**“区块链+网络货运”的关键技术研究**

**负 责 人：** 陈志立

**研 究 人 员：**庞雄韬、韦健、符捷、杨豪

**完 成 单 位：**华东师范大学 & 中交智运

**提 交 日 期：** 2022.03.01

目 录

一、引言 1

二、主子链多级组链技术 2

2.1 区块链 2

2.1.1 区块链概述 2

2.1.2 区块链层次概述 2

2.2 主子链架构 4

2.2.1 主子链架构介绍 4

2.2.2 主子链架构结构 4

2.2.3 现有案例-雪崩链 5

2.3 跨链机制 7

2.3.1 公证人机制 7

2.3.2 侧链/中继链机制 8

2.3.3 哈希锁定 9

2.3.4 跨链方案对比 11

2.4 链上数据监管 11

2.4.1 监管框架 11

2.4.2 数据监管 13

三、隐私保护算法及密钥生成技术 15

3.1 隐私保护概况 15

3.2 隐私保护技术 16

3.2.1 环签名 16

3.2.2 零知识证明 18

3.2.3 Merkle树 22

3.2.4 隐私交易通道保护 25

3.2.5 其他隐私保护技术 25

3.2.6 隐私保护技术使用与对比 29

3.3 密钥生成 29

四、身份及权限管理 32

4.1身份管理 32

4.1.1 身份标识 32

4.1.2身份认证 34

4.2权限管理框架 35

4.2.1 Fabric权限管理 35

4.2.2 物流链下的权限管理 42

4.3 区块链中数字签名认证方案 44

4.3.1 数字签名 44

4.3.2 商密标准的数字签名方案 46

4.3.3 国密标准的数字签名方案 54

4.3.4 哈希函数 62

4.4 认证方案对比 67

4.4.1 SM3与SHA-256 68

4.4.2 SM2与ECC-256 68

4.4.3 性能差异 68

五、高性能共识技术 70

5.1 共识机制概述 70

5.1.1 区块链中的共识机制 70

5.1.2 FLP定理和CAP定理 71

5.1.3 共识机制中的拜占庭问题 73

5.1.4 共识算法的分类方法 74

5.1.5 共识算法的评判标准 75

5.2 常见基于公有链的共识算法 75

5.2.1 POW共识算法 75

5.2.2 POS共识算法 76

5.2.3 DPOS共识算法 76

5.2.4 小结 77

5.3 基于联盟链的共识算法 78

5.3.1 故障容错协议 78

5.3.2 拜占庭容错协议 80

5.4 联盟链主流共识算法详解 83

5.4.1 典型故障容错共识算法Raft 83

5.4.2 典型拜占庭容错共识算法PBFT 88

5.5 高性能共识机制技术研究 94

5.5.1 有向无环图（DAG）结构 95

5.5.2 执行分片技术 97

5.5.3 可验证随机函数(VRF) 100

5.5.4 状态缓存技术 102

5.6 联盟链共识协议小结 103

六、敏捷应用技术 105

6.1 与云计算深度融合的区块链快速应用技术 105

6.2 基于 Docker 的区块链实现 106

6.2.1 Docker 106

6.2.2 Fabric在Docker中实现 107

6.3 基于 VM 的区块链实现方式 109

6.3.1 Ethereum在VM中实现 109

6.3.2 gas 计费机制 109

6.3.3 EVM 虚拟机 109

6.3.4 合约的创建与运行过程 109

七、结论 112

# 一、引言

本项目拟将区块链技术应用与网络货运平台，充分利用区块链信息技术开放透明、数据安全性高的优势，在现有的货运平台上，进一步实现网络货运平台业务数据的可视化管理及平台风险控制能力，解决货运物流行业存在的政府监管难、多主体间协同效率低及物流企业融资难等痛点。

通过对网络货运行业现状的深入调研，提出了基于“大数据+区块链”的新一代信息技术融合架构的技术框架，该架构计划采用以Hyperledger Fabric为底层的联盟链设施基础，在强调技术优势强融合的同时又确保功能模块的松耦合，充分发挥大数据技术在网络汇聚、存储、应用方面的优势和区块链技术得数据不可篡改性、可溯源性，实现系统的“存储即验证、服务即验真”。通过超融合跨链技术、高性能跨链访问技术和链上安全监管技术，在实现系统多平台，多生态的循环可扩展性的同时，增强传统区块链中所弱化的监管特征；通过发展隐私保护技术及安全密钥保护、数据加密算法和机制、身份认证、权限管理、数据可信采集和交互共享、分布式的消息总线和智能合约等核心技术，推动构建高安全、高可用、高性能的区块链网络货运平台。

区块链技术涉及到的内容较多，本文重点研究的关键技术简述如下：

1. 面向大规模应用的主子链架构多级组链技术
2. 自主可控安全隐私保护算法及可替换密钥生成保护技术
3. 高可用的区块链身份认证及权限管理技术
4. 基于高性能共识机制的数据可信交互技术
5. 敏捷应用技术

# 二、主子链多级组链技术

2.1 区块链

### 2.1.1 区块链概述

区块链是由复制状态机组成的分布式网络系统，类似于数据链的形式，后面的数据块引用单个祖先块，通常由其哈希值标识。当区块链增长时，状态机中包含新的块，并将其传播到网络中的所有参与节点，这样网络中的每个节点都具有所有事务的单一全局视图。区块链不是单一的技术，而是共识算法、密码学、网络、数据库等多种技术的复杂多学科混合。由于区块链的设计，基于区块链的应用程序和系统是分散的、可靠的、可跟踪的和不可变的。

### 2.1.2 区块链层次概述

Fabric是由 Linux 基金会维护的一个模块化、可拓展的区块链联盟链项目，它不依赖任何加密货币，它对有着共同目标（业务需求）但彼此不完全信息的实体之间的业务提供了保护，例如跨境电商、资金交易、溯源等。

在大部分公链中，架构为Order - Execute - Validate - Update State。如比特币区块链中，如果有一个新交易，会先采用 PoW 机制对 Block 进行排序，然后比特币网络中的每个节点逐个进行验证，最后更新状态。因为需要依序进行验证，这种方式决定了其执行效率相对较低。而Fabric采用了Execute - Order - Validate - Update State架构。如图2-1所示：

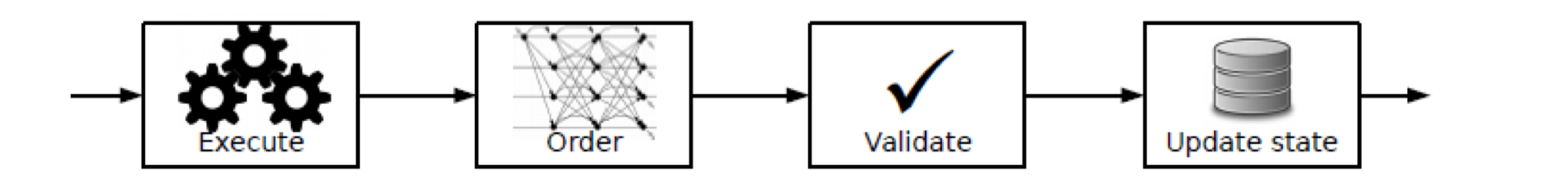


图2-1 Fabric执行架构

收到一笔新的交易后，首先提交至背书节点本地模拟交易执行（并背书），再将已背书交易排序并广播，各个节点对交易进行验证后更新状态。

总的来说，Fabric通过模块化、可插拔的架构来支持企业的复杂业务场景，通过身份验证（绑定现实身份）来弱化节点作恶，使用通道机制大大提升了系统的安全性和隐私保护。

区块链可以简单的分为三个层次，协议层、扩展层和应用层。其中，协议层又可以分为存储层和网络层，它们相互独立但又不可分割。在Fabric中，dapp处于最上层，它是整个区块链的展示层，通过RPC服务和智能合约（链码）层进行交换，区块链的应用层可以是移动端，web端，或是融合进现有的服务器，把当前的业务服务器当成应用层。所有的智能合约都运行在Docker上，并会用到RPC（Remote Procedure Call Protocol，远程过程调用直观说法就是A通过网络调用B的过程方法）。在链码和RPC下面是区块链的四大核心内容，包括：区块， 共识算法，挖矿以及网络层。

协议层，是最底层的技术。这个层次通常是一个完整的区块链产品，类似于我们电脑的操作系统，它维护着网络节点，仅提供Api供调用。网络编程、分布式算法、加密签名、数据存储技术等4个方面，分布式算法基本上属于业务逻辑上的实现，什么语言都可以做到，加密签名技术是直接简单的使用，数据库技术也主要在使用层面。

存储层：一个区块（Block） + 链表及区块链数据结构。

网络层：p2p网络，P2P网络的实现和并发处理才是开发的难点，所以对于那些网络编程能力强，对并发处理简单的语言，Nodejs开发区块链应用，逐渐变得更加流行，Go语言就开始流行了。

共识层：也就是激励层，在公链场景下往往通过挖矿机制，制定区块链的获取货币的方式。比如比特币用的是POW(Proof of Work工作量证明机制)：电脑的性能越好，越容易获取到货币奖励。还有POS(Proof of Stake权益证明机制)：类似于众筹分红的概念，会根据你持有的货币数量和时间，给持有者发放利息。在Fabric中，则没有这个激励措施。

扩展层：类似于电脑的驱动程序，Fabric中这部分包括了链码和Fabric CA两部分。是为了让区块链产品更加实用。

合约层：以往的区块链是没有这一层的。所以最初的区块链只能进行交易，而无法用于其他的领域或是进行其他的逻辑处理。但是合约层的出现，及智能合约”就是“可编程合约”，或者叫做“合约智能化”，其中的“智能”是执行上的智能，也就是说达到某个条件，合约自动执行使得在其他领域使用区块链成为了现实。

应用层：类似于电脑中的各种软件程序，区块链的展示层。DAPP产品其实也是这个层面的应用，将智能合约发布到区块链上，人们通过地址对这个区块进行访问继而和智能合约交互，以太坊使用的web3-js。区块链的应用层可以是移动端，web端，或是是融合进现有的服务器，把当前的业务服务器当成应用层。基于此开发的DAPP其实就是在智能合约基础上封装了一层Web应用，通过DAPP提供的Web应用和智能合约交互。

2.2 主子链架构

2.2.1 主子链架构介绍

考虑到越来越多的不同类型的服务将连接到区块链系统，单链体系结构很难支持复杂的需求。现有的解决方案多为单链优化。为了满足这种需求，需要一种异构的多链体系结构。在该体系结构中，服务被分配到不同的子链中，既保证了服务之间的私密性，又提高了整个系统的可扩展性和可扩展性。

目前大多数区块链系统都是单链架构，其中每个节点都是一个完整功能的单片节点。因此，每个节点都必须执行大量重复的计算任务，造成了能量的浪费。此外，当遇到流量峰值时，其性能下降更为明显。因此，单链架构已经不能满足日益增长的需求。因此主子链架构已经成为了一种能够解决现有问题的区块链架构模式。

2.2.2 主子链架构结构

主子链体系结构由单个主链和一组子链组成，用价值交换层连接子链和主链，以及子链之间的交互。与旨在创建具有全球交易账本的单一区块链的现有区块链架构相比，该架构允许一组区块链与另一组区块链并行运行，同时保持互操作性。

主链采用成熟的共识算法机制，而子链可根据业务需求设置适合自己场景的共识算法，通过原子交换或跨链节点与主链形成双向锚定的方式，与其他子链进行跨链交易。这种主子链机制可帮助子链在满足每秒数千级别性能的前提下同时获得主链所提供的最终一致性保障。

子链既可以是公链，也可以是联盟链，其链结构与主链类似，但因业务需求不同，可以采用不同的共识算法，达到数千的TPS性能，或者其他的定制开发功能。

图2-2示了多链体系结构的层次结构。该体系结构的核心是主链，它管理着许多半独立的子链。主链的设计原则是用最小的数据量、计算资源和网络带宽来提供一个可信的主网络。主链保存来自子链的事务元数据，但为了提高性能，不会包含细节。

子链设置在某种程度上类似于数据库分片技术。一般来说，子链独立于主链运行，除了它的创建，然后它们被合并回主链。在合并子链信息后，将这些子链的最终结果提交给主链，以使主链保持最新。类似地，当创建子链时，相关信息会从主链镜像到相应的子链，以便子链跟上主链的状态

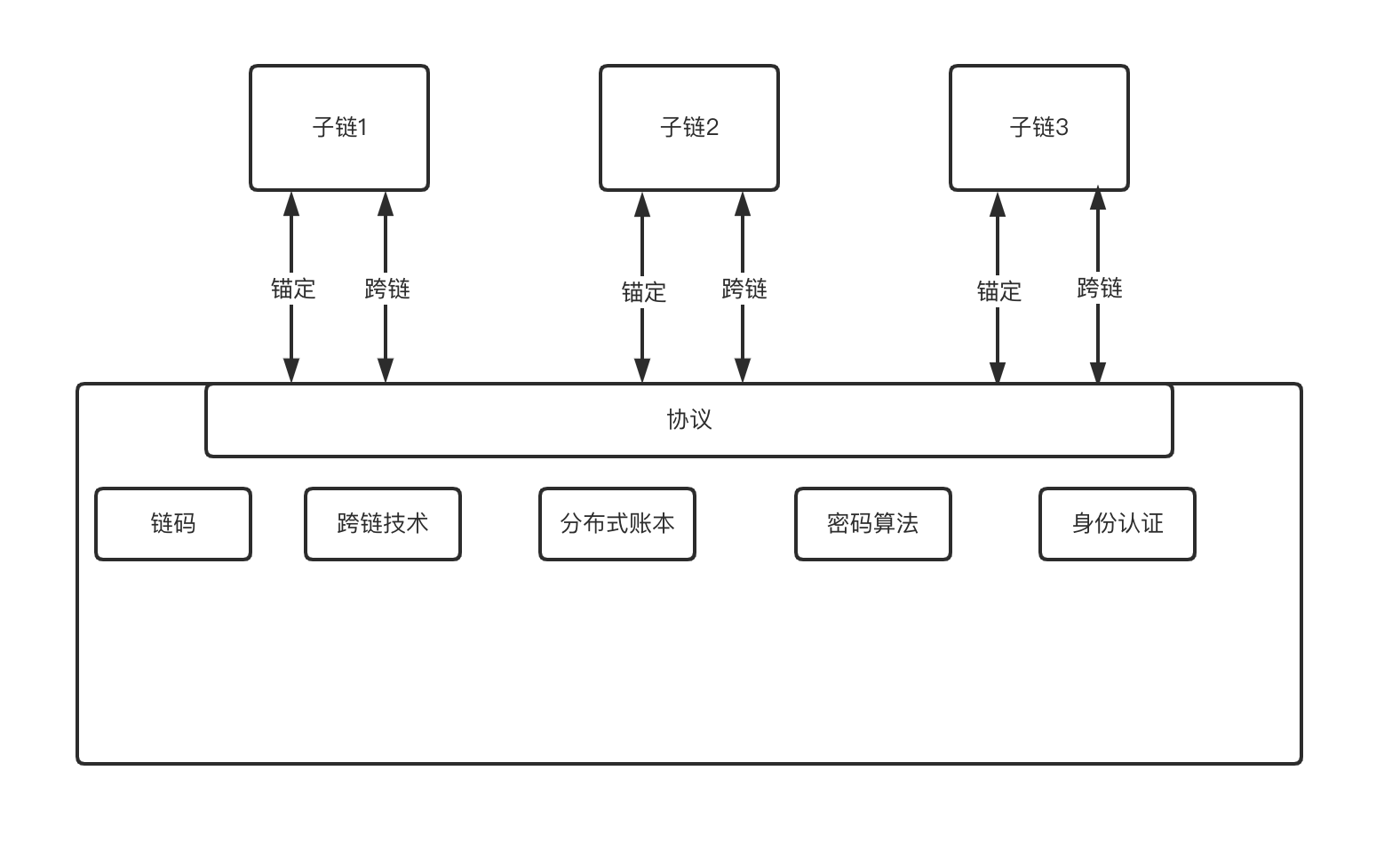


图2-2 多链体系结构

### 2.2.3 现有案例-雪崩链

与传统区块链不同，Avalanche 不是一条区块链，而是许多区块链的集合，类似波卡。更准确的说， Avalanche 由很多个子网构成，子网下有很多条的区块链。而在子网中，有一个特殊的子网，叫 Primary Network。它由3条区块链构成：

