何避免“双花”。在比特币和以太坊等区块链平台上，由于每个节点都拥有整个账本的复制，所以要解决双花问题很容易。Corda 为解决双花问题，引入了 notary 机制，简单来说就是在 notary 节点之间形成更广泛的共识，而 Corda 上的每一笔交易都需要通过至少一个 notary 节点的验证。

BCOS：由微众银行、万向区块链、矩阵元联合开发建设，金链盟开源工作组在此基础上，聚焦金融行业需求，进一步深度定制发展为FISCO BCOS。BCOS和FISCO BCOS皆已开源并互通有无，截止2018年中期，BCOS/FISCO BCOS开源社区实名用户已有1100余人，有70家企业在预研或开发阶段，10家已经实现应用上线。下表2-1对典型开源底层平台进行了对比。

**表 2-1 典型开源底层平台对比**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 以太坊 | Hyperledger  Fabric | R3 Corda | FISCO BCOS |
| 平台类型 | 公有链 | 联盟链 | 商用DLT平台 | 联盟链 |
| 治理 | 基金会 | 基金会 | R3CEV公司 | 微众银行、万向区块链，矩阵元，金链盟开源工作组等 |
| 权限管理 | 非授权 | 授权 | 授权 | 授权 |
| 共识算法 | 工作量证明（账本级） | 0.6版本支持实用拜占庭容错算法（交易级），1.0版本后支持Solo（单节点共识）、kafka（分布式队列）和SBFT（简单拜占庭容错） | 公证人（交易级） | PBFT/RAFT |
| 智能合约开发语言 | Solidity | Go、Java | Kotlin、Java | Solidity |
| 可扩展性 | 正在开发Sharding分片模型 | 支持通道设计，区分不同的业务 | 可并行交易，采用多公证人提升性能 | 多群组架构，多链平行扩展，支持跨链调用 |
| 隐私保护 | 暂无 | 用通道隔离不同的业务，1.0版本后引入了私有状态和零知识证明 | 采用Merkle树结构隐藏交易细节 | 数据脱敏，分级隔离，并实现了零知识证明，群签名，环签名，同态加密等 |

在过去几年中，开源社区发展迅速，参与人数不断增加，同时产品的特性也不断发展。截至2018年8月，Hyperledger开源社区成员由初创时的30多名成员，增长到超过250名成员。fabric1.0版本公开的数据表示，共27个组织，159名开发者参与了代码贡献。

BCOS平台于2017年开源，初始成员为微众银行、万向区块链，矩阵元。其金融分支FISCO BCOS依托金融区块链合作联盟（简称金链盟）开源工作组共9家机构共同建设，金链盟目前的成员超过了百家机构。

Corda所在的R3联盟初创成员为42家，目前超过200家成员。

以太坊社区由全球开发者合作贡献代码，据资料介绍核心开发组织包含400多名开发者，密码学者等。随着企业级市场对区块链的诉求，于2017年应运而生了以太坊企业联盟(EEA)，初创成员为30家，目前已经超过500家机构加入。

在数量的大幅增长之外，参与者的角色也在丰富，除了开发者之外，社区里出现了基于平台产品进行各种商业应用场景落地的参与者，这些参与者包括投资人，集成商，应用开发者，第三方安全审计公司等，包括DAPP在内的应用生态逐步繁荣。

从各开源软件平台的版本发布特性分析，各开发社区重点投入的产品方向包括易用性，隐私保护，可扩展性，安全防护，以及进行整体性的架构优化等。

在易用性方面，随着开发者和和社区用户的增加，对开源软件的部署，配置，应用开发，运营运维等方面都提出了更多的要求，各平台分别在开发工具，部署工具，数据查询和统计分析，系统运维工具等做了大量的建设，以降低使用者的门槛，加速开发效率。

在隐私保护方面，因为商业场景对商业数据、机构和人员等信息的隐私保护有很高的要求，各平台会通过架构优化或密码学等方式实现不同力度的隐私保护，如fabric在1.0版本加入了私有数据特性，在1.3版本实现了使用零知识证明来保持客户身份匿名和不可追踪等。BCOS/FISCO BCOS提供了可监管的零知识证明，环签名，群签名，同态加密等算法实现以保护用户隐私。隐私保护的理论复杂性和工程难度都比较高，目前属于起步和探索阶段，还会持续演进，以追求在保护的全面性，性能效率方面的突破。

在可扩展性方面，各平台分别根据自己的架构，提出不同的可扩展性方案，如fabric基于通道的设计，允许机构根据业务类型接入不同的节点，使不同的业务分布在不同的通道上。BCOS/FISCO BCOS采用平行多链架构支持更多的业务量并发，实现了同构链之间的跨链通信。以太坊目前正在开发类dpos的公式算法，以及推动侧链，sharding分片等可扩展方案，预期实现还需要较长时间。随着应用场景数量，链的使用者，使用频度的增加，各平台需要在可扩展性方面持续演进，包括跨链、侧链、分片等技术都会在各社区里逐渐引入和实现，以应对更大的规模的网络，以及满足更丰富的互联互通场景。

在安全防护方面，许可链社区通常比较认可基于PKI体系的身份认证、权限控制等措施，持续丰富和细化证书的运用，以及在身份，网络，数据，交易规则等不同层面加入严密的保护。公有链的安全问题主要体现在网络攻击，智能合约漏洞，恶意分叉等，也促使以太坊社区持续对合约引擎，代码漏洞进行多方查验和修复，以及通过社区治理的方式，决定如何应对安全漏洞等问题导致的资产损失。

在整体的架构上，各社区均推崇插件化的可扩展的设计，通过插件化体系，使得平台产品可灵活的支持不同的共识算法，密码算法，存储引擎，兼容多个版本的网络协议等，使得产品的演进具备更高的速度和更好的灵活性。如Fabric支持Solo（单节点共识）、kafka（分布式队列）和SBFT（简单拜占庭容错），BCOS/FISCO BCOS支持PBFT和Raft共识算法，Fabric状态数据库可采用Level DB或者Couch DB，或其他的key-value数据库,bcos/fisco bcos可支持LevelDB，以及分布式的关系型数据库，此外，在我国的商业场景中需要实现的国密算法，也可以通过插件化进行支持[10]。

（三） 分布式账本的未来

一般认为，分布式账本的技术发展路线可以分为四个阶段：市场需求分析和平台技术体系研究阶段、分布式账本关键技术方案选型和平台具体建设阶段、分布式账本技术开源与平台运行优化阶段、技术实践应用试点与技术推广阶段。中国的分布式账本技术研究应当遵循国际研发规律，加速分布式账本技术的落地[11]。

分布式账本技术可以有效的改善当前基础设施中出现的效率极低成本高昂的问题，而导致当前市场基础设施成本高的原因可以分为三个：交易费用，维护资本的费用和投保风险费用。在某些情况下，特别是在有高水平的监管和成熟市场基础设施的地方，分布式账本技术更有可能会形成一个新的架构，而不是完全代替当前的机构。

分布式账本技术有潜力帮助政府征税、发放福利、发行护照、登记土地所有权、保证货物供应链的运行，并从整体上确保政府记录和服务的正确性。在英国国民健康保险制度（NHS）里，这项技术通过改善和验证服务的送达以及根据精确的规则去安全地分享记录，有潜力改善医疗保健系统。对这些服务的消费者来说，这项技术根据不同的情况，有潜力让消费者们去控制个人记录的访问权并知悉其他机构对其记录的访问情况。分布式账本已经在金融领域取得众多成果，将使金融行业产生新的活力与机遇。

**二、加密技术**

若某一公司开发了一款软件，该公司可以在该软件流入市场时设定软件只能单机单用户使用，并设定诸多的使用权限，而只有当用户条件全部符合使用权限时，该软件的用户才能使用该软件产品，换言之即使用户下载并安装了该软件，但若他没有被分配授权或授权无效，那他也是不能使用软件产品的功能的，这意味着如果该公司的授权被其他第三方破解后，软件可以无需经过该公司允许被随意的复制和使用，甚至可能被不法第三方冠名以另一产品的名称，并以远远低于原产品市场售价的形式流通于市面，对于投资巨额开发该产品的软件开发公司造成了不可估量的经济和利益损失。至此授权加密的重要性可见一斑。

（一）基本介绍

为了保证账本的完整性、公开性、隐私保护、不可篡改、可校验等一系列特性，区块链技术高度依赖加密技术。

加密技术是电子商务采取的主要安全保密措施，是最常用的安全保密手段，利用技术手段把重要的数据变为乱码（加密）传送，到达目的地后再用相同或不同的手段还原（解密）。加密技术的应用是多方面的，在电子商务、VPN、通讯和存储领域都有广泛的应用，深受广大用户的喜爱。

（二）基本特点与分类

加密技术包括两个元素：算法和密钥。算法是将普通的文本（或者可以理解的信息）与一串数字（密钥）的结合，产生不可理解的密文的步骤。密钥是一种参数，它是在明文转换为密文或将密文转换为明文的算法中输入的参数[12]。在安全保密中，可通过适当的密钥加密技术和管理机制来保证网络的信息通讯安全。密钥加密技术的密码体制分为对称密钥体制和非对称密钥体制两种。相应地，对数据加密的技术分为两类，即对称加密和非对称加密。对称加密以数据加密标准（DES，Data Encryption Standard）[[5]](#footnote-5)算法为典型代表，非对称加密通常以RSA（Rivest Shamir Adleman）[[6]](#footnote-6)算法为代表。对称加密的加密密钥和解密密钥相同，而非对称加密的加密密钥和解密密钥不同。

对称加密采用了对称密码编码技术，它的特点是文件加密和解密使用相同的密钥，即加密密钥也可以用作解密密钥，这种方法在密码学中叫做对称加密算法，对称加密算法使用起来简单快捷，密钥较短，且破译困难，除了数据加密标准（DES），另一个对称密钥加密系统是国际数据加密算法（IDEA），它比DES的加密性好，而且对计算机功能要求也没有那么高。IDEA加密标准由PGP（Pretty Good Privacy）系统使用。

1976年，美国学者Dime和Henman为解决信息公开传送和密钥管理问题，提出一种新的密钥交换协议，允许在不安全的媒体上的通讯双方交换信息，安全地达成一致的密钥，这就是“公开密钥系统”。相对于“对称加密算法”这种方法也叫做“非对称加密算法”。与对称加密算法不同，非对称加密算法需要两个密钥：公开密钥（publickey）和私有密钥（privatekey）。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫做非对称加密算法。在现实世界上可作比拟的例子是，一个传统保管箱，开门和关门都是使用同一条钥匙，这是对称加密；而一个公开的邮箱，投递口是任何人都可以寄信进去的，这可视为公钥；而只有信箱主人拥有钥匙可以打开信箱，这就视为私钥。

非对称加密技术工作流程如图2-7所示。A想要给B发信息，首先对信息用公钥进行加密处理，形成密文，进行传输，接收者通过私钥对密文进行解密，得到明文信息输出。加密技术在生活中有广泛的应用，例如在网络银行或购物网站上，因为客户需要输入敏感消息，浏览器连接时使用网站服务器提供的公钥加密并上传数据，可保证只有信任的网站服务器才能解密得知消息，不必担心敏感个人信息因为在网络上传送而被窃取。

明文输入

加密算法

解密算法

传输的密文

明文输出

B的私钥

B的公钥

**图2-7非对称加密技术**

在非对称加密中，为了保护隐私，通过签名和验签完成权属证明问题。在加解密过程中发送者用私钥加密，即签名，接收者用公钥解密，即验签。

（三） 哈希函数

1.基本介绍

区块链账本数据主要通过父区块哈希值组成链式结构保证其不可篡改性。哈希（Hash）即一般翻译为散列、杂凑，或音译为哈希，是把任意长度的输入（又叫做预映射pre-image）通过散列算法变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来确定唯一的输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。以比特币为例，使用的是SHA256算法（是安全散列算法2（SHA-2）下细分的一种算法），其哈希值长度是256个二进制的字符串，以十六进制数字表示时字符串的长度为64位，比如“区块链”的SHA256信息摘要为：6E3110B33188C7A3056CB91E4C35EFE609E8E565DD560300502403EBDE626196

公式表示形式：

h=H(m) （2.1）

式中m：任意长度消息（不同算法实现，长度限制不同，有的哈希函数（SHA-3）不限制消息长度，有的限制（SHA-2），但即使有限制其长度也非常大，可以认为是任意长度消息）

H：哈希函数

h：固定长度的哈希值

2.基本特点

一个优秀的哈希算法应具备正向快速、逆向困难、输入敏感、强抗碰撞性等特征。

a.正向快速

正向快速是指对于给定的数据，在输入到输出的过程中，能够在极短的时间内计算出哈希值。

b.逆向困难

逆向困难是指无法在短时间内根据哈希值计算出原始数据。这一特性是哈希算法的安全性基础。

c.输入敏感

输入敏感是指在输入信息发生非常微小变化变化的情况下，经过计算得出的哈希值会与原数据计算得出的哈希值产生巨大的区别。因此无法根据变更数据前后的哈希值推测出原始数据发生了怎么样的变化，同时也是检验两组原始数据是否相同的方法之一。

d.强抗碰撞性

对不同的关键字可能得到同一散列地址，即key1≠key2，而H(key1)=H(key2)，这种现象称碰撞。不同的输入很难输出相同的哈希值。当然，由于哈希算法的输出位数是有限的，但输入是无限的，所以不存在永远不发生碰撞的哈希算法。但只要保证哈希算法发生碰撞的概率足够小，哈希算法仍可被使用。优秀的哈希算法只要保证找到碰撞的输入信息所耗费的代价远大于收益即可。

哈希算法的以上特性，保证了区块链的不可篡改性。对于一个区块链的所有数据通过哈希算法得到一个哈希值，但是通过哈希值无法反推出原区块数据，区块链上的哈希值可以唯一、准确地表示一个区块，任何节点通过简单快速的对区块数据内容进行哈希计算都可以得到哈希值，流程图见下图2-8。

原始数据

哈希函数

AXSJ765SNWCRVRVLFFAPNRDSNZG4VUSU2HAS7DJWZR2U

哈希算法

防碰撞性

输出长度唯一性

不可逆性

仅仅是单向的

不可进行反向计算

输出值被称为哈希值，可以称作信息摘要，或者数字指纹；

**图2-8哈希函数**

3.防篡改

区块头包含了上一个区块数据的哈希值，这些哈希值层层嵌套，最终将所有区块串联起来，形成区块链。区块链里包含了自其诞生以来的所有交易，因此要篡改一笔交易就要将其后所有区块的父区块的哈希值全部篡改，运算量非常大。如果要进行数据的篡改，必须要伪造交易链，保证连续伪造多个交易，同时使得伪造的区块在正确的区块产生之前出现。只要网络中的节点足够多，连续伪造的区块运算速度超过其他节点几乎是不可能实现的。另一种伪造区块链的方式为某一方控制全网超过50%的算力。因为区块链的特点为少数服从多数，一旦某一方控制全网超过50%的算力，即可篡改历史交易。但是，在区块链中，只要参与的节点足够多，控制全网超过50%的算力几乎是不可能做到的。即使某一方真的拥有全网超过50%的算力，那么这一方即是获得利益最多的一方，必定会维护区块链的真实性[13]。