1. P链（Platform Chain）：主要负责质押通证、协调网络验证者、创建自定义子网。当需要参与质押获得挖矿奖励时，是在这个链上进行的。
2. X链（Exchange Chain）：主要负责点对点地创建和交易资产。
3. C链（Contract Chain）：用于创建智能合约。地址格式同以太坊，主要在这个链上参与各种Dapp项目。

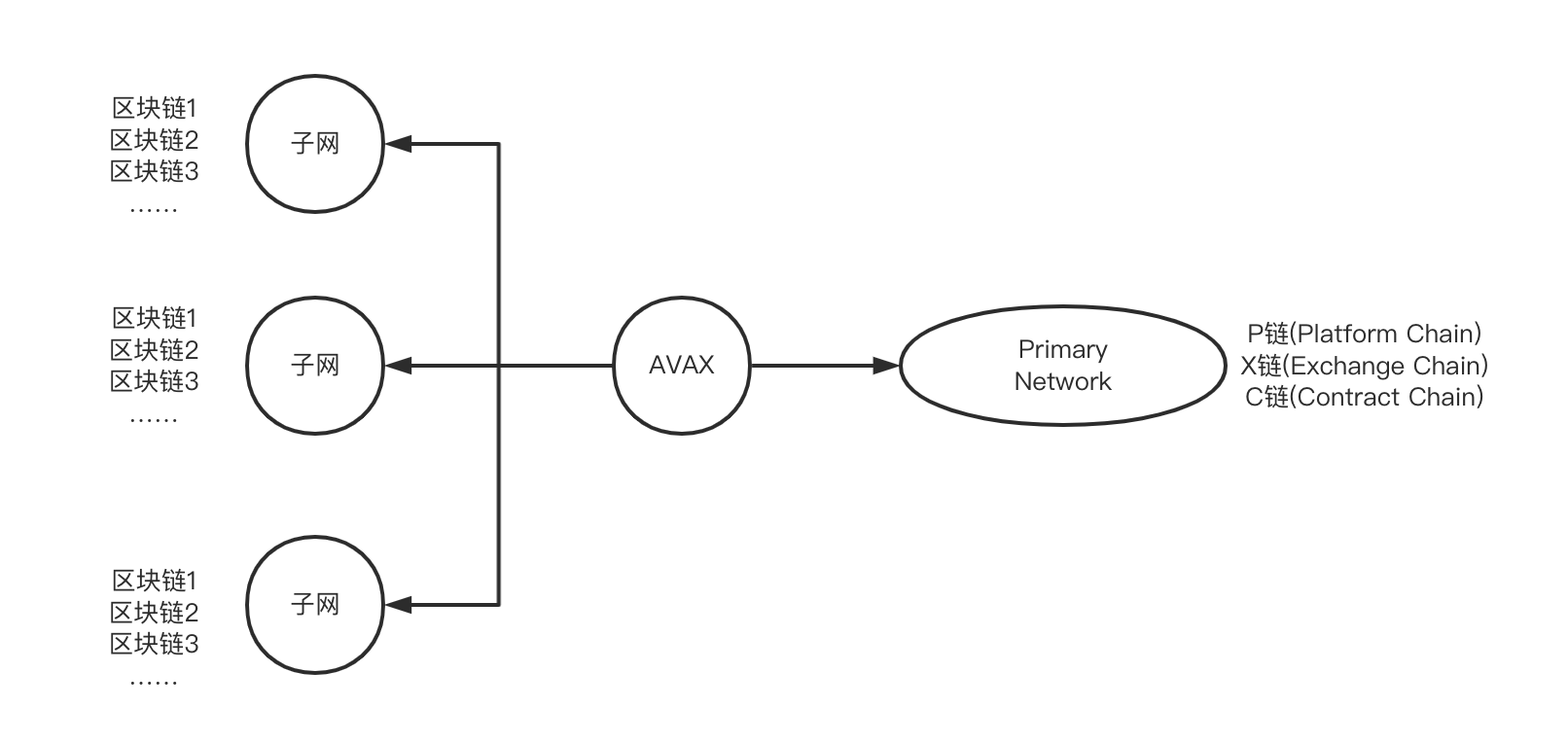


图2-3 Avalanche

**子网是一个验证小组：**子网中有很多个验证节点，这些节点的工作是验证该子网下的单个或多个区块链。子网会设置成员资格条件，只有满足条件的验证者才能加入该子网。比如：

1. 要求验证者必须是某个国家的公民；
2. 要求验证者完成实名认证；
3. 要求验证者的电脑满足某种配置要求。

除此以外，子网还具有以下三个特点：

1. 每个区块链只可以由一个子网验证，但一个子网可以验证多个区块链；
2. 验证者可以加入多个子网，但必须满足对应子网的成员资格条件；
3. 每个验证者必须参与验证 Primary Network。

Primary Network 相当于 Avalanche 的主网，负责验证 P链、X链和C链。这里也就明白了，为什么所有验证节点必须参与验证 Primary Network。因为节点越多越安全，Primary Network 负责的是整个 Avalanche 生态的安全。

P链、X链和C链的数据结构：P链和C链采用的是与传统区块链相同的线性结构，而X链的数据结构不是线性的，是一种叫做有向无环图的数据结构，简称DAG。

举个例子总结下Avalanche的架构，我们可以将 Avalanche 比作大学的一个专业，Primary Network 就是必修课，其他子网是选修课。所有同学相当于节点，同学们必须上必修课课程（维护网络安全），而选修课（相当于其他子网）可以由同学们自由选择。有些同学想毕业后出国进修，所以他们会选择英语课（相当于实现特定功能的子网）。为了更好的完成英语的学习，同学们会组织各种学习小组（相当于区块链），如听力小组、作文小组等。最后举一个形象的例子总结下Avalanche的架构。我们可以将 Avalanche 比作大学的一个专业，Primary Network 就是必修课，其他子网是选修课。所有同学相当于节点，同学们必须上必修课课程（维护网络安全），而选修课（相当于其他子网）可以由同学们自由选择。

2.3 跨链机制

不同区块链间无法相互通信，使人们无法享受区块链技 术的全部好处。跨链技术的出现，致力于通过实现区块链之 间互操作来解决这些问题，从而使不同区块链之间易于相互 通信并共享信息。目前主流的区块链跨链技术主要包括公 证人机制、中继机制和哈希锁定。

### 2.3.1 公证人机制

公证人机制是一种最简单实现跨链的方法。通过引入一个或多个可信实体作为“公证人”，完成数据收集和交易验证。在跨链过程中，可信实体向一条链声明另一条链上 发生了某事件，为跨链行为做背书。这些可信实体既可以不 断监听链上事件，达到条件后自动地做出响应；又可以在收 到请求时才对某些消息进行签名。根据公证人类型又可以分 为三类：单签名公证人机制、多签名公证人机制、分布式签名公证人机制。

单签名公证人机制，即中心化公证人机制，是最简单也 是最容易理解的一种模式，一般是由一个单独的中心化节点 充当公证人，这个单独的公证人负责所有跨链数据的收集验 证工作。常见的中心化交易所本质上就是这种模式，交易所 作为公证人，为不同用户之间交易做背书。这种模式虽然简 捷高效，但是存在中心化风险，于是就产生了多签名公证人机制。

多签名公证人机制，顾名思义，是由多个公证人组成的 公证人小组，组内每个人都有一个属于自己的私钥，跨链时 需要一定数量的签名达成共识，交易才能完成。这种方式与 单签名公证人机制比起来更加安全和稳定，但同时存在一定 局限性，需要跨链双方链上都要支持多重签名功能，有一定 的实现难度。

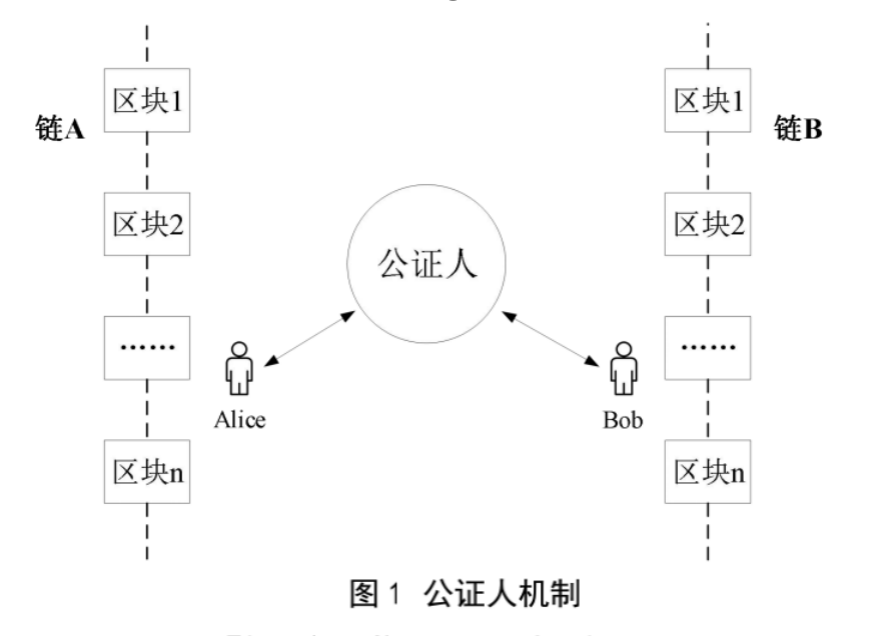
 分布式签名公证人机制实现起来相对比较复杂，该机制借鉴安全多方计算的思想，基本思路是整个系统只产生一个私钥，然后将私钥拆散成多个碎片，并进行加密处理后随机 分配给各个公证人，公证人之间互相不知道对方的密钥。跨 链时只有达到一定比例的公证人签名才能复原出完整私钥，验证完成跨链交易。这种方式更加安全，当少数公证人遭受攻击时，不会对系统稳定性造成影响。

图2-4证人机制

### 2.3.2 侧链/中继链机制

侧链是一条区块链（主链）将另一条区块链作为自身的 延伸（侧链），主链维护一个资产分类账本，并连接到侧链， 侧链是通过跨链通信协议连接到主链的独立系统。这里侧链 不一定是次要的，两条链可以成为彼此的侧链。最常见的 是主链与侧链之间资产转移：当用户想从主链向侧链转移资 产时，首先在主链上向一个特殊地址发送 X 个令牌，这些资 金即被锁定在主链上，同时相应数量通证被创建在侧链上， 于是完成跨链资产转移，用户就可以使用侧链上的令牌。如 果用户想换回主链令牌，侧链上的资产被锁定或销毁，同时 将主链上的令牌解锁。主侧链的连接验证方式一般通过简单 支付证明（Simplified Payment Verification，SPV）完成，由轻客户端完成，只需区块头数据即可验证区块链上的交易，而无需完整的区块数据。如BTC-Relayer 中的 Relayer，由美国区块链 ConsenSys 团队设计开发，被认为是区块链上 的第一个侧链，通过以太坊上的智能合约读取比特币链的区 块头并对其执行计算。Relayer 使用 SPV，仅使用区块头来验证支付，事务验证是通过对存储的 Merkle 树进行计算来实现的。

中继机制是侧链与公证人机制的一种融合。与公证人机 制相比，中继机制用中继链代替原先的公证人，接管公证人 的任务，是一种去中心化的公证机制。中继链是一条功能完 备的区块链，可以读取并验证连接到其上的区块链数据信息， 中继机制可以说是一种更加直接地实现跨链的方法。中继机 制可以在不依赖可信第三方的情况下完成交易验证，是一种 更加灵活、更易于扩展的跨链技术，实现形式多种多样，如 Cosmos 中的 Hub、Polkadot 中的中继链等都充当中继的角 色，一些侧链的实现也采用了中继模式。

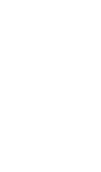
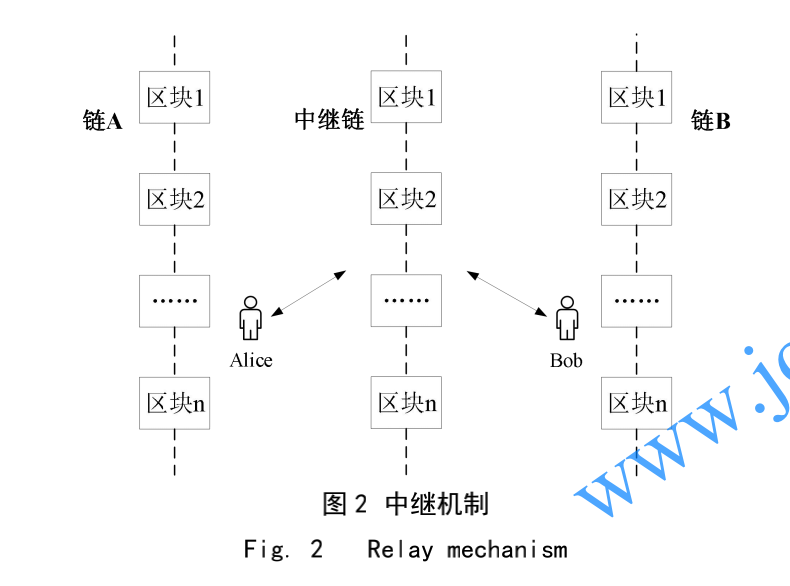