4.快速检测

在区块链中，哈希函数除了具有防篡改的特性，基于哈希函数构建的Merkle树，可以实现内容改变的快速检测，也在区块链中发挥着重要的作用。Merkle树是由Ralph Merkle提出的一种用于验证数据完整性的数据结构。

Merkle树的本质就是一种哈希树。在区块链中Merkle树就是当前区块所有交易信息的一个哈希值。其构建示意图如图2-9所示。Merkle树通常包含区块体的底层（交易）数据库，区块头的根哈希值（即Merkle根）以及所有沿底层区块数据到根哈希的分支。Merkle树运算过程一般是将区块体的数据进行分组哈希，并将生成的新哈希值插入到Merkle树中，如此递归直到只剩最后一个根哈希值并记为区块头的Merkle根。最常见的Merkle树是比特币采用的二叉Merkle树，其每个哈希节点总是包含两个相邻的数据块或其哈希值，其他变种则包括以太坊的默克尔patricia tree等。Merkle树有诸多优点：例如Merkle树可支持“简化支付验证”协议，即在不运行完整区块链网络节点的情况下，也能够对(交易)数据进行检验。例如，为验证下图中交易6，一个没有下载完整区块链数据的客户端可以通过向其他节点索要包括从交易6哈希值沿默克尔树上溯至区块头根哈希处的哈希序列（即哈希节点6，5，56，78，5678，1234）来快速确认交易的存在性和正确性。一般说来，在N个交易组成的区块体中确认任意交易的算法复杂度仅为log2N。这将极大地降低区块链运行所需的带宽和验证时间，并使得仅保存部分相关区块链数据的轻量级客户端成为可能。



**图2-9 Merkle树结构示意图**

Merkle树的叶子节点的值是数据集合的单元数据或者单元数据哈希。

由Merkle树的构造可知，如果查看两个文件是否相同，只需要比较两个文件的根哈希即可，如果想进一步找出两个文件的不同处，则可以根据树节点哈希值从树的根哈希开始，比较左右两个子树的根哈希，左子树根哈希不同的话就比较左子树的数据，右子树根哈希不同就比较右子树的数据。然后以此类推，不断加大比较的深度，直到找到不同的数据块，最终可以准确识别被篡改的交易信息[14]。

在实际应用中，采用这种Merkle树找不同的数据块，可以不完全下载全部的数据。在数据访问带宽较低的场景中，例如P2P网络下载中，可以较快的比较数据的缺失部分，降低重复下载的数据量。因此，在一些网络传输速率不高的场景中，Merkle树的应用能有效提高系统的数据交互效率。

（四） 数字签名

数字签名（又称公钥数字签名）是一种类似写在纸上的普通的物理签名，但是使用了公钥加密领域的技术实现，用于鉴别数字信息的方法。一套数字签名通常定义两种互补的运算，一个用于签名，另一个用于验证。数字签名，就是只有信息的发送者才能产生的别人无法伪造的一段数字串，这段数字串同时也是对信息的发送者发送信息真实性的一个有效证明。

数字签名是非对称密钥加密技术与数字摘要技术的应用，即每个节点需要一对私钥、公钥密钥对。私钥为发送者所拥有的密钥，签名时需要使用私钥。不同私钥对同一段数据的签名是完全不同的。数字签名一般作为额外附加信息附加在原消息中，以验证信息发送者的身份。公钥即所有人都可以获取的密钥，验签时需要使用公钥。

数字签名的具体流程如下：a.发送者对原始数据通过哈希算法计算数字摘要，并且发送者使用非对称性密钥中的私钥对数字摘要进行加密，形成数字签名；b.发送者将数字签名和原始数据一同发送给验证签名的接收者。

举例来说数字签名技术就是，对传输文本进行摘要採作，从“我爱批京天安门”这个字符串使用哈希算法，抽取出例如“天”这个字作为文章摘要，我把“天”字用私钥加密作为数字签名附在文本传输给对方。对方收到文件后，也对文本进行同样的哈希操作，得到“天”，再用公对我的签名进行解，如果也得出“天”，那么说明，整传输是可信的，如果不是，则传输过程有可能受到了篡改。

验证数字签名的具体流程如下：a.接收者首先要具有发送者的非对称性密钥的公钥；b.在接收到数字签名与发送者的原始数据后，先使用公钥，对数字签名进行解密处理，得到原始的摘要值；c.接收者对发送者的原始数据进行同样的哈希算法计算摘要，将其与解密处理后得到的摘要值进行对比，如果二者相同，则签名验证通过，保证了原始数据在传输过程中未经过篡改。签名及验证过程的示意图如图2-10、图2-11所示。

数据

散列函数

数字签名的数据

附加到数据

散列值

认证

签名

私钥加密

**图2-10签名过程示意图**

散列函数

散列值1

公钥验签

散列值2

？

若散列值1=散列值2，则此数字签名验证通过

数字签名的数据

签名

数据

**图2-11验证过程示意图**

（五）多重签名

在数字签名应用中，有时需要多个用户对同一个文件进行签名和认证。在一个多重签名体制中，所有参与签名的相对独立而又按一定规则关联的实体的集合，我们称为一个签名系统。签名子系统就是所有签名者的一个子集合。签名系统中的任何一个子系统的各成员按照特定的承接关系对某个文件进行签名，这个承接关系就称为这个签名系统的一个签名结构。

按照签名结构的不同，多重数字签名分为两类：有序多重签名，即签名者之间的签名次序是一种串行的顺序，和广播多重签名，即签名者之间的签名次序是一种并行的顺序。后来，提出了具有更一般化签名结构的签名方案：结构化多重签名。在结构化签名方案中，各成员按照事先指定的签名结构进行签名。

根据签名过程的不同，多重签名方案可以分为两类：有序多重数字签名方案和广播多重数字签名方案。每种方案都有三个过程：系统初始化、产生签名和验证签名。每种方案都包含三个对象：消息发送者、消息签名者和签名验证者。广播多重数字签名方案中还包含签名收集者[15]。

**代码示例**

不妨通过OpenSSL来演示非对称加密以及秘钥对以及生成。OpenSSL是一个开放源代码的软件库包，可以进行安全通信，避免窃听，同时确认另一端连接者的身份。这个包广泛被应用在互联网的网页服务器上。比特币中涉及的哈希计算以及生成私钥公钥操作都可以在OpenSSL命令完成。

我们选择在Linux发行版Ubuntu的Bash下进行演示。安装OpenSSL命令

~ sudo apt-get install openssl

当安装好OpenSSL后，首先使用sha256sum程序进行哈希单向加密的演示。分别将字符串“Hello,Blochchain!”以及“Hello,blochchain!”写入文本Bc.txt和bc.txt。

~ cat bc.txt

Hello,blochchain!

~ cat Bc.txt

Hello,Blochchain!