图2-5中继链机制

### 2.3.3 哈希锁定

哈希锁定最初是作为集中式交换的替代品出现的，支持 跨链原子操作，即允许一个用户向另一个用户发送一定数量 的加密货币，交换另一个区块链上持有的加密货币。哈希 锁定主要利用哈希锁和时间锁技术，不需要两条链之间过多 交互，不需要第三方公证人存在，通过交易者承诺在超时前 向另一方提供加密证明来进行交易。

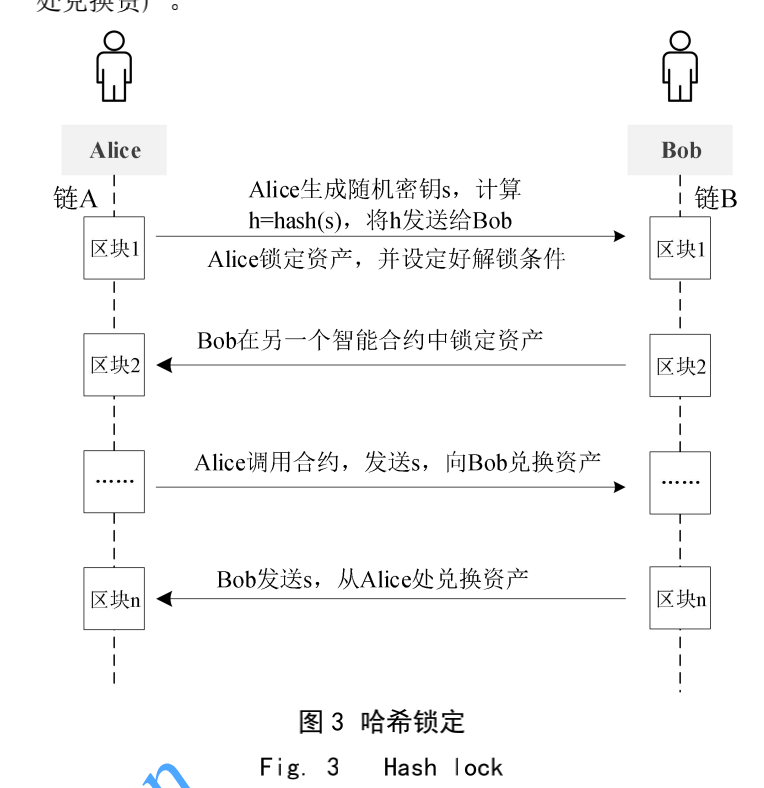


图2-6哈希锁定

假设 Alice 和 Bob 想要交换不同链上的资产，哈希锁定 流程机制如下：

（1）Alice 生成一个随机数 s，计算其哈希值 h=hash(s)， 然后将 h 发送给 Bob；

（2）Alice 将她的资产锁定在一个智能合约中，并设定 好解锁条件： Bob 在规定时间 T 内用猜出密钥 s，用 s 取走 智能合约中锁定的资金，否则资金将被退回 Alice；

（3）Bob 将其资产锁定在另一个智能合约中，解锁条件 为：在 t<T 时间内，Alice 可以输入密钥 s，取走锁定的资金， 如果 t>T，资金将被退回给 Bob；

（4）Alice 及时调用合约，提供正确的 s，向 Bob 兑换 资产；

（5）Bob 从 Alice 处获取 s 后，在剩余的时间内，从 Alice 处兑换资产。

这个过程中，Alice 和 Bob 取走资产的时间不确定，并 且在不同链上有所不同。这个过程可以说是原子级的，在 Alice 采取操作后，Bob 有足够的时间获取资金。否则 Alice 可能会拒绝输入密钥 s，所有资金都将被退回。然而在某些 情况下，比如汇率波动时，Alice 会占据有利地位，Alice可以在时间 t 内等待有利的汇率。为了缓解这种不平衡，经常 利用状态通道来加速交换，从而削弱发起方的优势。在跨链 操作的情况下，哈希锁只能支持原子交换，在资产可移植性 和数据跨链方面还存在困难。

跨链的实现是一个相对比较复杂的过程，其本质是将一条链上的消息安全可信地转移到另一条链上并在该链上达到 预期效果。公证人机制、侧链/中继机制、哈希锁定机制三种技术各有侧重，公证人机制跨链方式比较简单和单一，只支持资产交换；侧链/中继机制更加稳定和易于扩展，可以实 现资产转移，数据交换，或者其他更加复杂的需求；哈希锁 定机制只适合偏资产或者关键数据的交换，在支付领域应用 较多。在这些跨链技术的基础上，目前已涌现出非常多的跨链项目，致力于推动区块链网络的发展和完善，实现区块链互联互通。

### 2.3.4 跨链方案对比

对于上述三种不同的跨链方案，我们可以做出一个简单的总结对比。

中继链与公证人方案两种解决方案主要的重点在于如何实现一个消息转发机制，并针对消息转发机制进行优化以及安全性保证。公证人依托于可信机构来实现跨链消息的安全性保证，而中继链则依赖于多共识节点的投票一致性。这二者的效率各有千秋，其中，公证人的可扩展性强，但是安全风险更大。而中继链的可拓展性主要依赖于对于中继链模型的选型，但安全风险较小。

而对于哈希时间锁定而言，它的重点在于如何实现跨链资产兑换，其安全性由链上的智能合约来保证。它并不关注主要的消息转发机制，而是让用户自己在第三方平台上完成消息交换，可扩展性较差，也基本无法嵌入更复杂的功能。

因此，我们建议选择中继链来作为本项目的跨链方案选型。

2.4 链上数据监管

### 2.4.1 监管框架

#### 2.4.1.1 分级多层智能合约交易监管

分级多层智能服务交易监管架构采用了联盟链的架构，链上各个节点通常有与之相对应的行业，如物流、金融实体机构组织，参与者通过授权加入网络并组成利益相关联盟，共同维护区块链的运行。因此，该框架作为一种通用解决方案，不仅适用于物流领域，同样适用于普惠金融等其他行业领域。整个交易监管架构部署了两级监管点。其中，监管点 I 可以邀请监管机构加入到联盟链的部署维护中。

监管架构的中间层为联盟链中的实施管理和监管部分。该层的部署和实施单位为相应行业的实体组织结构，该层的主要任务是设计和实现服务交易及其监管平台。其中的监管模块是整个架构中的第二个监管点，以下称之为监管II。与监管I的作用不同，监管 II 更加侧重对用户的认证和管理以及编写智能合约实现对底层区块链数据的操作和分析。具体研究内容如下：

1) 节点注册进入系统时，通过合法匿名的方法实现对用户身份在联盟链中的认证。

2) 节点注册进入系统时，还需缴纳一定的保证金。

3) 异常数据检测与审计，即通过机器学习方法对读取的区块链数据进行审计，对于异常行为、异常得分的节点及其产生的异常数据，采取隔离审查的策略，审查期间该节点及其产生的数据进入黑名单。

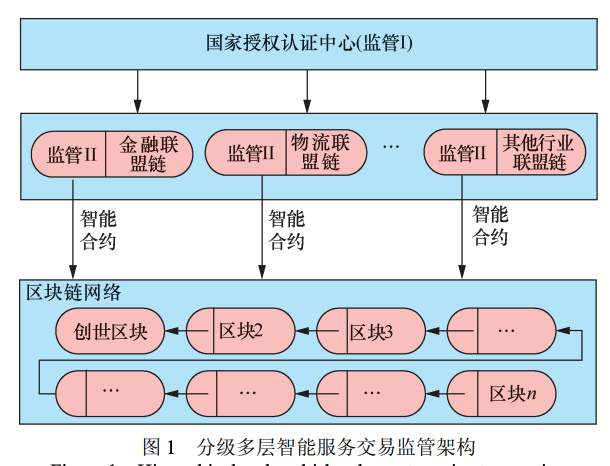
4) 惩罚措施，即通过审计后确定节点从事异常行为或某种网络击后，按照对行业或个人的影响程度，扣除其相应数额的保证金并列入其诚信记录。监管架构的最底层为所监管的交易数据，采用区块链结构对交易数据进行加密签名存储，交易数据的实际存储则通过 P2P网络协议实现。监管架构的中间层通过编写智能合约实现与区块链网络的交互，交互内容主要是读写操作。

图2-7 分级多层智能服务交易监管架构

#### 2.4.1.2 双链结构可监管模型

可监管模型的核心是由银行、金融机构和监管机构等参与组成的联盟链。联盟链中根据参与者协商好的对系统的控制比例分配给各个机构不同数量的系统节点，系统节点的多少决定了在共识过程中对系统的控制权的大小。

系统中包括3个模块:用户钱包、联盟链节点（含监管）和公有链节点。系统中节点间使用 P2P 通信，除了节点担任的任务不同，不存在服务器和客户端的定位区别。用户钱包是在系统中交易的发起者，帮助用户管理着所拥有的账户对应的公私钥和交易，方便用户查询和使用。联盟链完成交易接收、确认、混淆和打包记人区块的功能。公有链存储的转入、转出交易作为验证交易和用户钱包获得账户状态的凭据。

### 2.4.2 数据监管

#### 2.4.2.1 区块链数据类型

了解区块链的数据类型是进行区块链数据分析的前提，因此，本章主要介绍区块链数据的特点及其形式。区块链技术有 3 个不同的层次，但这 3 个层次并不是递进的关系，而是同时在发展着。目前，以比特币为代表的区块链1。0和以以太坊为代表的区块链2。0 广受市场关注，发展相对成熟。因此本节主要介绍目前这2个层次中最重要的数据类型：交易数据和合约数据。

区块链数据相当于一个账本，交易是改变账本状态的基本事件。交易通常发生在账户之间，在区块链系统中，为了匿名性，通常用地址代表账户，地址是由字母、数字组成的字符串。

尽管地址是区块链系统中通用的数据载体，但不同的区块链系统可能采用不同的账本组织方式。在比特币系统中，并没有通常意义下的“账户”概念，作为一种强化匿名的手段，一个用户可以拥有任意多个地址，一个交易也可能同时涉及多个地址。

智能合约是区块链2。0的核心要素。智能合约本质上是一段可以根据预先指定的条件被触发执行的代码。基于不同的区块链系统，智能合约可以有不同的实现方法。由于在目前的区块链技术中，以太坊是最流行的智能合约平台，因而以下主要介绍以太坊中智能合约数据。以太坊提供了一个图灵完备的虚拟机（EVM）用于实现智能合约。为了方便编写智能合约，以太坊提供了多种高级语言，其中Solidity应用最为广泛。Solidity是一门类似于JavaScript的面向以太坊虚拟机的智能合约编写语言。目前，在以太坊平台上，采用Solidity 编写的智能合约有超过 200 万个，而且每天都在不断地增加。

通常智能合约会涉及到 2 类数据：1）实现合约相关的代码数据；2)合约在运行过程中被触发的交易数据。由于触发的交易数据并没有记录在区块链上，因而，需要运行相关的合约才能得到相应的数据。一个替代的做法是通过网络爬取，目前，可以在etherscan。io上获取以太坊系统上智能合约相关的内部交易数据，但由于 API 的限制，只能获取每个账户最近 10 000笔的交易．

#### 2.4.2.2 交易分割检测

基于建立的实体网络，通过选择大于5000个币的交易构建了一个大额交易子网络。通过分析这个子网络，发现在早期的比特币交易中有许多特殊的交易模式，如分叉和自循环，即一个实体将一个地址上的比特币通过交易分割到不同的地址上，如此反复但最终所有的币又都汇集到该实体的某个地址上。如果说这种交易很有可能与早期的洗钱行为有关，那么另一种交易则似乎是刻意为之。在交易网络中存在二叉树形式的交易模式，即一个地址将其上的币等额地存人 2 个地址，这2个地址又将存入的币等额地分别存入2个新的地址，如此反复使得交易网络变成一个类似二叉树的结构。

#### 2.4.2.3 剥离链

剥离链主要出现在拥有大额比特币的地址上，在交易过程中该地址每次支付小额比特币给某个地址，看上去像每次从原地址中“剥”去小部分，剩下部分则转入一个只用2次的找零地址（一次用于接收比特币，另一次则用于全部转出），这个过程可能重复成百上千次，直至找零地址上余额很少。这样，通过许多只用2次的找零地址，这些交易形成一条“链”。具有这类模式的交易可能对应许多交易类型，如用户从钱包服务中取钱或矿池给参与者支付收益等。

# 三、隐私保护算法及密钥生成技术

3.1 隐私保护概况

在区块链系统的实际使用中，为了保证区块链上记录数据的可溯源、可验证等特性，所有数据都必须公开给区块链网络中的所有节点。这一特性在保障安全、可验证的同时，导致恶意攻击者可以直接获取区块链账本中记录的数据，并通过分析数据窥探用户隐私。基于隐私保护对象的不同，将保护对象分类为：组织机构、软件、区块链参与方、上链数据，其中，参与方和数据是区块链上记录的核心，因此本节将重点介绍关于参与方和数据这两部分的隐私保护技术。相似类型的技术，本文只挑其中具有代表性的进行介绍。

组织机构是指技术提供方机构与参与方机构。技术提供方机构，即本项目主体软件开发商，参与方机构是指附属软件提供方和客户。组织机构的安全管理是实现本项目隐私保护的前提，一般也是项目中最容易被攻破的一环 ，因此项目负责人需要整合现有资源，对本公司的安全管理进行查漏补缺，同时与合作方和参与方制定切实可行的行为规范和监督流程。组织机构面临的隐私泄露问题如下：门禁管理，外部人员自由进出，造成隐私泄露；人员管理，工作人员隐私意识淡薄，涉密资料容易外传；资料管理，涉密资料与机器未分级管理，非授权人员可获得。

隐私保护相关软件主要有两类，一类是技术提供方用于开发、维护、升级、管理的软件集合，另一类是参与方用于使用的软件。除却软件主体涉及的隐私保护外，受到隐私威胁的软件还包括依赖的三方件、公司自研软件集，开发工具及授权客户使用的软件等，需要对授权状态、版本管理和bug反馈等做统一管理。软件隐私保护通常关注如下部分：软件保密层级，非授权人员可获得机密技术细节；软件完整性与一致性，需要确保各方获得的软件完整且版本一致；软件可用性，软件需具备一定的吞吐量，避免高负载下隐私保护相关功能失效。

区块链的去中心化与存储机制使得隐私保护难度增加，一方面区块链网络中的节点往往是个人电脑，和专用服务其相比性能和安全性均较低，既影响区块链的可扩展性，又导致隐私保护难度上升，另一方面区块链中的所有交易均公开存放于区块中，可通过分析寻找其关联性，甚至暴漏真实身份。联盟链增加了准入机制，新增节点必须通过审核方可加入，但仍不能改变区块链本身较为脆弱的安全性。本章根据当前热点研究方向，从身份保护出发，并考虑链上数据保护，分别介绍了限制接入、节点隐藏以及零知识证明、机密交易相关技术。

3.2 隐私保护技术

3.2.1 环签名

群签名由Chaum和Heyst于1991年提出，群体中任意成员都可以以匿名的方式代表群体对信息签名，并且可以通过群公钥验证，群管理者可以管理群的公钥和私钥，利用群私钥对群用户生成的群签名进行追踪，并获得签署者身份。环签名可以看作简化的群签名，与群签名相比，其不再需要管理者，并且其他人仅可以验证环签名的有效性，而无法获知签名者身份。交易发起者使用环签名技术，把自己的公钥混合在多个公钥中，验证签名时无法确认签名者，达到隐藏交易发起者的身份。环签名具有以下三个性质：

1. **签名人混淆。**旁观者只能确定签名出自群体的某一成员
2. **可链接性。**同一私钥对不同消息签名，有办法判定两个签名出于同一私钥。其中根据两次环签名选择的群体是否相同，可以分为群体无关可链接性和群体相关可链接性。
3. **不可伪造性。**没有私钥不能伪造合法的环签名。

环签名根据使用的密码学技术有不同的实现，以下选取具有代表性的椭圆曲线环签名和RSA环签名进行介绍。

3.2.1.1 椭圆曲线环签名

1. 签名者生成密钥对。发布公钥，所有签名者的公钥组成集合。
2. 签名者生成n个随机数，对应n个公钥。签名者本身公钥对应的随机数在之后的计算中得到，n个随机数组成集合。
3. 签名者生成。定义，，，因此只需要得到即可求出。
4. 签名者生成随机数k。
5. 签名者计算得到。定义递推式，，接下来分别求。
6. 签名者组装环签名。。
7. 验证者验证签名。获得，获得消息m，根据递推式，从开始，依次求解，直到根据求出，判断求出来的是否等于获得的，即证。

3.2.1.2 RSA环签名

首先进行一些定义。为对称加密算法。公钥集合。加密操作：使用公钥对加密，。解密操作：使用对应私钥对解密，明文消息m。签名验签过程如下：

1. 生成环签名。假设签名者是。得到待签名信息：计算的对称密钥，随机取值v，随机取n-1个值，计算，，令，得到，签名消息，得到，，生成环签名。
2. 验证环签名。公钥验签，得到，使用公钥集，。得到对称密钥。验证等式成立。

环签名具有完全匿名性和不可追踪性，因此有着广泛的应用。如匿名投票，组织成员对投票信息进行环签名，并将签名信息和投票结果写到链上，其他人可在链上验证签名时，仅可获取发布投票到链上的机构，却无法获取投票者身份信息；又如匿名存证、征信，或匿名交易，当参与方具有较强的隐藏身份需求，又希望能发布消息时，环签名技术均可胜任。

门罗币引入了一种改进版的环签名，称为“多层可链接的自发匿名群签名”，它允许以合理的效率隐藏交易的金额、来源和目的地，并可验证、无信任地生成硬币。通过区块链系统中无关节点无法追查交易的发送方，当其它节点验证交易时，只能确定签名是诸多公钥中的一个，却无从定位到哪个公钥才是具体的发送方。

3.2.2 零知识证明

零知识证明用于向对方证明自己拥有某个信息或行为符合协议要求，同时不向对方泄露关于信息的任何内容。比较经典的零知识证明算法如zk-SNARKs，zk-STARK，Bulletproofs等，它们之间的区别和联系如下图所示：

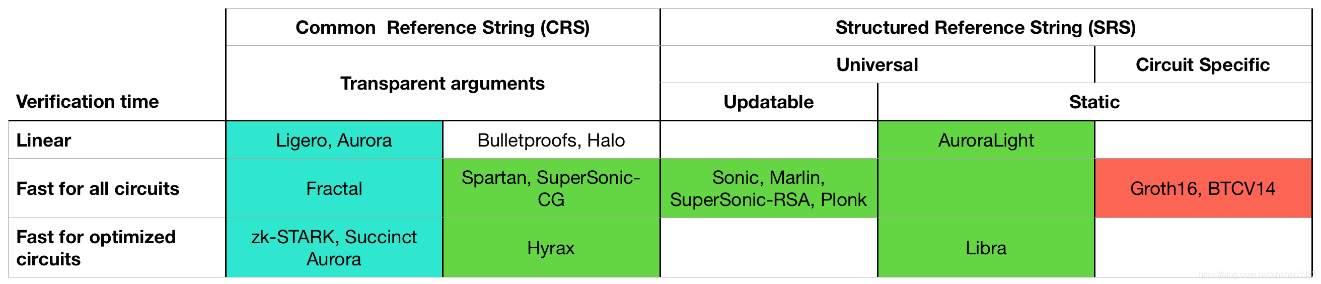


图3-1 零知识证明算法

3.2.2.1 Groth16

zk-SNARK全称zero-knowledge succinct non-interactive arguments of knowledge。即零知识证明、证据信息较短、几乎无交互、证明过程是计算完好的（证明者无法在合理时间内造出伪证）的零知识证明技术。Groth16由Groth于2016年提出，是目前最快、数据量最小的zk-SNARK，其设置需要绑定到一个特定的电路，由于其速度和证明的小数据量，因此常常被新的zk-SNARK拿来比较性能。

##### 1 背景知识

* NP问题。P，即多项式时间内可解的问题；NP，在多项式时间内可验证的问题，即不一定多项式可解，但多项式可验证；NPC，是NP问题，且所有的N

P问题都可以约化称NPC问题；NPH，所有的NP问题都可以称NPH问题，但NPH本身不一定是NP问题。他们之间的关系如图所示。PCP全称Probabilistically Checkable Proof，意思就是，所有的NP问题，都可以在多项式时间内通过概率验证的方法被证明。布尔可满足性问题是典型的NPC问题，所有算法可以编码为布尔型问题，这为Groth16证明的生成奠定了理论基础。

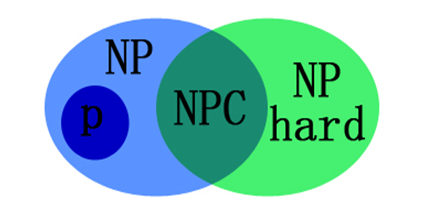


图3-2 NP相关问题分类

* 拉格朗日插值法。用于生成一个公式以拟合所有数据点，要求数据点坐标x轴均不相同，其生成流程如下：
  1. 首先定义一组坐标，，其中任意两个都不相同。
  2. 拉格朗日多项式为，其中称为拉格朗日基本多项式，，基本多项式的特点是在上取值1，在其他点取值0。
  3. 基本多项式构造思路如下：先找到在非点取值为0的多项式；该多项式在处取值为；该多项式赋值为除以得1；这个除法结果就是基本多项式。
* 椭圆曲线对。椭圆曲线加密算法基于对椭圆曲线的研究，在Groth16算法中setup环节生成参数时可选用椭圆曲线对以满足加密需求。椭圆曲线的定义为，其中{0}表示一个无穷大的点。
* 椭圆曲线群具有如下性质：群里的元素都在曲线上；单位元指无限远点；点P的逆元与P关于X轴对称；给定三个非零的点PQR，则P+Q+R=0（无限远点）成立。
* Merkle树。将拥有的信息进行摘要，与平级信息摘要放在一起继续进行摘要操作，如此形成的摘要集合称为Merkle树。账户0通过展示世界状态承诺，用于证明自己拥有账户0余额。
* 非交互协议。在计算过程中尽可能地减少信息交互，比较常见的有Fiat-Shamir Heuristic和公共参考字串。Fiat-Shamir Heuristic可以把任何交互式的随机验证协议转换为非交互式的协议，它把原本验证方生成的随机值使用方法H生成，然后在证明方完成所有计算，验证方拿到结果后重新算一遍，方法H可以使验证方相信可以替代自己的随机值，不过在交互过程中需要把自己计算中的所有参数都发给验证方，达不到零知识的要求。公共参考字串模型是指协议双方共享一段实现生成的字符串，该字符串生成方式对双方都保密。

##### 2 算法流程

Groth16流程由步骤Flatten、R1CS、QAP、LPCP、去除交互性和零知识检查组成：

1. Flatten 。将复杂的语句转成如下两种形式：x=y或x=y(op)z。
2. R1CS。即rank-1 constraint system。包含三个向量a、b、c和一个解向量，且满足s。a\*s。b-s。c=0。首先定义变量映射['~one'， 'x'， '~out'， 'sym\_1'， 'y'， 'sym\_2']，通过赋值给出结果变量，满足上述所有约束，得到[1， 3， 35， 9， 27， 30]，最后获得向量a、b、c。
3. QAP。采用拉格朗日插值法将R1CS最终转化成三个多项式fg=h，这样在一个点上验证，即相当于验证所有约束。
4. LPCP。了对生成证明的验证方法，流程如下
   1. 设范德蒙矩阵为V，多项式f在r点上取值。
   2. 拆分多项式。，可以通过验证方生成，g(r)同理，。至此，拆分成的形式，其中q可以在验证方计算。
   3. 验证协议。目的是验证，，流程如下：证明方Prover提供；验证方Verifier选择验证点r，计算出，因为上半部分是公开的，所以只需要另一半发给证明方即可；证明方Prover将分别与内积，获得a、b、c，其中内积的时候维度不够补零；验证方Verifier计算，自己计算公开的部分，再加上a，b，c 。
5. 去除交互性。使用CRS模型，首先设置Setup方，生成两个参数，Sv表示验证方要用的一半q，Sp表示证明方要用的一半q；然后添加线性加密系统Linear-only Encoding，用于将模型中的参考串进行同态加密，这里可使用椭圆曲线对；添加加密系统之后，q是密文，证明方可以伪造三个不同的，使得最终验证等式成立，而验证方无法察觉，因此需要加入随机线性检查，生成线性检查并要求证明方提供，验证方验证，以确保是一致的 。
6. 零知识检查。当前证明方暴漏给验证方的数据是a、b、c，在计算多项式的过程中，如果给定的一组取值点中再额外增加一个随机取值点的话，那么最后新得到的多项式就会拥有和这个随机数类似的一个随机分布，这样就无法从新得到的多项式中看到和原来的多项式相关的任何信息，达到零知识的要求。

Groth16在区块链上有很多应用，如Zcash，Filecoin、coda等。以Zcash[15]为例，利用零知识证明的技术使其交易达到了绝对匿名的效果，Zcash 的匿名性比门罗币（XMR）、达世币（DASH）更好，不过目前对于 Zcash 来说匿名交易只是可选项，比较尴尬的是目前为止大部分交易都是公开的，真正的匿名交易不到20%，并且很多钱包都不支持 Zcash 的匿名交易功能。Bellman是Zcash团队用Rust语言开发的一个zk-SNARK软件库，实现了Groth16算法。

3.2.2.2 累加器

累加器是一种单项成员散列函数，密码学累加器最早由Benaloh和Mare于1993年提出，当时是为了解决区块链上的数据可访问性问题。累加器通过设计函数，将数据隐藏在累加器值中，同时支持证明数据是否存在于累加器。在元素加入或删除时，根据commitment与membership proofs是否需要重新生成分为静态累加器和动态累加器。通常的实现方式有RSA（如BP97、CL02）、bilinear maps（DT08、CKS09）、Merkle hash trees（Mer88、CHKO08）等。以RSA累加器为例说明其构造过程，流程如下：

1. 累加器Accumulator创建。Setup环节选择质数g作为基底，再选两个大质数p，q，计算N=p\*q；加入元素，以添加a为例，计算；删除元素，删除a，计算。
2. 成员证明。假设当前，要证明存在与本累加器中，提供证明。
3. 成员验证。验证者得到，计算是否成立
4. 聚合证明。提供用于验证
5. 非成员证明。假设集合有三个元素，要证明不在集合内，相当于证明不是的素因子，即互质，只需找到一组(a，b)，使得，生成证明。在验证时，需要验证等式。

3.2.3 Merkle树

3.2.3.1 Merkle树简介

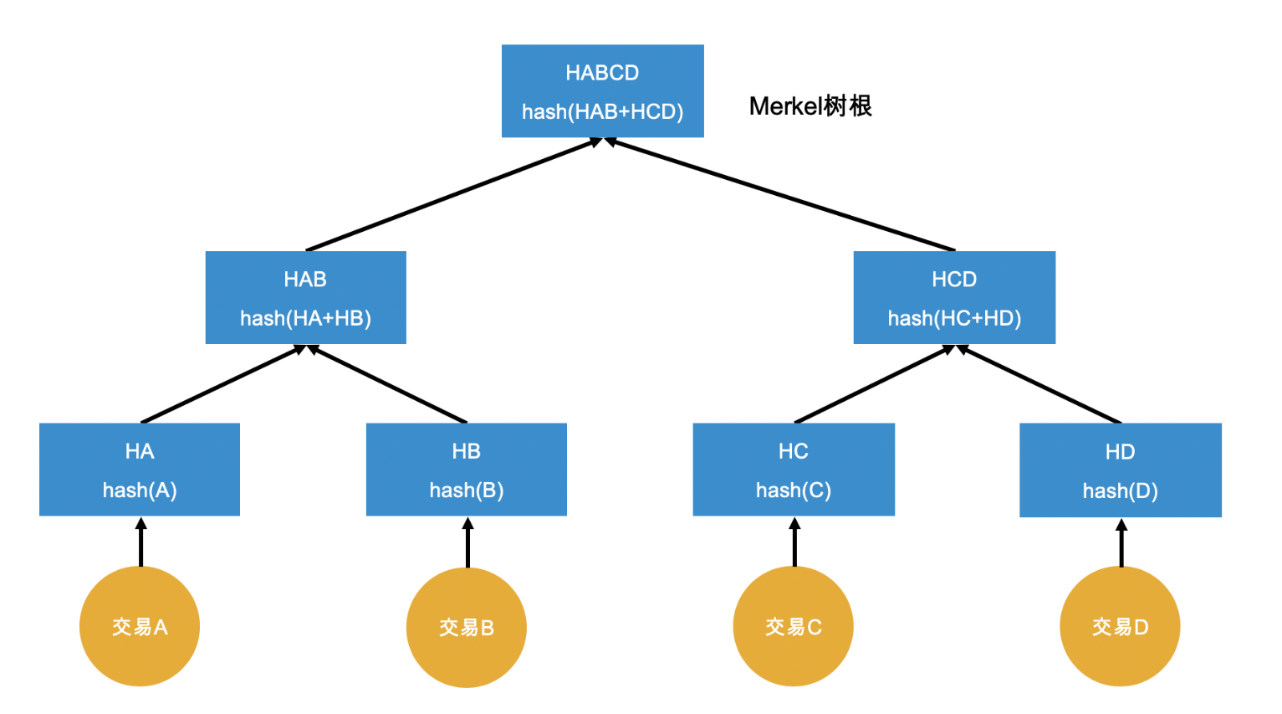


图3-3 Merkle树

默克尔树，也叫哈希树，是由美国计算机科学家拉尔夫·默克尔（Ralph Merkle）于1979 年提出的，它本质上是一种树型数据结构。与标准二叉树一样，Merkle树由叶节点，中间节点和一个根节点构成。叶节点包含存储数据或其哈希值，中间节点是其两个子节点内容的哈希值，最上层的根节点同样也是由它的两个子节点内容的哈希值组成。