然后在文本所在路径输入sha256sum Bc.txt，sha256sum bc.txt

~ sha256sum Bc.txt

07cbbca7364be23e80da0ccd171a700ff8ad746ea6a7a70e6559c8849b8ad46f Bc.txt

~ sha256sum bc.txt

27e3697b18399ce8efd826a0858788588282a2db293f23edb4ebe444d10c0d21 bc.txt

可以看出两个文本中的内容被“压缩”成32个16进制数，并且字符大小写的差别就产生哈希值巨大的区别。读者可以拿任意数字资料通过SHA256的算法进行数字摘要操作，例如软件安装包和视频文件。除了SHA256算法，比特币还使用了ripemd160哈希算法，可以通过openssl的标准命令为 dgst 进行，不过抗碰撞性不如SHA256，具体用法如下：

~ openssl dgst -ripemd160 bc.txt

另外，openssl dgst支持的算法有：[-md5|-md4|-md2|-sha1|-sha|-mdc2|-ripemd160|-dss1]

以RSA密钥对作为示例。首先可以使用openssl中的标准命令 genrsa随机生成私钥，然后再使用标准命令rsa从私钥中提取公钥。

在当前路径下就保存了随机生成的私钥文件private.key。

～ openssl genrsa -out private.key

通过随机生成的私钥文件生成公钥文件public.key。

～ opesnssl rsa -in private.key -pubout -out public.key

通过openssl 使用RSA加密算法进行加密、解密、签名和验证等运算。 标准命令是rsautl，使用示例如下：

使用公钥对明文进行加密。其中-encrypt表示进行加密，-in message.txt输入明文, -in keypublic.ke公钥文件，-pubin 根据公钥进行加密,-out encrypt.txt输出的加密文件名(合规的文件名即可)。

～ openssl rsautl -encrypt -in message.txt -inkey public.key -pubin -out encrypt.txt

使用私钥对加密文进行解密。其中-decrypt表示进行解密 -in encrypt.txt输入加密文, -inkey private.key输入私钥文件，-out decrypt.txt输出的解密文件名

～ openssl rsautl -decrypt -in encrypt.txt -inkey private.key -out message.txt

使用私钥对明文钥进签名。其中-sigh表示进行签名操作，-in message 进行签名的明文，-inkey private.key输入私钥文件，-out sign.bin输出的签名信息。

～ openssl rsautl -sign -in message.txt -inkey private.key -out sign.bin

使用公钥对签名进行验签，其中-verify表示进行验签操作，-in sign.bin输入签名信息，-inkey public.key输入公钥文件，-pubin根据公钥进行验签，-out verify.txt表示验证结果。理论上是需要明文和签名文件两个，这里验签所需要的信息都包含在sign.bin。

～ openssl rsautl -verify -in sign.bin -inkey public.key -pubin -out verify.txt

比特币采用椭圆曲线的可以通过openssl标准命令ecpam实现，使用过程与RSA一样。

**三、共识机制**

区块链的信任问题通过分布式账本解决，加密技术是区块链数据的不可篡改的技术基础，共识机制则是可以实现区块链达成一致性的重要方法。在传统的中心化帐本中，存在权威中心，各参与者以中心数据为准，对其数据进行复制即可。但是在区块链的去中心化的分布式账本中，并没有这样的权威中心存在，每个参与者都可以进行数据的输入。这种方式虽然避免了中心化账本所产生的腐败问题，但也会引入许多问题，比如：参与者来自世界各地不同地区，彼此之间相互不熟悉，甚至互不相识，参与者有可能会上传虚假或恶意数据，以使其从中获利，那么如何保证他添加的账本数据是正确的、可信的，为了解决这些关键问题，共识机制由此而生[16]。

（一） 基本介绍

由于加密货币多数采用去中心化的区块链设计，节点是各处分散且平行的，所以必须设计一套制度，来维护系统的运作顺序与公平性，统一区块链数据的版本，并奖励提供资源维护区块链的使用者，以及惩罚恶意的危害者。这样的制度，必须依赖某种方式来证明，是由谁取得了一个区块链的打包权（或称记账权），并且可以获取打包这一个区块的奖励；又或者是谁意图进行危害，就会获得一定的惩罚，这就是共识机制[17]。

简单来说共识机制就是面对一个原本并不认识的人，但是通过一个机制对进行检测，如果这个人通过了检测认定，那么你就基本可以认为这个人是可信任的。

（二）基本特点

区块链中可以支持不同的共识机制，但是不同的共识机制需要具有以下两个性质，即一致性与有效性。

1.一致性

所有诚实节点保存的区块链的前缀部分完全相同。

2.有效性

由诚实节点发布的消息最终将被所有其他诚实节点记录到自己的区块链中。

（三）评价标准

除了满足一致性与有效性这两个基本特点外，不同的共识机制在区块链上的应用时，还会对整个区块链产生其余影响，所以可以从以下四个标准评价共识机制。

1.资源消耗

资源消耗是指共识机制在运行的过程中所消耗的资源。例如：共识机制需要利用计算机来达到共识的目的，在分布式账本的各方参与者在达成共识的这个过程中，系统在计算的过程中会对资源产生一定量的消耗。例如：内存与CPU等计算资源的消耗。举例来说，在采用工作量证明机制的比特币系统中，需要消耗大量的计算资源进行挖矿，提供信任证明完成共识。

2.性能效率

在区块链上进行的交易，其性能效率是指从交易达成到交易数据信息被上传到分布式账本上，一直到上传的数据交易信息通过认证的时间。与通过第三方平台交易相比，区块链通过共识机制达成一致，在完成交易的性能效率问题一直备受关注，是当前区块链研究的重中之重。以比特币系统为例，目前每秒最多只能够处理7笔交易数据，这一数量远远不能满足现今区块链的需求，所以提高共识机制交易的性能效率是区块链当前亟需解决的问题之一。

3.扩展性

在区块链中，扩展性也是在设计的过程中需要重点考虑的问题之一。扩展性是指网络节点的扩展。在区块链中的扩展性主要针对两个部分数量的增加，一是指参与者数量，另一部分是指交易数量。区块链的扩展性需要考虑参与者和交易数量在增加的过程中，系统能否承载大量的数据量的增加，同时还要考虑在传输大量数据时，网络、设备、端口等其他设施能否保持高效传输，这种能力通常以网络吞吐量来衡量。因此，扩展性也是共识机制优劣的评价标准之一。

4.安全性

安全性的优劣是指区块链是否具有良好容错能力。例如：能否有效防止双重支付、自私挖矿等恶意攻击。双重支付与自私挖矿是区块链中存在的两种最大的安全问题。自私挖矿是一种针对比特币工作量证明机制区块链的挖矿策略，简单说就是挖到区块先不公布，而是继续挖矿，然后根据策略择机公布。而这种策略，根据研究者们的探讨，实际上会降低网络验证区块的速度，同时会削弱诚实矿工的盈利能力，而在难度调整之前，这也会对自私矿工本身带来不利影响。除此之外，区块链中还存在其余的安全性问题，例如，对交易对象的网络进行攻击，形成网络分区，对交易信息产生阻隔作用。或者通过产生大量的无意义节点，影响系统的安全性[18]。

（四） 基本分类

目前，区块链的共识机制主要可以分为以下四类：工作量证明机制（Proof of Work，PoW）、权益证明机制（Proof of Stake，PoS）、股份授权证明机制（Delegated Proof of Stake，DPoS）、拜占庭容错类机制（Byzantine Fault Tolerance，BFT）。