在上图中，假设该区块中存在交易A，交易B，交易C 和交易D 这4 笔交易信息。那么在默克尔树中，首先对每个交易信息进行哈希计算分别得到HA，HB，HC，HD 这4 个哈希值。其次将每个交易信息对应的哈希值生成叶子节点，然后对于相邻两个叶子节点的哈希再进行一次哈希计算生成中间节点。最后再对相邻的中间节点进行哈希运算得到的哈希值生成根节点。根节点也就是默克尔树根，如此，Merkle树根的值与所有交易相关，任何一笔交易发生变化，Merkle树所发生的变化都是不可预知的。通过Merkle树根值的变化情况，结合中间节点和叶子节点的变化路径，可以验证任意一笔发生变化的交易，从而为记录的交易提供了不可篡改性，同时，还能提供某笔交易的存在性证据。

3.2.3.2 Merkle树应用

Merkle树的应用场景有以下几种：

1. 快速比较大量数据：当两个Merkle树的根哈希值相同时，说明所代表的的数据都相同
2. 快速定位修改：如上图，如果交易C发生改变，那么就会导致HC、HCD和Merkle Root发生改变。所以，我们想要快速定位，只需要沿着Root==>HCD==>HC就可以定位到交易C发生改变。
3. 零知识证明：例如，想要证明一组交易中包含某个交易A，但又不想让对方知道交易A的具体内容，那么就可以构建Merkle树（如上图），向对方公布HA、HB、HCD和Root，对方就可以确认交易A的存在，但无法知道交易A的具体内容。

3.2.3.3 多层哈希鉴证

Merkle树可以实现链上的多层哈希鉴证，具体而言，就是利用Merkle树的数据结构以及哈希函数的单向性，验证树内某个交易支付是否存在。当然，对于存储数据的区块链，我们需要验证的就是数据存储的正确性与完整性。简单支付认证（即Simple Payment Verification，简称SPV）就应用到了多层哈希鉴证。SPV最早在比特币中提出，目的是为了验证某个交易支付是否存在，以及得到比特币网络多少个确认（多少个区块）。

在SPV节点上，不保存全部区块链数据，不下载区块全部交易，只保存区块头数据。所以这种节点不能验证全部交易，只能用于验证支付（确认支付是在区块链中，以及确认多少次）。那么按照这种存储模式，这对整个链上存储容量的要求就大大减少，可以充分利用区块链的验证特性，提高溯源效率。

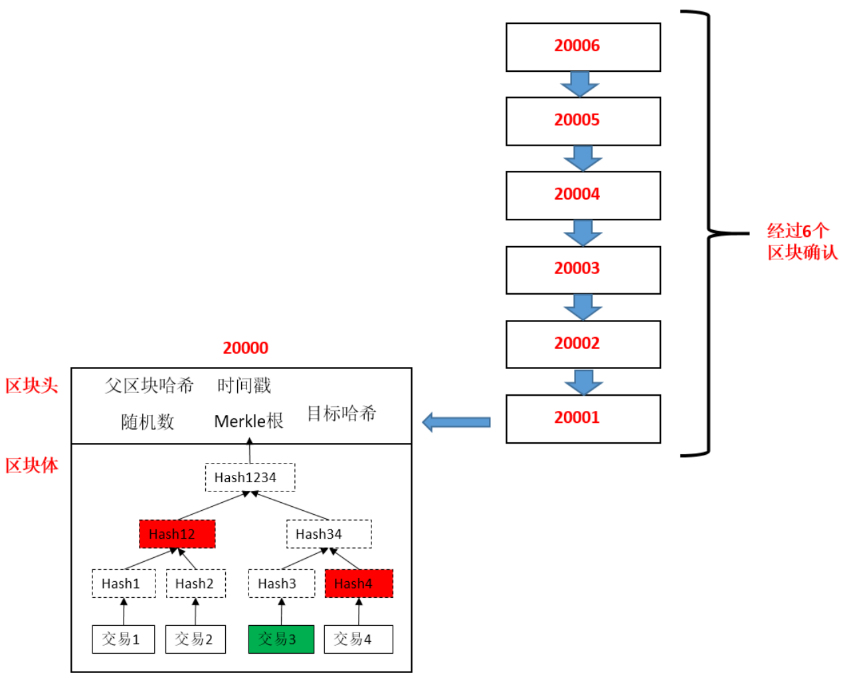


图3-4 SPV验证

例如，货主A在支付运费时通过区块链平台支付，并声称自己已经转账了1000$给承运商托管账户，承运商需验证支付有效（SPV验证），这整个过程如下图所示：

1. 接收含有托管账户地址的交易。SPV节点只关注某个支付到自身账户地址的交易，则可以通过建立布隆过滤器（一种基于哈希的高效查找结构，能够快速确定某个元素是否在一个集合内）限制只接受含有目标地址的交易。
2. 获取相关交易的Merkle根及存在性证明（Merkle路径）。一旦区块链网络中其他节点探测到某个交易符合SPV节点设置的布隆过滤器条件时，其它节点将以Merkleblock消息的形式发送该区块，Merkleblock消息包含区块头和一条连接目标交易与Merkle根的Merkle路径。
3. 验证交易，检查交易存在性和交易是否双花。SPV节点通过该Merkle路径找到跟该交易相关的区块，并验证对应区块中是否存在目标交易（该过程也被称为：MerklePath Proof）。若SPV确认了交易确实存在于链中，但此时依然无法保证这笔交易没有被双花。这时候SPV节点通过去看这笔交易所在区块之后的区块个数，区块个数越多，说明该区块被全网更多节点共识，一般来说，一笔交易所属区块之后的区块个数达到6个及以上时，我们认为该交易已经无法篡改。

### 3.2.4 隐私交易通道保护

Fabric中的隐私交易通道保护技术，主要可分为通道隔离和隐私保护。通道隔离定义在通道上，是更加粗粒度的隐私交易保护手段。它可以将Fabric网络中不同通道中的账本数据进行隔离和保密，主要用于实现联盟链中业务的隔离，每个通道可视为一条子链，并且对应一套账本。同一个区块链网络，不同通道之间的账本数据不可见。隐私保护定义在通道内部，是一种细粒度更高的隐私保护方法。在同一通道内的不同成员，例如A、B成员，经过隐私保护配置，指定A成员可以访问相关数据的权限，只有拥有指定权限的成员A才能访问相关数据，没有权限的成员B则无法访问相关数据，保证在一个通道内数据的隐私性。

Fabric1.0中采用通道隔离，使用多通道特性，账本网络中若干成员可以协构建一个专属通道与外部隔离，通道外的节点无法看到其中的账本和交易数据，提高了交易中的数据隐私性。由于统一通道内节点拥有相同的账本且交易及账本上的读写集都是公开的，所以想实现通道内部敏感数据的隔离，只能在此通道外建立新的通道。对于一个复杂的业务场景，有时需要建立上百条通道来作为数据隔离方法，但随着通道数量的上升，会严重影响Fabric的性能。因此，从一开始就按照业务需求的不同对通道中所保存的数据进行合理归类是必要的。

3.2.5 其他隐私保护技术

3.2.5.1 机密交易

2013 年，Adam提出引入同态加密技术隐藏交易金额的思路，在不泄漏交易输入输出金额的具体数值的前提下，保障交易的输入输出资产金额相等。利用范围证明技术保障资产金额的数值在合法范围内，避免负数或越界的情况。2015 年，Maxwell对该想法进行了完善及实现，通过引入盲化因子避免了暴力枚举攻击，提出了机密交易技术(confidential transactions)。

机密交易技术中采用Pedersen 承诺，在不透露交易金额的情况下，使得其他节点能验证交易输入输出金额之和相等的正确性，利用椭圆曲线密码体系(elliptic curve cryptography，简称ECC)隐藏金额，并添加盲化因子保证在输入金额与输出金额相等的情况下，发起者才能对交易进行正确签名；同时，盲化因子也能防止攻击者通过枚举暴力破解金额。

流程如下：

1. 使用同态加法隐藏金额。，G，H为椭圆曲线生成元，x为随机数，a为金额。
2. 构造。
3. 对承诺。私钥签名，公钥验证，令，消除利用得到的可能。对总数额签名，当签名验证通过，可以证明总数额没有变化，如果验证不通过，因为无法得到，发送方无法构造可验证的签名。
4. 构造中使用的通过秘密传递给接收人，便于接收人将其花出去。

3.2.5.2 一次性地址

在大多数采用地址作为标识符的密码货币项目中，地址都是由公钥压缩编码得出，虽然无法通过地址反推出用户信息，但是仍然可以通过追踪交易记录和利用社工手段猜测到用户身份。门罗币提出了一种一次性地址方案，在每次交易时，发送方生成临时公私钥对，其中临时公钥作为接收方地址，而接收方可以通过对应的临时私钥消费，具体过程如下：

1. B维持一公私钥对(k，K)。K=kG，G为椭圆曲线的基点(生成元)。
2. A希望发送一笔交易给B。A构造一对公私钥，，r是A生成的随机数；公钥由A生成，作为B的临时地址，私钥由A的r和B的k组合而来，A将r发送给B，使得B可以获得私钥。
3. A设法秘密传递r。利用DH算法共享，A发送给B，此时，B计算得到，再计算。
4. A公开发送信息。发送，使用Hash函数将椭圆曲线上的二维点映射成标量；发送
5. B验证一次性地址。计算，与对比，若一致，说明是自己的一次性地址，恢复一次性地址对应的私钥。
6. 代理验证。生成两对密钥，将view密钥交由代理人，每当出现一次UTXO，由代理人验证，通过后再由自己生成一次性私钥；A发送和；B验证恢复和。

3.2.5.3 限制接入及恶意节点检测

限制接入是目前热门的联盟链隐私保护技术之一，即对区块链中的节点进行验证授权，通过严格的验证机制大大降低恶意参与方出现的可能性，常常用在私有链或联盟链中。在现有的比特币系统中，为了防止恶意节点进行拒绝服务攻击，节点设置了黑名单机制，具体而言：当从某一节点接受到不合法消息后，会降低该节点信誉度，降低到0后加入黑名单。然而，目前对于恶意节点自动检测相关的技术并不多，在现有的区块链平台上，大多是人工设置策略进行管理，下面提出一种相对可行的恶意节点检测技术。

私有链和联盟链虽然本身具有认证机制，但链内节点仍可能存在恶意行为。下面提出的自动分类节点的行为模式，根据序列数据表示节点行为，定义相似性度量并通过相关聚类算法对行为模式分类，先挑选出具有代表性的各行为模式作为行为模板，然后找到不遵守任何行为模板的节点。其中对度量序列相似性和聚类方法的选择需要根据具体应用场景进一步确定。对于度量序列相似性，目前有几种比较经典的算法，其特性分别如下：欧几里得距离算法（Euclidean distance），要求两个序列具有相同长度，使用L2范数（norm）计算距离；动态时间扭曲算法（Dynamic Time Warping/DTW），不要求长度一致，并且可以处理时移（Time Shifting）问题，这也是本技术最终选用的方法；实序列距离编辑算法（Edit Distance on Real sequence/EDR），利用了杠杆差（Leverages Gap）和不匹配惩罚，并且可以处理噪音和时移；最长公共子序列算法（Longest Common SubSequences/LCSS），对于序列噪音引入了门限，用于发现两个序列中的最长公共子序列，具体算法如下：

* + 1. **欧几里得距离。**假设s和x为二维向量，则s和x之间的欧几里得距离为
    2. **DTW距离。**序列x和s之间的DTW距离为
    3. **EDR 距离。**序列x和s之间的EDR距离为
    4. **LESS相似度。**序列x和s之间的LESS相似度为

随着机器学习的日益发展，涌现了许多优秀的聚类算法，根据类别介绍如下：基于分区的聚类算法，即将未标记数据分成n份，每份决出中心，不过对于中心的定义根据算法会有所不同，例如k-means以平均值为数据中心，而k-medoids以集中点为中心；基于密度的聚类算法，从一个元组开始，根据设定的密度阈值吸收周围元组，其中比较著名的如DBSCAN，其核心要点是先找到密度较高的点，然后把相近的高密度点逐步相连。《DBSCAN Revisited: Mis-Claim， Un-Fixability， and Approximation》文中对其进行了改进，分析了DBSCAN在高维空间的限制并给出了解决方案；基于分层的聚类算法，对数据集进行层次分解，根据分解策略又可以分为凝聚分层聚类（自上而下）和分裂分层（自下而上）聚类，凝聚分层是指从各样本点开始，根据设定规则合并直到达到设定条件，分裂分层是指在初始状态下合并所有样本点，根据设定准则逐步分裂直到达到要求，分层聚类算法比较著名的有Chameleon，BIRCH等；基于网格的聚类算法，将数据集网格化，调整分辨率并识别高密度网格单元，能够获得密度聚类的优点并减少一定的计算复杂度，典型算法如STING，CLIQUE等。

3.2.6 隐私保护技术使用与对比

不同隐私保护技术的侧重点不同，整体而言，隐私保护技术可以分成身份保护和数据保护两个方向。

在身份保护方向上，比较常见的技术如一次性地址、限制接入与恶意节点检测、环签名等。限制接入与节点检测技术处于保护链条的最前端，通过较为粗放的技术过滤大部分潜在攻击者。一次性地址和环签名均可用于隐藏参与者身份，而其侧重点则各有不同，其中环签名的目的是将发送者身份混杂在人群之中，而一次性地址隐藏力度更强，可以完全隐藏接收者的身份。联盟链中参与节点均有所属组织，可以使用环签名对参与者所属组织认证的同时，模糊具体参与者以防止定位。而对于需要对接收方身份保密的场合，可以使用一次性地址，避免通过公钥分析定位接收者身份信息。

在数据保护方向上，除了常见的数据加密手段外，本章还推荐了两个已有成功应用案例的密码学技术：零知识证明和机密交易。二者均可用于隐藏关键信息，而其应用侧重点不同，机密交易使用同态加密技术，其目的是在不泄露参数的前提下进行相关计算，可用于隐藏交易金额。零知识证明相比于机密交易，运算量更高但使用范围更广，常用于证明交易符合规则。

3.3 密钥生成

密钥生成根据流程，一般可分为密钥提取与密钥协商两个阶段，其中在密钥提取阶段，用户可根据信道及其他随机因素提取私有随机信息，在密钥协商阶段，可以通过设立可信机构与客户交互进行密钥生成，也可以通过各种密钥协商算法讨论得出。基于区块链本身的去中心化特性，可以考虑通过密钥协商协议完成密钥生成。分布式的密钥协商，常见的如BD（burmester-desmedt）协议、STR（steer，et a1）协议、TGDH（tree-based group diffie-Hellman）协议等，其目的在于通过去中心化的信息交流，在各参与方本地计算得出统一的密钥值，其研究方向则在于如何在效率、信息大小和安全性之间取得平衡，并设法抵御恶意参与者的破坏。本节以BD-GKA协议为例，介绍密钥协商的常规步骤：

1. 各参与者随机选取，广播。
2. 各参与者收到其他参与者的广播后，计算并广播。
3. 各参与方计算密钥。，其中，各参与者绕成一个圈。表示i向前移动一位；表示绕了一圈，一直移动到初始值的前一位，由于参与者围成一个圈，，最终

对于BD-GKA，如果缺少任一参与方，计算链都会中断，所以鲁棒性低。以Stanisław Jarecki等人于2011年发表的《Flexible Robust Group Key Agreement》为例，介绍如何对密钥协商算法做出安全性和效率等方面的改进。关于安全性的改进如下：

1. 各参与者随机选取，广播。
2. 完成上一轮的参与者组成新的参与者列表AL。每个参与者广播，。
3. 完成上一轮的参与者组成新的参与者列表AL。每个参与者计算密钥，，最终。

在这次改进中，在第二轮信息交换中，通过设立冗余广播信息，极大增加了对相关节点故障的容错性。对于效率的改进如下：

1. 各参与者随机选取，广播，
2. 完成上一轮的参与者组成新的参与者列表AL。每个参与者广播，。
3. 完成上一轮的参与者组成新的参与者列表AL。每个参与者计算密钥，，此处与的相等，所以最终结果也相等，最终。

通过将参与节点拆分成两份的形式将信息交换大小降低至。

四、身份及权限管理

4.1身份管理

从过去单一的去中心化公有链，区块链发展到今天开放性的公有链、具有准入机制的私有链和联盟链并存的多种技术形态，并在行业应用中得到了普及和推广。区块链技术对当今互联网金融及物流、能源、农业等传统领域造成颠覆式的影响。

区块链系统中的身份管理是区块链应用中实体之间进行通信交互和资产交易的基础，它包含对实体身份从产生、存储、认证、使用、审计到注销的全生命周期的管理。身份管理一直是区块链系统中的关键组件，也是构成去区块链系统安全的基础。区块链身份管理主要包括2方面的内容：身份标识和身份认证。其中，身份标识是构成链上资产权属标记和支付的确权基础，而身份认证则构成了区块链共识及交易的信任基础。

4.1.1 身份标识

在用户身份管理方面，目前的区块链系统大多基于非对称密码算法来实现用户的身份标识。但公有链和联盟链在具体实现方案上存在较大的差异。公有链中的比特币、以太坊、门罗币、零币等强调用户的隐私保护，采用匿名身份认证机制，用户公钥作为身份标识符，用户自主生成和管理标识身份的密钥；以Fabric和Corda等为代表的联盟链则更关注强监管环境下用户身份识别的需求，采取实名或可控匿名的身份认证机制，使用数字证书作为身份标识符，采用CA（certificates authority）来生成和管理数字证书。

4.1.1.1 公钥标识身份

目前区块链系统大多基于公钥密码体系的椭圆曲线密码算法来构建用户的身份标识。以比特币为例，比特币系统通过调用操作系统底层的随机数生成器生成256位随机数作为私钥。比特币私钥总量大，极难通过遍历全部私钥空间来获得存有比特币的私钥，因而从密码学计算复杂性的角度来说是安全的。比特币系统通过非对称密码生成私钥和地址的过程如图4-1所示。

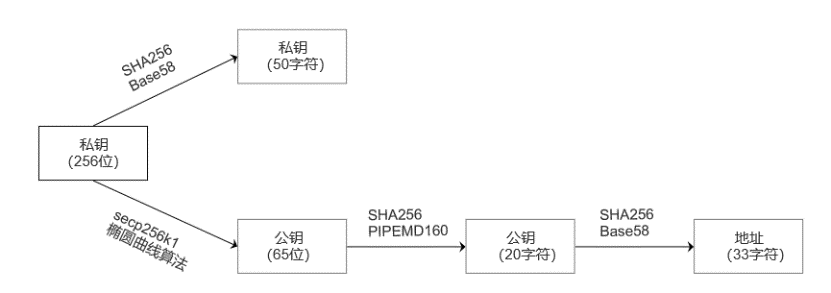


图4-1 比特币系统通过非对称密码生成私钥和地址

为便于识别，256位二进制形式的比特币私钥通过SHA-256哈希算法和Base58转换，形成50个字符长且易识别和书写的私钥提供给用户。比特币的公钥是由256位随机数私钥经过secp256k1椭圆曲线算法生成65字符长度的随机数。该公钥可用于产生比特币时使用的地址，其生成过程是首先将该公钥进行SHA-256和PIPEMD160双哈希运算生成20字符长度的摘要结果，再经过SHA-256哈希算法和Base58转换形成33字符长度的比特币地址。公钥生成过程是不可逆的，即通过公钥不能反推出私钥。比特币系统中用户的公钥和私钥通常保存在比特币钱包文件中，其中私钥最为重要，与传统中心化货币系统验证源丢失后可找回不同，比特币系统中私钥丢失就意味着丢失了对应地址的全部比特币资产。

以太坊系统生成私钥和地址的过程与比特币类似，不同之处在于以太坊在生成公钥后不再经过SHA256和PIPEMD160双哈希运算生成哈希摘要，而只经过一次SHA256后选取后40个字符作为以太坊地址。

4.1.1.2 数字证书标识身份

以Fabric和Corda等为代表的联盟链更关注强监管环境下的用户身份识别，均提供了基于数字证书的用户身份标识，以实现实名认证。例如在Linux基金会所主导的开源联盟链平台Hyperledger Fabric中，Fabric的成员服务(membership services，简称MSP)为区块链网络提供用户注册、身份管理和权限管理等服务。Fabric中默认的MSP使用X.509证书作为身份标识符，采用传统的公钥基础设施(public key infrastructure，简称PKI)分层模型，并支持RSA和ECDSA公钥密码算法。PKI通过引入CA机构为注册真实身份的用户颁发数字证书，数字证书包含用户身份信息、用户的公钥信息、CA数字签名和有效期等信息。数字证书作为身份标识符与用户具有一一映射关系。在区块链交易过程中，数字证书可代替公钥来作为数字资产的权属标记，用户使用对应的私钥签名来完成支付行为的确认。此外，CA负责数字证书的全生命周期管理，包含数字证书的签发和更新，数字证书的撤销、查询或下载等。如图4-2所示，通过数字证书可实现身份认证、交易的发布和确认。

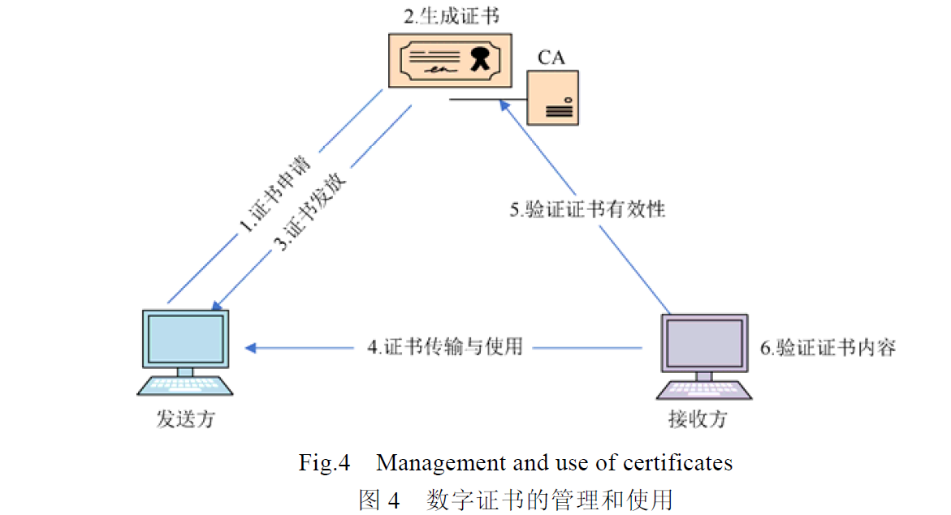


图4-2 数字证书的管理和使用

4.1.2身份认证

4.1.2.1 匿名认证

公有链中的比特币、以太坊、门罗币和零币等强调用户的身份隐私保护，为交易用户提供了匿名认证，即用户在注册过程中无需出示真实身份即可获得身份标识并将其用于身份认证，由于系统采用无准入机制的开放网络架构，系统中无类似CA的可信认证中心来管理身份，用户基于非对称密码算法自主生成和管理身份。此外，为了实现交易过程中的身份隐私保护，同一用户还可以使用多个不同的身份标识符来降低交易地址与真实身份以及不同交易之间的关联性。虽然匿名认证能够在一定程度上保护用户隐私，但值得注意的是，由于区块链交易系统具有张本公开、多方确认的特点，简单使用匿名认证并不能完全有效地解决交易身份隐私保护的问题。

4.1.2.2 实名认证

为了实现用户的准入控制并符合监管要求，Fabric、Corda、趣链等联盟链更关注强监管环境下用户身份识别的需求，在身份管理中采用基于数字证书的实名认证方案。通过在系统中部署PKI/CA来管理身份，用户基于非对称密码算法通过CA生成和管理数字身份，用户实名认证获得数字证书身份标识的过程如下:

1. 用户发送实名注册信息给CA申请数字证书；
2. CA核实用户实名信息，如果有误，则中止申请过程；
3. CA基于用户实名信息为用户生成公私钥对并签发实名数字证书，确保数字证书与用户身份的一一绑定；
4. CA将生成的数字证书和私钥发送给用户；
5. 在区块链交易过程中用户使用数字证书作为身份标识符，通过私钥签名实现身份认证，身份管理的全周期用户都是实名的。

基于数字证书的实名认证方案可以较好地适用于中心化系统中账本保密的应用场景。但是对于区块链中的公开账本系统，却带来了极大的隐私泄露风险。观察者可通过对公开账本中实名身份的分析实现交易的关联和追踪，从而严重威胁到用户的交易安全。因此，实名认证方案不适合应用在公有链系统中，但对于Fabric，Corda等联盟链，其本身利用通道隔离、隐私数据加密机制等进行隐私保护，可一定程度上保障隐私安全。

4.2权限管理框架

4.2.1 Fabric权限管理

权限管理是区块链技术的关键部分，尤其是对数据访问有更多要求的许可链。Fabric是一个认证性的网络，所以区块链参与者需要一种向网络中其他参与者证实自己身份的机制从而在网络中进行交易。Fabric中的不同参与者，包括Peer节点、排序节点、客户端等，每一个参与者都具有封装在X.509数字证书中的数字身份，这些数字身份确定了它们对资源的确切权限。同时，要使身份可以被验证，它必须来自可信任的权威机构。成员服务提供者（MSP）是Fabric中可以信任的权威机构，一个MSP是定义管理组织有效身份的组件。总的来说，Fabric中默认MSP使用X.509证书来作为身份，并采用传统的公钥基础设施（PKI）分层模型进行数字证书的分发管理。

如图4-3所示，Fabric对联盟链进行模块化分层管理，分别包括网络层，交易层，业务层，其中交易层进一步模块化为共识机制模块和权限管理模块。在权限管理模块中，MSP从节点、组织、联盟等多层次对联盟链进行权限管理。

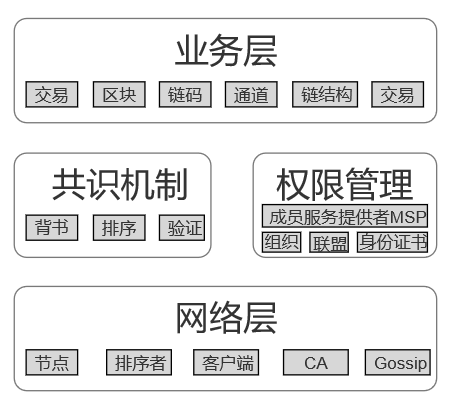


图4-3 Fabric逻辑模块

4.2.1.1 MSP作用

Fabric提出了MSP的概念，抽象代表了一个身份验证的实体，基于它可以实现对不同资源进行基于身份证书的权限验证。MSP的具体作用如下：

1. 权限管理：MSP分层管理网络、通道、结点的权限。
2. 身份验证：每一个想加入网络的节点必须提供其有效且合法的MSP信息。
3. 交易验证：在各节点之间进行数据传输时，需要验证各节点的合法MSP签名。

从以上作用可知，MSP将颁发与校验证书，以及用户认证背后的所有密码学机制与协议抽象了出来。MSP管理用户id，并对网络中的所有参与者进行身份验证。一个Fabric区块链网络可以由一个或多个MSP控制。这提供了成员操作的模块化，以及跨不同成员标准和体系结构的互操作性。