1.工作量证明机制（Proof of Work，PoW）

工作量证明机制是分配一段时间内交易信息的打包记账权，从而达成系统共识的机制。最早工作量证明机制的提出是为了防止垃圾邮件。这种证明机制应用于比特币系统中，其核心就是通过节点的算力选取打包节点。节点通过计算随机哈希散列的数值争夺上传数据的权利。比特币系统中，这种共识机制能够保证所有节点对一个待确认的交易达成一致。只有完成工作量证明的节点才能够提出这一阶段的待定区块，其余网络节点在此之上将继续完成工作量的证明，以产生新的区块[19]。工作量证明机制的基本工作步骤：

a.节点对所有的数据进行检测，将通过验证的数据记录并暂存。

b.计算节点通过尝试不同的随机数进行哈希运算，直至找到形成符合指定条件的随机数，该过程需要消耗节点自身的算力。

c.找到合理的随机数后，生成新区块，输入区块头信息后记录其余数据信息。

d.将新生成的区块对外公布，使得其他节点验证数据信息，经过验证后，将这些数据信息添加至区块链中，这些节点继续进行工作量的证明并继续生成新的区块链。

以比特币区块链为例，验证节点通过对随机数进行运算，争夺比特币的记账权，进行运算的过程中需要消耗算力等资源，因此验证节点也被称为“矿工”。尝试不同随机数，找寻合适随机数的过程称为“挖矿”。如果说，两个节点在同一时间找到区块，那么网络将根据后续节点和区块的生成情况决定以哪个区块为最终的区块。工作量主要体现在：要找到合理的随机数需要进行大量的尝试性计算，找到合理的随机数是一个概率事件，表明在找到合理的随机数之前进行了大量的工作。

工作量证明机制对于计算问题的选取需要满足以下几点性质：

a.伪随机性

保证节点在工作量证明，找到合适的随机数仅依赖于自身算力，保证相对公平。

b.难度可控

要根据实际情况，选择难度合适的问题进行计算，如果选择难度过高的问题计算，会导致算时间过长，影响效率。如果如果选择难度过低的问题计算，会产生分叉，影响系统的一致性。

c.可公开验证

由于区块链是去中心化，要求计算结果可以通过简洁的操作进行公开验证。

工作量证明机制具有完全去中心化的优点，在以工作量证明机制为共识的区块链中，节点可以自由进出。但同时这种共识机制存在的缺陷也是十分明显的。

d.效率低

首先，产生每个新区快需要耗费时间，同时新产生的区块需要若干区块，经过后续区块的认证才能保证有效，这将花费更长的时间，严重影响到系统的效率。

e.消耗与浪费

在这种工作量证明机制中，实际工作中以寻找合适的随机数为主，并不是记录账本数据，这也就导致了用于计算随机数的资源与能量消耗巨大，该过程的计算过程都是无意义的，是一种浪费现象，同时，达成共识的周期较长。

f.算力集中化凸显

由于工作量机制的运行原理可知，“挖矿”过程本质上就是比拼算力的能力，自然也就导致了算力集中的问题。目前，普通的个体或者几十台规模的矿机由于算力不足，很难挖到矿，这导致了各方联合起来进行挖矿，这些算力集中的地方称为矿池。以比特币的Ghash矿池为代表，其算力已经接近比特币算力的一半，这使得其余用户很难获得同样规模的算力来维持自身的安全。

2.权益证明机制（Proof of Stake，PoS）

权益证明机制（Proof of Stake，PoS）最早应用在2012年，化名Sunny King的神秘黑客推出了Peercoin，该加密电子货币采用工作量证明机制发行新币，采用权益证明机制维护网络安全，这是权益证明机制在加密电子货币中的首次应用。权益证明机制同样是一种挖矿游戏，但是通过节点拥有加密数字货币的数量和时间来降低寻找随机数的速度。权益证明机制由系统中具有最高权益的节点获得记账的权力，而不是算力最高的节点。简单来说，权益证明机制就是一个根据持有数字货币的数量与时间，进行利息发放和区块产生的机制。在这种机制下，产生了一个新的名词—币龄，币龄的值为持有数字货币的数量与持币时间的乘积，在产生新的区块时，币龄就会被清零，同时也可以从区块中获得一定的利息。这样就可以保证区块链的有效性是由具有经济效益的节点来保障，拥有更多币龄的节点将会有更大的概率决定下一区块[20]。

权益证明机制虽然在一定程度上解决了工作量证明机制的浪费资源的问题，同时缩短了区块产生的时间，提高了系统效率。但是，这种共识机制本质上仍需要网络中的节点进行挖矿的情况，并没有从根本上解决工作量证明机制的问题。同时，在网络的同步性较差情况下，权益证明机制产生的多个区块容易产生分叉现象，影响系统的一致性。如果有恶意的节点获得了记账的权力，那么就可以通过控制网络通信，形成网络分区，向各网络分区发送不同的区块，就会造成网络分叉，从而进行双重支付，使系统的安全性受到严重威胁。

3.股份授权证明机制（Delegated Proof of Stake，DPoS）

权益证明机制是使用一个确定的算法随机选择持币节点去产生新的区块，节点的币龄越大，去产生新的区块的概率就越大。但是这种机制并没有解决区块链的安全性与公平性，持有币龄多的少数节点会决定新区块的产生，同时，如何快速高效达成共识也是一个严峻的考验。为了解决这些问题，产生了一种新的保障网络安全的共识机制--股份授权证明机制（Delegated Proof of Stake，DPoS）。股份授权证明机制与董事会投票机制相似，机制内部存在一个投票系统，股东们进行投票以决定公司决策，并非该区块链的所有节点。在股份授权证明机制下，每个持币节点可以进行投票选举，选出一定数量的节点作为代表，由这些代表代替全体节点进行投票等操作，维持区块链的运行，一旦产生交易，这些代表们会获得一定的报酬。同时，如果代表节点的工作存在危害区块链的行为，所有节点可以同通过投票取消其代表资格，然后所有重新投票，选取新的代表。

在股份授权证明机制中，首先是成为代表，这需要在网上注册公钥，然后分配到一个一个特有的标识符。该标识符被每笔交易数据的头部引用。接下来是投票选择代表，各节点可以选择多个代表，同时可以实时查询所选择代表的操作，如果发现代表的表现过差，错过许多区块，那么，可以再次进行投票选择新的代表。最佳区块链是最长的有效区块链，如果错过了产生新区块的机会，就会意味着你已经落后于竞争对手，因为你的区块链会短于竞争对手。

该机制可以及时发现网络分叉的问题。如果交易被写入区块后有51%以上被生产出来，则可以认为是在主区块链上，如果错过了50%以上，则有可能是在支链上，一旦写入支链，应该停止交易，解决分叉问题。

虽然股份授权证明机制解决了工作量证明机制和权益证明机制的问题，减少了参与记录数据节点的数量，节省时间，提高了效率，可以达到秒级的共识验证。但是，这种共识机制一般还无法脱离代币运行，但实际商业应用并不需要代币。因此，股份授权证明机制也不能完全解决区块链在商业中的应用问题。

4.拜占庭容错类机制（Byzantine Fault Tolerance,BFT）

前几种证明机制都是将其余所有节点都视为对手，每个节点都需要进行计算或提供凭证以获取利益。但是拜占庭容错类机制（Byzantine Fault Tolerance，BFT）是希望所有节点共同合作，以协商的方式产生被所有节点都认可的区块。

拜占庭容错类问题最早于1982年被Lselie Lamport等人在The Byzantine Generals Problem一文中提出，主要讲述的是分布式网络节点通讯的容错问题。针对此问题所提出的诸多解决办法都被统称为拜占庭容错类机制。

在拜占庭容错类机制中，当拜占庭节点（失效节点）不超过总节点的1/3时，拜占庭将军问题[[7]](#footnote-7)才能得到解决。实用拜占庭容错机制（Practical Byzantine Fault Tolerance，PBFT）是最经典的一种拜占庭容错类机制。在保证安全的前提下，在有（3n+1）个节点的区块链中，失效节点不超过n，也就是说可以提供n个容错性。

在实用拜占庭容错机制中可以实现区块链的一致性，同时避免多余的计算量，节省资源，极大地缩短了达成共识的时间，可以达到秒级共识，提高效率，并且系统可以脱离代币进行运转，基本可以达到商业应用的要求。此外，只有主节点可以产生并发送新区块的信息，其余节点只起到验证信息准确性的作用，避免分叉。

但是，拜占庭容错机制在安全性与扩展性方面还存在不可避免地问题。其安全性主要依赖于失效节点数量的限制，当有1/3及以上节点失效时，系统将无法正常工作。当系统中存在或超过1/3的节点联合起来发布恶意信息时，会使系统的安全性和一致性遭到破环。同时拜占庭容错机制依赖参与节点的数量，因此该机制不适用于节点数量过于庞大的区块链，扩展性差。若主节点为了自身利益，散布虚假信息，提出无效区块链，则不会产生新的区块，造成时间的浪费，影响效率[21]。

除了以上四种验证机之外，Pool验证池也很重要，Pool验证池基于传统的分布式一致性技术建立，并辅之以数据验证机制，是之前区块链中广泛使用的一种共识机制。但随着私有链的减少，这种机制被使用的次数也逐渐减少。

Pool验证池不需要依赖代币就可以工作，在成熟的分布式一致性算法(Pasox、Raft)基础之上，可以实现秒级共识验证，更适合有多方参与的多中心商业模式。同时，相较于其余验证机制，Pool验证池的去中心化程度不如工作量证明机制等。

对工作量证明机制、权益证明机制、股份授权证明机制和实用拜占庭容错机制的各个方面进行对比，得到结果如表2-2所示。

**表2-2 常用共识算法性能对比**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 共识机制 | 性能效率 | 资源消耗 | 容错率/% | 去中心化程度 | 扩展性 | 一致性 |
| PoW | 低 | 高 | 50 | 高 | 差 | 差 |
| PoS | 较高 | 低 | 50 | 高 | 良好 | 差 |
| DPoS | 高 | 低 | 50 | 低 | 良好 | 良好 |
| PBFT | 高 | 低 | 33 | 低 | 差 | 良好 |

现今来说，对于区块链而言，还没有一种共识机制可以使其各个方面都做到完美无缺，各种机制或多或少都存在一些缺陷，在“不可能三角”[[8]](#footnote-8)评价体系中，任何共识机制都不能在3个方面都达到最佳状态。因此需要根据该区块链系统需要实现的目标，对比各种共识机制，根据实际需求，进行权衡，选择最符合的共识机制，进行应用。工作量证明机制在去中心化和容错率方面较好，但可用性较低；权益证明机制在节能方面较优，但不够灵活；股份授权证明机制可用性于容错率较高，但去中心化程度较低；实用拜占庭容错机制在去中心化和容错率方面较好，但扩展性差。

为了更好应对共识机制在实际应用将要面对的困难与挑战，采用两种或多种机制相互结合的混合机制，也是解决单种共识机制在某些方面不能够达到完美的有效手段之一。比如，工作量证明机制和权益证明机制结合、权益证明机制和实用拜占庭容错机制结合。

a.工作量证明机制和权益证明机制结合

2012年诞生Peercoin，采用工作量证明机制发行新币，采用权益证明机制维护网络安全，这是工作量证明机制和权益证明机制结合的典型案例。利用权益证明机制可以较少系统的资源消耗，提高公平性与安全性。简单来说在该机制中，节点尝试完成工作量证明，提出新的区块，随后由完成权益证明的节点对新区块进行验证。具体来讲区块持有节点通过消耗币龄获得利息，同时具有生成新区块和用权益证明机制造币的优先权。此外，第一个输入信息的权益核心需要符合某一协议，具有一定的随机性，但与工作量证明机制的区别在于这一过程是在有限空间内完成而不是像工作量证明机制实在无限区域内随机寻找。

这种机制的安全性也会得到提高，可以更好的防止分叉问题，每个区块的交易信息都会将消耗的币龄提供给自身，消耗的币龄高的区块将在主链上，因此，对于恶意攻击者来说，必须要控制大量的币龄并且同时拥有超过50%的算力，这将大大增攻击的成本，同时攻击过程中币龄的消耗也会降低进入主链的几率。

在该机制中，只要是拥有币龄的节点，无论数量的多少，都可以进行区块的挖掘，避免矿池的产生，防止算力的集中。

b.权益证明机制和实用拜占庭容错机制结合

这种混合机制通过权益证明机制限制参与实用拜占庭容错机制节点的数量，可以提高系统的扩展性。具体工作过程如下：通过权益证明机制选出代表节点，提出新的区块，然后再通过权益证明机制选出新的代表节点对新区块进行验证，经过有限次的重复后，通过实用拜占庭容错机制达成一致，这样既解决了权益证明机制一致性差的缺点，同时也解决了实用拜占庭容错机制扩展性差的缺点。

由于每种共识机制都在某些方面存在不足，如何将各种共识机制有效的相互结合，弥补各自的不足，将会是今后共识机制的发展趋势。

**代码示例**

比特币中实现PoW共识算法源码位于/src/pow.cpp，共有设置有三个函数CheckProofofWork，CalculateNextWorkRequired以及GetNextWorkRequired，实现了判断是否挖矿成功以及调节挖矿的难度两种功能。其中判断挖矿是否成功通过CheckProofOfWork函数，核心即两次区块的哈希值是否满足前n位为0，通过与目标值比较大小即可。随着新的算力增大，哈希计算难度不变将导致出快速度爆增。比特币为了保证出块速度维持在每十分钟打包一个区块，通过CalculateNextWorkRequired和GetNextWorkRequired两个函数实现提高计算哈希目标值的难度。

CheckProofOfWork函数用来检查nNonce是否符合规则，使得当前的区块的双倍哈希值小于目标值，即哈希值uint256 hash前n位都为0，n根据挖矿难度进行调节，使得总是出块速度保持在十分钟左右。nBits表示当前的难度值，通过使用SetCompact函数将nBits转换为长度为256的类哈希值。该函数的第一个if语句，用来判断当前目标难度的转换哈希值是否符合要求，例如难度低于设定的极限值bnTarget > UintToArith256(params.powLimit))。若不满足第一个条件，则者修改nBits。第二个if语句就是判断当前nNonce的哈希值是否合规，只需要判断大小即可。当该函数返回路逻辑变量Ture时，说明该节点找到合适的nNonce，挖矿成功。若不满足第二个条件，则换下一个nNonce进行哈希计算再调用CheckProofWork函数进行判断。

bool CheckProofOfWork(uint256 hash, unsigned int nBits, const Consensus::Params& params)

{

bool fNegative;

bool fOverflow;

arith\_uint256 bnTarget;

bnTarget.SetCompact(nBits, &fNegative, &fOverflow);

// Check range

if (fNegative || bnTarget == 0 || fOverflow || bnTarget > UintToArith256(params.powLimit))

return false;

// Check proof of work matches claimed amount

if (UintToArith256(hash) > bnTarget)

return false;

return true;

}

难度的调整是在独立的节点中进行。在BIP9中（Bitcoin Improvement Proposals，是提出 Bitcoin 的新功能或改进措施的文件。可由任何人提出，经过审核后公布在 bitcoin/bips 上），选择每2,016个区块的时间作为参考值，实际时长通过时间戳来确定。难度的调整公式是由最新2,016个区块的花费时长与20,160分钟（两周内以10分钟一个出块速度所需要的时间）比较得出的。难度是根据实际时长与期望时长的比值进行相应调整的。简单来说，如果节点发现区块产生速率比10分钟要快时会增加哈希计算难度，如果比10分钟慢时则降低难度。