4.2.1.2 MSP结构

##### 1 Fabric CA

Fabric-CA是一个Fabric中内置的CA组件，它允许用户在区块链网络中创建CA，同时能够管理具有X.509证书形式的Fabric参与者的数字身份。图4-4说明了Fabric-CA服务器如何应用于整个Fabric架构。

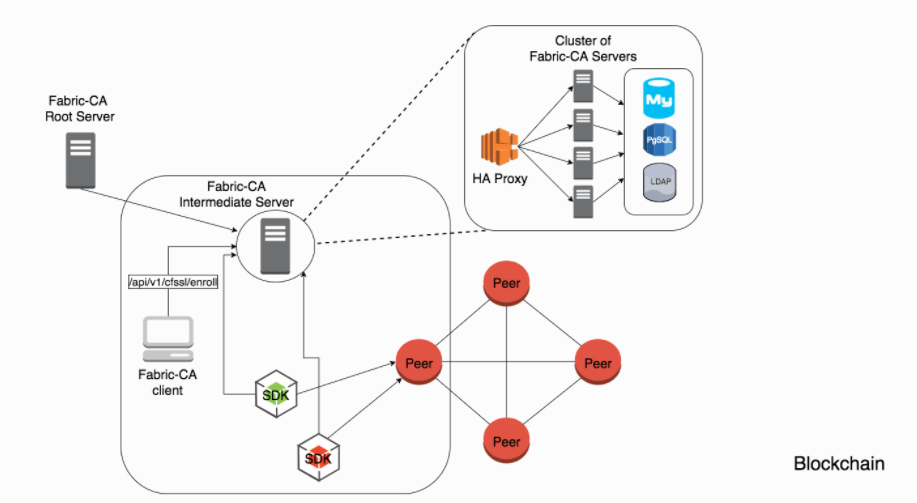


图4-4 Fabric CA

通过Fabric-CA Client或Fabric-SDK两种方式与Fabric-CA-Server进行交互。同样，在大规模网络场景下，也可以直接与Cluster of Fabric-CA-Server进行交互，以便缓解网络压力，更高效地进行身份管理。但无论是通过以上哪种方式获取数字证书，CA服务提供者必须包含一个联盟链网络可信的root-Ca或者其他包含可信root-CA的Intermediate-CA。

##### **2 MSP逻辑结构**

MSP可分为以下几类：

* 网络MSP。负责联盟链网络的准入机制。
* 通道MSP。负责通道的配置，包括通道内的组织，通道管理员列表等。
* Peer MSP。负责交易背书。
* Orderer MSP。负责联盟链内网络的交易排序。

通常情况下，一个组织或联盟可以对应到一个层级化的MSP。所谓层级化的MSP，是指由根证书，生成中间件证书等组成的层级化的结构，从而实现不同管理员和普通成员之间的权限控制。一个资源实体的MSP中往往包括签名和验证算法，以及一组符合X.509格式的证书，这些证书都需要追溯到同一个信任的根。图4-5是一个MSP结构中常见的组成部分：

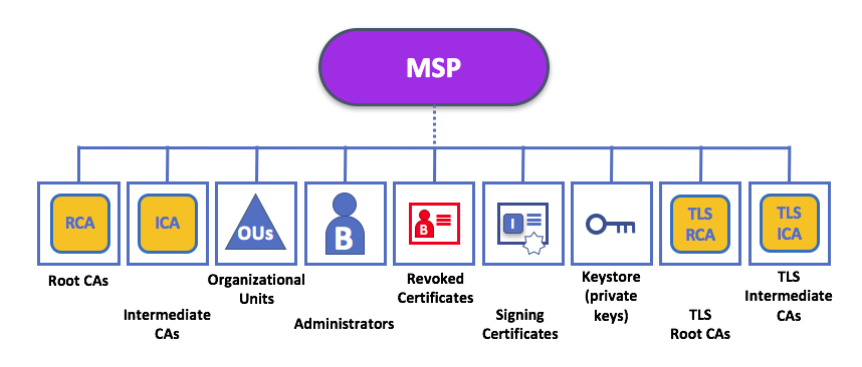


图4-5 MSP结构

* 一组信任的根证书，是整个组织证书信任的基础，根证书可以签发中间层证书；
* MSP的管理员身份证书，管理员可以对MSP中证书进行管理；
* 组织单元列表（可选）；
* 一组信任的中间证书，中间证书由根证书签发（可选）；
* 证书撤销列表，代表被吊销的证书名单（可选）。

##### **3 MSP域**

在联盟链网络中，MSP出现在两个位置：

* 在参与者节点本地（本地MSP）
* 在通道配置中（通道MSP）

本地MSP和通道MSP之间的关键区别不在于它们如何运作——它们都将身份转化为角色，而在于它们作用的范围不同。本地MSP是为客户端和节点（包括Peer和Orderer）定义的，它定义了在节点级别的权限（例如，谁可以是操作结点的Peer结点管理员），每个节点都必须定义一个本地MSP。通道MSP在通道层面上定义了管理权和参与权，应用程序通道上的Peer结点和Orderer结点共用通道MSP的相同视图。因此，实际上MSP是抽象的分层逻辑，每个MSP列出特定管理级别上的角色和权限。图4-6说明了本地MSP和通道MSP在网络中是如何共存的。

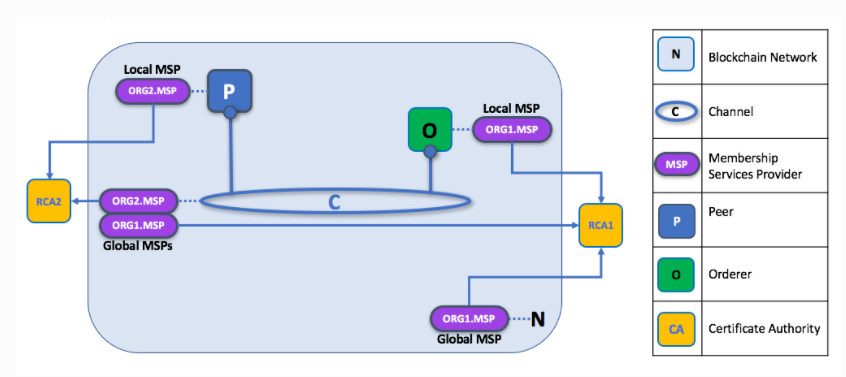


图4-6 本地MSP和通道MSP的关系

Peer结点和Orderer结点的MSP是本地化的，而一个通道（包括网络配置通道，也称系统通道）的MSP是全局化的，被该通道的所有参与者共用。在该图中，系统通道由ORG1管理，而另一个应用程序通道可以由ORG1和ORG2管理。Peer结点是ORG2的成员并由ORG2管理，而ORG1则管理图中的排序节点。ORG1信任来自CA1颁布的身份，而ORG2信任来自RCA2颁布的身份。需要注意的是，这些是管理身份标识，反映了谁可以管理这些组件。所以当ORG1管理网络时，网络定义中确实存在ORG2的MSP。

##### **4 组织和组织单元**

组织是一个逻辑上成员们的管理组。它可以大到跨国公司，也可以小到花店。对一个组织而言，最重要的是在它们单个MSP下管理其成员，MSP允许将标链接到组织，具体实现来说，就是将已经通过验证的成员的公钥及相关属性对应的权限加入到组织MSP的列表中。组织和MSP并不是一一对应的，但每个组织MSP会有独一无二的名称，一个组织也可能会需要多个成员组。在这种情况下，可以采用多个MSP并约定命名，例如，ORG2-MSP-NATTIONAL和ORG2-MSP\_GOVERNMENT。

一个组织也可以被划分为多个组织单元，每个单元都有一定的责任。我们可以将组织单元看作组织内部的一个部门。例如，在物流方A组织中，可能拥有货运组，采购组，技术组，那么就可以分别采用orgA。Transportation，orgA。Purchase和orgA。Technology组织单元，这反映了相隔离的业务流水线。这样使用组织单元的好处是，通过设置不同组织单元的MSP，可以根据属性值定义策略，以限制访问，进行权限管理，或者用于基于属性的访问控制的智能合约。否则，就需要为每个组织创建单独的MSP。

4.2.1.3 权限管理

联盟链中节点的权限管理主要可分为网络准入、组织准入和节点资源访问权限。

##### **1 网络准入**

准入层面主要分为以下步骤：

1. 首先，基于PKI的分层模型允许想要加入网络的用户向Fabric-CA提出请求获取数字证书，提供个人身份信息；
2. Fabric-CA确认用户是否在网络中已注册或存在于证书撤销列表中（Certificate Revocation List，简称CRL）；
3. 第二步确认无误后，Fabirc-CA返回用户注册的数字证书，数字证书上附有用户各项属性值及用户公钥；
4. 用户可使用CA颁发的数字证书向对应联盟链网络申请准入，若联盟链网络认可该身份，则用户加入联盟链网络。

数字证书是能证明证书持有者身份且与持有者相关的属性的文档。最常见的证书类型是符合X.509的证书，它允许在其结构中编码一些用于身份识别的信息。证书还包含更多信息，其中证书持有者的公钥是在他的证书分发的，而他的私钥不是。私钥用于持有者进行签名认证，必须保密。同时，证书内容将通过CA进行签名，如果发生篡改将使证书无效。联盟链网络信任证书颁发者，即证书授权中心（CA），那么该证书就可以证明持有者身份。只要CA安全地保存CA私钥，任何拥有数字证书的人都可以确定有关的证书信息没有被篡改。

##### **2 组织准入**

节点进入联盟链网络后需要加入相应组织，特殊情况下可能节点自身作为组织成员。加入组织同样需要获得组织信任的root-CA或intermediate-CA的认证，然后在本地MSP则将节点写入组织列表，并基于其角色或属性分配权限，最后共识更新组织内所有节点及相关通道的MSP。具体步骤如下：

1. 申请节点向请求加入组织的管理员节点提出交易，申请加入组织；
2. 组织内的管理员节点，审核申请节点身份信息，判断是否合法；
3. 若第二步审核通过，组织信任的root-CA或intermediate-CA将为申请节点分配数字证书；
4. 组织MSP更新写入申请节点的数字证书。申请节点的数字证书数字证书将被打包为交易进行组织内广播，组织内的其他节点达成共识更新本地MSP。

##### **3 资源访问**

Fabric中组织之间通过通道隔离进行资源访问的权限管理，组织内部也可以进一步划分组织单元进行组织内部的资源访问的权限管理。Fabric网络提供的是联盟链服务，联盟是由多个组织构成，组织中的成员提供了节点服务来维护网络，同时通过数字证书来进行权限管理。通道概念可以理解为一个通道对应一个联盟。一个通道上可以由多个组织，每个组织有多个成员，也就是说，成员提供节点服务。需要注意的是，一般情况下，组织加入某个通道A并不代表组织内所有节点都加入通道A。一个组织中的成员可以根据网络中流量的要求，提供不同数量的节点服务来满足网络中的访问量要求。

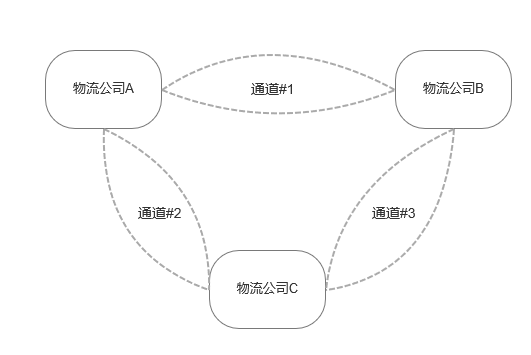


图4-7 通道隔离

组织之间能够通过通道隔离进行资源访问的权限管理。如图4-7所示，三家物流公司可视作三个组织，两两加入到同一个通道中进行相关数据交互，而无需担心被其他人看到，不同通道中所保存的是不同的账本。

组织内部进一步可以划分问组织单元，如图4-8所示，物流公司A划分了组织单元的货运部门、技术部门、市场部门，通道#1共享货运账单，则通道A的MSP将MSP-orgA。Transportation加入到通道#1的MSP成员列表中，即A的货运部门节点才能进行通道#1中账本的查看。

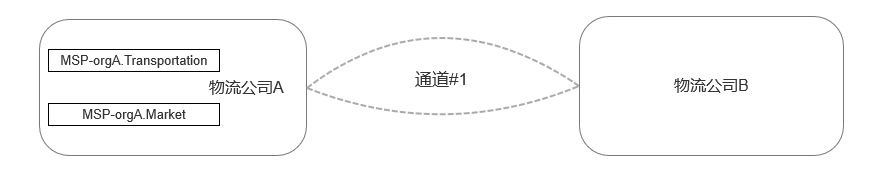


图4-8 组织单元隔离

##### **4 总结**

如果需要加入一个现有的网络，需要一种方法把身份转化为可以被网络识别的东西。MSP是一种机制，使节点能够加入一个需要许可的区块链网络。具体来说，实现用户管理与权限认证的过程输入：

1. 生成相关的证书和签名；
2. 在节点，通道，网络等组件的MSP配置文件中设置相关信息。

也就是说，要想初始化一个MSP实例：首先，要生成用户权限管理和签名认证的证书。然后，每一个Peer节点、Orderer节点、Channel等都需要在本地指定其配置。

如果需要在Fabric网络上进行交易，成员需要这样做：

1. 拥有一个被网络信任的CA颁发的身份；
2. 成为一个被网络成员认可和认可的组织的成员。MSP将身份与组织的成员资格联系在一起，成员资格是通过将成员的公钥（也称为签名证书）添加到组织的MSP来实现的；
3. 将MSP添加到网络上的一个联盟或通道；
4. 确保MSP包括在网络中的策略定义。

4.2.2 物流链下的权限管理

4.2.2.1 网络货运背景

区块链产业即将迎来全面发展的时代，多个传统行业在积极打造垂直领域的区块链解决方案。当前，网络货运行业普遍存在以下痛点：

* 数据孤岛：物流数据分布在不同系统中，没有集成，造成数据失真，存在信任问题
* 财务结算延迟：纸质单据耗用大量的人力物力，查账过程冗长，结账容易延迟。
* 账实不一致：供应商和采购方系统未打通，物流和结算容易滞后。
* 监管追溯难：各参与方在其独立系统中容易篡改数据，追溯困难，合作难以共赢。