在CalculateNextWorkRequired函数中，与CheckProofOfWork类似的，除了调整哈希难度外，其余的都是限定条件。bnNew **\*=** nActualTimespan**;**与 bnNew **/=** params**.**nPowTargetTimespan**;**即为调整难度，当nActualTimespan实际时间比params**.**nPowTargetTimespan预期时长大时，将提高bnNew的数值，即降低判断当前区块哈希值是否满足前n位等于零中的n的数目。限定条件有限制有保持实际时长在理想值的(0.25,4)的区间内，不低于设置的哈希计算难度最低值等。

unsigned int CalculateNextWorkRequired**(**const CBlockIndex**\*** pindexLast**,** int64\_t nFirstBlockTime**,** const\

Consensus**::**Params**&** params**)**

**{**

**if** **(**params**.**fPowNoRetargeting**)**

**return** pindexLast**->**nBits**;**

// Limit adjustment step

int64\_t nActualTimespan **=** pindexLast**->**GetBlockTime**()** **-** nFirstBlockTime**;**

**if** **(**nActualTimespan **<** params**.**nPowTargetTimespan**/**4**)**

nActualTimespan **=** params**.**nPowTargetTimespan**/**4**;**

**if** **(**nActualTimespan **>** params**.**nPowTargetTimespan**\***4**)**

nActualTimespan **=** params**.**nPowTargetTimespan**\***4**;**

// Retarget

const arith\_uint256 bnPowLimit **=** UintToArith256**(**params**.**powLimit**);**

arith\_uint256 bnNew**;**

bnNew**.**SetCompact**(**pindexLast**->**nBits**);**

bnNew **\*=** nActualTimespan**;**

bnNew **/=** params**.**nPowTargetTimespan**;**

**if** **(**bnNew **>** bnPowLimit**)**

bnNew **=** bnPowLimit**;**

**return** bnNew**.**GetCompact**();**

**}**

GetNextWorkRequired函数检查是否到了难度调整周期。比特币规定，每14天（2周）调整一次难度。另外用来回溯前14天前的第一组区块，返回的数据将作为CalculateNextWorkRequired函数的输入

unsigned int GetNextWorkRequired**(**const CBlockIndex**\*** pindexLast**,** const CBlockHeader **\***pblock**,** const\

Consensus**::**Params**&** params**)**

**{**

assert**(**pindexLast **!=** **nullptr);**

unsigned int nProofOfWorkLimit **=** UintToArith256**(**params**.**powLimit**).**GetCompact**();**

// Only change once per difficulty adjustment interval

**if** **((**pindexLast**->**nHeight**+**1**)** **%** params**.**DifficultyAdjustmentInterval**()** **!=** 0**)**

**{**

**if** **(**params**.**fPowAllowMinDifficultyBlocks**)**

**{**

// Special difficulty rule for testnet:

// If the new block's timestamp is more than 2\* 10 minutes

// then allow mining of a min-difficulty block.

**if** **(**pblock**->**GetBlockTime**()** **>** pindexLast**->**GetBlockTime**()** **+** params**.**nPowTargetSpacing**\***2**)**

**return** nProofOfWorkLimit**;**

**else**

**{**

// Return the last non-special-min-difficulty-rules-block

const CBlockIndex**\*** pindex **=** pindexLast**;**

**while** **(**pindex**->**pprev **&&** pindex**->**nHeight **%** params**.**DifficultyAdjustmentInterval**()** **!=** 0 **&&\** pindex**->**nBits **==** nProofOfWorkLimit**)**

pindex **=** pindex**->**pprev**;**

**return** pindex**->**nBits**;**

**}**

**}**

**return** pindexLast**->**nBits**;**

**}**

// Go back by what we want to be 14 days worth of blocks

int nHeightFirst **=** pindexLast**->**nHeight **-** **(**params**.**DifficultyAdjustmentInterval**()-**1**);**

assert**(**nHeightFirst **>=** 0**);**

const CBlockIndex**\*** pindexFirst **=** pindexLast**->**GetAncestor**(**nHeightFirst**);**

assert**(**pindexFirst**);**

**return** CalculateNextWorkRequired**(**pindexLast**,** pindexFirst**->**GetBlockTime**(),** params**);**

**}**

**四、智能合约**

在交易过程中，往往会涉及到多方参与，在传统的交易模式中，各参与方都将交易过程中的数据信息储存在自己的数据库中，但这往往会导致整个过程浪费时间、过程冗余复杂、信息公开透明程度差、效率低下；同时，由于各方信息交流差，往往会导致参与各方数据不一致，存在某一方随意篡改数据的可能，相互之间信任度下降，但区块链技术的存在可以很好的解决这些问题，由于智能合约的存在，各程序按照预先设置好的规则自动完成，使得数据篡改的可能性大大降低，同时免去许多繁琐复杂的流程，节省时间，提高效率。

智能合约引入区块链中是发展过程的一个里程碑。区块链从在最初应用的单一数字货币，到现今融入金融的各个领域中，智能合约一直起着无可替代的重要作用，这些应用几乎都是以智能合约的形式运行在区块链的平台上。

（一）基本介绍

1995年，学者Nick Szabo最早提出智能合约（Smart Contract）的概念，并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。智能合约又称智能合同，一种旨在以信息化方式传播、验证或执行合同的计算机协议。允许交易在没有第三方的条件下进行，且交易是可信的，并且交易是单向不可逆的。简单来说，就是在满足一定条件的情况下就可以自动执行计算机程序。生活中举个例子，我们经常乘坐飞机会购买飞机延误险，但是真正延误之后，你可能还要拨打客服电话了解流程、在线下开证明、找保险公司，才能执行完你的延误险赔付。这时候，如果有了智能合约，输入条件，连线航班数据，就能够确保保险公司在航班延误之后就自动为你打款了。合约的执行不需要第三方参与，是自动执行，这就大大提高了社会经济活动的效率了。

（二）智能合约与区块链

智能合约这一概念自产生以来，就缺少一个能够良好运行的平台，但是区块链的问世，由于其去中心化等特性，很好的解决了智能合约运行的诸多问题。例如：确保智能合约一定能够被执行，且不会再执行的过程中被修改。智能合约在区块链上运行，所有节点都会严格按照既定的逻辑执行，如果恶意节点修改了逻辑，那么由于区块链的验证机制存在，修改后的逻辑不会被其余节点承认。区块链上的智能合约是在沙盒[[9]](#footnote-9)中的可执行程序，智能合约的各种操作与状态均需要通过共识机制记录在区块链上，由于区块链上的所有交易数据都是公开的，因此，智能合约处理的数据也是公开的，任意节点均可查看数据。智能合约与区块链二者紧密相连，相辅相成，智能合约为区块链提供了应用接口，使区块链可以构建信任的合作环境，是区块链的核心技术之一。同时，区块链为智能合约运行的平台。

（三）运行原理与环境

在区块链平台上运行的智能合约通常包含初始状态、转换规则、触发条件以及相应的操作，上传数据，经过共识机制的验证后，合约将在区块链上运行，区块链实时监测智能合约的运行状态，当有新的数据上传至区块链时，一旦满足智能合约的触发条件时，将根据预设的逻辑对其进行运算即共识机制将对该数据进行验证，一旦验证通过，数据的输入、运行状态及输出都将在区块链上记录。智能合约运行机制如图2-12所示。

智能合约

预置触发事件

预置相应规则

条件i--响应i

区块链

**图2-12智能合约在区块链上运行示意图**

智能合约需要在与外界环境隔离的沙盒中运行，合约运行环境与宿主系统之间、合约与合约间需要靠沙盒相互隔离，可提高智能合约的安全性，目前来说，区块链对沙盒的支持主要包括虚拟机和容器，这两种都可以有效的保证智能合约在沙盒中独立运行，相互之前不产生干扰。

智能合约是图灵完备的语言，具备强大的可编程能力；支持多种数据类型，int，string，map，array；支持判断，循环，跳转，分支，且可以应对停机问题；支持接口，继承等面向对象的特性。

（四）安全问题

目前来说，智能合约还存在诸多安全隐患，现实中的合约通常是由具有法律基础的专业人士进行编写，但是对于智能合约来说，大多数的智能合约都是由熟悉计算机语言的人员进行代码的编写，这些人员绝大部分可能对法律的知识知之甚少，因此，编写出的智能合约或多或少的会存在一些法律上的缺陷，存在一定的漏洞。此外，支持智能合约运行的区块链部分采用Java等高级语言编写，这些高级语言会存在一些不确定的指令，可能会导致分歧，影响系统的一致性。所以，对于智能合约程序的编写一定要慎之又慎，防止上述情况的出现。现今，许多区块链平台都对语言的不确定性进行了改进，比如Fabric引入先执行、排序，再验证写入账本的机制；以太坊也只允许各节点实用确定性的编写语言进行操作。随着区块链技术的不断发展，相信智能合约的编写会愈发严谨与规范，各专业人士的协作和自身知识储备量的提升，安全性问题将会得到有效的解决。

**代码示例**

以太坊上通过Solidity语言实现智能合约，下面是一种实现投票功能的智能合约示例。可以从示例代码看出，智能合约开头标注了solidity的版本号，合约名为Ballot。名为Voter的复合结构体用来表示一个选民，包含了该选民的机票权重unit weight，是否投过票的布尔值bool votred，选择的提案unit vote三种属性。名为Proposal的复合结构用来表示一个选票，包含了提案名称bytes32 name和计票数uint voteCount两种属性。

各种函数包括构造函数Constructor(…)，选择指定节点地址作为选民函数giveRightToVote(…)，委托他人投票的delegate(…)，对指定提案投票的vote(…)， 以及统计和现实最高得票数的winningProposal()，winningName()。调用构造函数Constructor生成需要的提案以供节点投票，指定调用Constructor的节点作为“主席”，同时通过传递字符串数组设置所有的Proposal 对象。address地址即由公钥计算得到的地址。当“主席”调用giveRightToVote函数指定节点作为选民时，需要将节点的地址传入该函数，进。作为选民的节点可以调用delegate函数将投票权授权给其他节点，也可以调用vote进行投票。每当选民调用vote后，被投票的Proposal结构体中unit voteCount将增加计数，同时考虑选民的权重sender.weight，代码为proposals[proposal].voteCount += sender.weight。 winningProposal得到最大投票数的提案的序号，然后winningName可通过winningProposal返回的序号提取Proposal的名称属性，代码为winnerName\_ = proposals[winningProposal()].name

从投票智能合约Ballot中，可以看出内置的地址变量address是区块链中对应的概念，但更它更像常见程序语言，例如定义结构体和函数、循环语句求最大投票数的提案。可以看出，通过智能合约拓宽了区块链除了发行通证以外的应用范围，编写去中心化的应用的难度也大大降低，同时搭配以太坊平台，大大降低了搭建底层区块链平台的难度，使得人们更加专注于扩展区块链的应用价值。

pragma solidity ^0.4.22;

contract Ballot {

struct Voter {

uint weight;

bool voted;

address delegate;

uint vote;

}

struct Proposal {

bytes32 name;

uint voteCount;

}

address public chairperson;

mapping(address => Voter) public voters;

Proposal[] public proposals;

Constructor (bytes32[] proposalNames) public {

chairperson = msg.sender;

voters[chairperson].weight = 1;

for (uint i = 0; i < proposalNames.length; i++) {

proposals.push(Proposal({

name: proposalNames[i],

voteCount: 0

}));

}

}

function giveRightToVote(address voter) public {

…

}

function delegate(address to) public {

…

}

function vote(uint proposal) public {

Voter storage sender = voters[msg.sender];

require(!sender.voted, "Already voted.");

sender.voted = true;

sender.vote = proposal;

proposals[proposal].voteCount += sender.weight;

}

function winningProposal() public view

returns (uint winningProposal\_)

{

uint winningVoteCount = 0;

for (uint p = 0; p < proposals.length; p++) {

if (proposals[p].voteCount > winningVoteCount) {

winningVoteCount = proposals[p].voteCount;

winningProposal\_ = p;

}

}

}

function winnerName() public view

returns (bytes32 winnerName\_)

{

winnerName\_ = proposals[winningProposal()].name;

}

}

1. 数字货币简称为DIGICCY，是英文“Digital Currency”（数字货币）的缩写，是电子货币形式的替代货币。数字金币和密码货币都属于数字货币（DIGICCY）。数字货币是一种不受管制的、数字化的货币，通常由开发者发行和管理，被特定虚拟社区的成员所接受和使用。欧洲银行业管理局将虚拟货币定义为：价值的数字化表示，不由央行或当局发行，也不与法币挂钩，但由于被公众所接受，所以可作为支付手段，也可以电子形式转移、存储或交易。 [↑](#footnote-ref-1)
2. 具体来说，在以太坊网络上的交易而产生的每一次计算，都会产生费用——这个费用是以称之为“gas”的单位来支付。 [↑](#footnote-ref-2)
3. 以纯文本描述内容，能够保存各种格式信息，可以用写字版，Word等创建。也称富文本格式（Rich Text Format，一般简称为RTF）是由微软公司开发的跨平台文档格式。大多数的文字处理软件都能读取和保存RTF文档。 [↑](#footnote-ref-3)
4. 对于每类特定的产品，都会有一些属于这一类产品特有的产品属性，这一类的属性的数掘，我们将其称之为富数据。比如，对于相机来说，变焦范围，分辨率等这一类属性，都属于富数据类属性。 [↑](#footnote-ref-4)
5. 数据加密标准算法（Data Encryption Standard，DES）是一种对称加密算法，很可能是使用最广泛的密钥系统，特别是在保护金融数据的安全中，最初开发的DES是嵌入硬件中的。通常，自动取款机（Automated Teller Machine，ATM）都使用DES。 [↑](#footnote-ref-5)
6. RSA加密算法是一种非对称加密算法。在公开密钥加密和电子商业中RSA被广泛使用。RSA是1977年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）一起提出的。当时他们三人都在麻省理工学院工作。RSA就是他们三人姓氏开头字母拼在一起组成的。 [↑](#footnote-ref-6)
7. 拜占庭将军问题（Byzantine failures），是由莱斯利·兰伯特提出的点对点通信中的基本问题。含义是在存在消息丢失的不可靠信道上试图通过消息传递的方式达到一致性是不可能的。因此对一致性的研究一般假设信道是可靠的，或不存在本问题。 [↑](#footnote-ref-7)
8. 不可能三角（Impossible trinity）是指经济社会和财政金融政策目标选择面临诸多困境，难以同时获得三个方面的目标。在金融政策方面，资本自由流动、汇率稳定和货币政策独立性三者也不可能兼得。 [↑](#footnote-ref-8)
9. 在计算机安全领域，沙盒（英语：sandbox，又译为沙箱）是一种安全机制，为运行中的程序提供的隔离环境。通常是作为一些来源不可信、具破坏力或无法判定程序意图的程序提供实验之用。 [↑](#footnote-ref-9)