目前，国内外物流巨头已着手落地区块链在物流领域的应用。例如，在国外，欧洲最大的港口之一，安特卫普港口，在2017年的时候就开始使用基于区块链的物流系统来实现港口物流自动化；丹麦物流公司马士基和IBM联手，宣布通过区块链技术提供高效安全的物流服务。在国内，阿里巴巴蚂蚁链2017年提出了基于区块链的数字物流解决方案，致力于解决物流运输及资金真实性证明问题，实现数据交叉核验；顺丰区块链医药溯源平台，利用区块链+物联网技术，破解医药供应链数据缺失且可信度低的难题，赋能药品防伪及流通追溯、问题药品追踪找回、居民用药信息查询等医药溯源需求；京东物流依据区块链和电子签名技术，推出了“链上签”平台，为物流供应链内的企业打造一个低成本、高信任的物流供应链协同平台。

4.2.2.2 物流链权限管理

基于联盟链的网络货运平台，最大的优势之一是上链数据的可信化。多联盟共同保存账本，多方可见的特性是上链数据的可信化基础，但由于这一特性，也为上链数据带来了非常大的隐私威胁。因此，决定谁可以在联盟链内发出合法交易，上链数据谁可以进行查看非常重要，权限管理为该问题带来了解决方案。首先，联盟链本身具有一定的准入机制，即已在联盟链网络内的节点，我们可认为其已具有一定的信任基础。但当网络内的节点增加到一定数量级时，隐私安全问题仍然不可忽略。其次，联盟链中不同联盟所保存的账本不同，通过通道隔离技术，限制组织加入通道，通道内的账本只对通道内的组织可见。当然，也可以通过进一步细粒化权限管理，通过启用组织单元，以组织单元的形式加入通道。最后，Fabric中的MSP身份管理模块通过为各节点、组织提供不同的身份进行权限管理。但基于身份的权限管理（Role-base access control，简称RBAC），对于复杂的网络货运场景，权限的划分相对较为粗粒度。随着网络货运的应用场景不断复杂，我们也可以考虑基于X.509数字证书中的属性特征，应用细粒度更高的基于属性的权限管理方案（Attribute-base access control，简称ABAC）。

4.3 区块链中数字签名认证方案

4.3.1 数字签名

数字签名是用于验证数字和数据真实性和完整性的加密机制。数字签名使用非对称密码，意味着可以使用公钥与任何人共享信息。类似于在纸质合同上进行签名以确认合同内容和证明身份，数字签名即可以证实某数字证书内容的完整性，又可以确认其来源，也就是不可抵赖性。数字签名的典型应用场景是，发送者A采用自己的私钥加密信息后发送给B，B使用A的公钥对信息进行解密，从而确保信息是由A发送的。

一个典型的消息签名和验证的场景图4-9所示。Alice通过通信信道发给Bob一个文件（一份信息），Alice可以先对文件内容做个数字摘要，然后用自己的私钥对摘要进行加密（签名），之后同时将文件和签名都发给Bob。Bob收到文件和签名后，用Alice的公钥来解密签名，得到数字摘要，再与对文件进行摘要后的结果进行对比。如果一致，则说明该文件确实是Alice发过来的（因为别人无法拥有Alice的私钥），并且文件内容没有被修改过（摘要结果一致）。

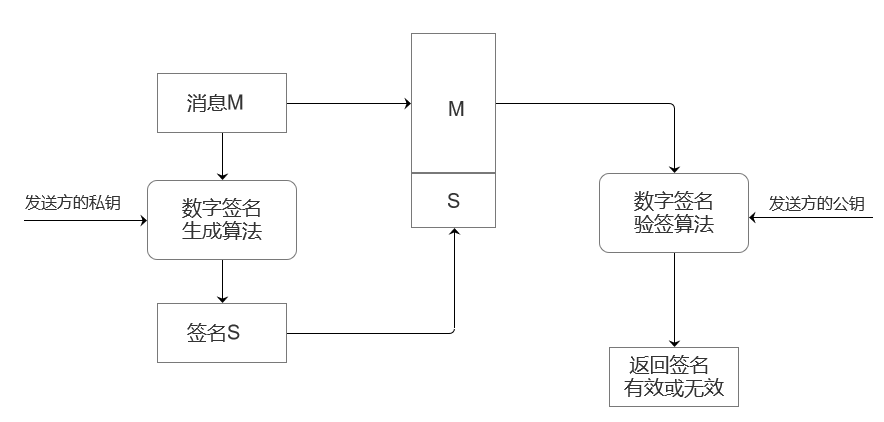


图4-9 消息签名和验证的基本过程

数字签名的主要过程是：

1. 系统初始化，生成数字签名所需的参数；
2. 发送方利用自己的私钥对消息进行签名；
3. 发送方将消息原文和作为原文附件的数字签名同时传给消息接收方；
4. 接收方利用发送方的公钥对签名进行解密；
5. 接收方将解密后获得的消息与消息原文进行对比，如果二者一致，那么表示消息在传输中没有受到过破坏或篡改，反之不然。

在区块链系统中，数字签名涉及公钥、私钥和钱包等工具，具有的作用如下：

一是防伪造。数字签名由私密的持有者自己签署，而其他人无法伪造。

二是防篡改。数字签名与原始文件或摘要一起发送给接收者，一旦信息被篡改，接受者可通过计算摘要和验证签名来判断该文件无效，从而保证了文件的完整性。

三是保密性高。传统合同的手写签名容易丢失，且可能会被造假模仿，文件信息极有可能被泄露篡改，而数字签名可以加密要签名的消息，在网络传播中，可以将报文用接收者的公钥加密，以保证信息机密性。

四是身份确认。在数字签名中客户的公钥是其身份的标志，当使用私钥签名时，如果接收方或验证方用其公钥进行验证并获通过，那么可以肯定，签名人就是拥有私钥的那个人。

在比特币系统中，每个人都有一对密钥（公钥和私钥），比特币系统使用用户的公钥经过转换作为交易账户的地址。图4-10给出了比特币系统中一次交易的签名流程。

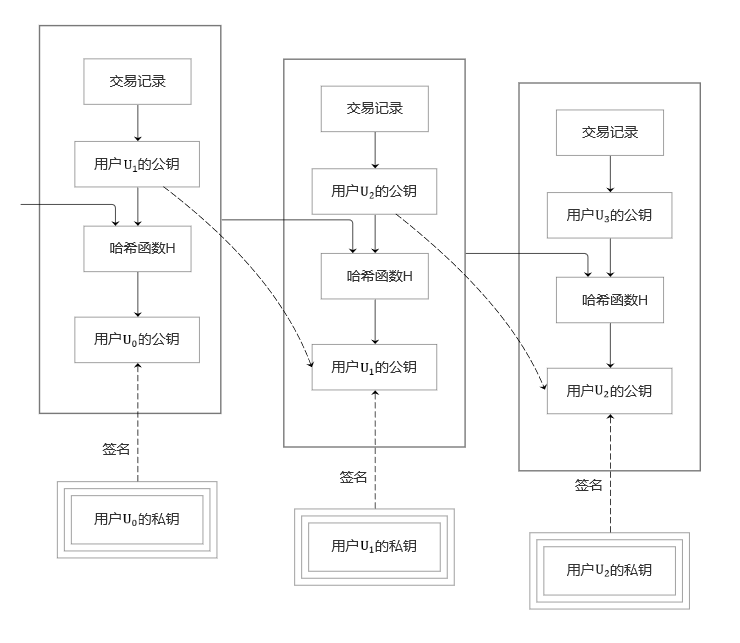


图4-10 比特币系统中一次交易的签名流程

1. 首先，用户写好交易信息：（明文，例如用户转账100元转给用户）；
2. 用户使用哈希算法对交易信息进行计算，得出，然而再使用自己的私钥对签名，即，目的是防止交易信息被篡改。
3. 基于区块链网络，系统将签名和交易信息传递给用户;
4. 用户使用用户的公钥解密，就得到了交易信息的哈希值;
5. 用户使用哈希算法计算交易信息，得出；
6. 对比上面2个哈希值，如果，则交易合法，说明用户在发起交易的时候确实拥有真实的私钥，有权发起自己账户的交易；
7. 网络中每一个节点都可以参与上述交易步骤的验证。

这个示例就是比特币系统中一次交易的签名流程，即将哈希算法与非对称密码算法结合在一起，用于比特币交易的数字签名。

4.3.2 商密标准的数字签名方案

椭圆曲线加密算法（ECC）是Koblitz和Miller两人在1985年提出的，其数学基础是利用椭圆曲线上的有理点构成Abel加法群上椭圆离散对数的计算困难性。废队很密码体制根据其所依据的难题一般被分为三类，大整数分解问题、离散对数问题和椭圆曲线，有时也把椭圆曲线归类为离散对数类。区块链主要使用了非对称加密中的ECC椭圆曲线算法。ECC相比RSA明显具有以下三个优点：

1. 安全性更高，160位的椭圆密钥与1024位的RSA密钥安全性相同；
2. 处理速度更快，在私钥的加密解密速度上，ECC算法比RSA、DSA速度更快；
3. 存储空间占用小、带宽要求低。基于以上优点，ECC广泛被应用于区块链底层的加密系统中。

本节以ECDSA签名算法和EDDSA签名算法为例，进一步说明非对称加密算法在区块链中的应用。

4.3.2.1 ECDSA签名算法

椭圆曲线数字签名ECDSA是使用椭圆曲线密码ECC对数字签名算法DSA的模拟。与普通的离散对数问题和大数分解问题不同，椭圆曲线离散对数问题没有亚指数时间的解决方法，因此椭圆曲线密码的单位比特强度要高于其他公钥体制。

比特币和以太坊中使用的数字签名方案是ECDSA，其所选的椭圆曲线参数是Secp156k1。Secp256k1是指基于有限域上的椭圆曲线，由于其特殊构造的特殊性，其优化后的实现比其他曲线性能上高30%。Fabric提供了模块化的、可拔插的密码服务，该服务由bccsp（Block Chain Crypto Service Provider）模块提供，它包含了ECDSA算法、RSA算法、AES算法、SHA算法及可拔插的国密算法。但对于可拔插国密算法而言，Fabric支持国密并非仅仅是在bccsp中增加1个国密实现那么简单，还需要让数字证书支持国密，让数字证书符合X.509。在数字证书支持国密后，还可能需要进一步考虑，是否需要TLS证书使用国密数字证书，是否在通信过程中也使用国密算法。

##### **1 主域参数**

首先选择一个有限域，一个奇素数；或者，一个2的幂。这样，在的情况下，选择的域便是，p是模数；在的情况下，选择的域便是。ECDSA主域参数有以下各部分组成：

1. 选定一个域，或者；
2. 域表示法表示域中的元素；
3. 中的两个域元素a和b；
4. 两个中的域元素和来定义上的一个基点；
5. 基点的阶n，；
6. 余因子h。

##### **2 密钥对**

确定椭圆曲线的主域参数，便可确定ECDSA的密钥对。签名实体A必须在产生公钥之前确定主域参数的有效性。实体A可以按如下算法产生公钥：

1. A在区间中选取一个随机整数d；
2. 计算；
3. A的公钥是，私钥是。

##### **3 生成算法**

ECDSA签名方案的参与者有：签名实体A，可信的中间机构CA负责产生主域参数和密钥，验证签名的实体B。

输入参数为：签名消息，实体A所需要的主域参数，以及相应密钥对；

输出为：椭圆曲线数字签名；

产生椭圆曲线数字签名的算法如下：

1. 选择一个随机数，；
2. 计算；
3. 计算：如果，则回到步骤（1）
4. 计算；
5. 计算；
6. 计算，若，则回到步骤1；
7. 对消息的签名为;

最后签名者A把消息和签名发送给接收者。

##### **4 验证算法**

1、ECDSA签名的验证

输入参数为：A的数字签名，签名消息，实体B主域参以及A的公钥；

输出为：接受或者拒绝签名。

验证过程如下：

1. 验证和是内的整数；
2. 计算；
3. 计算;
4. 计算；；
5. 计算；
6. 如果，拒绝签名；否则，，计算；
7. 如果，接受签名；否则拒绝签名。

2、ECDSA验证签名的依据

如果签名是消息的合法签名，有；

则





再有，因为

其中的横坐标的横坐标为；即有。

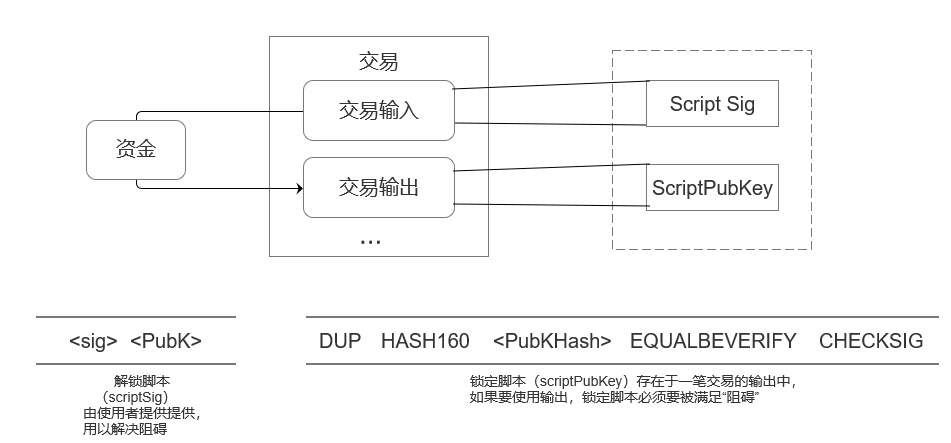
##### 5 ECDSA安全性

ECDSA的安全性是指在选择明文下是不可伪造的，而一般采取此种攻击的对手是在获得有选择的一系列消息的签名后（不包括），来获得关于某个消息的有效签名。

在离散对数问题是困难的而且所使用的哈希函数是随机函数的前提下，Pointebeval和stern已经证明了对于选择明文共计ECDSA在现有情况下是不可伪造的。Brown已经证明了在基本群为普通群而且所使用的的哈希函数是抗碰撞的前提下，ECDSA本身是安全的。

4.3.2.2 ECDSA在区块链中的应用

在区块链系统进行交易的环节中，一个很重要的信息是签名。例如，Alice给Bob付款，当Alice准备好这笔交易者后，需要将交易信息发给全网所有节点，而其他人看到这笔交易的时候，他们需要去认证这笔交易是否由Alice发起的。这时候就需要用到数字签名技术。以比特币系统为例，交易的具体流程如图4-11所示。

图4-11 比特币交易流程

交易模块分为交易输入呵交易输出，一笔交易由多个输入或者多个输出组成。图11右边虚线方框里面的Script Sig是一个脚本签名，这个字端属于交易输入，ScriptPubKey属于交易输出。脚本包含一些数据和操作码来支持着个脚本语言的运行。ScriptSig和ScriptPubKey也可以直观解释为解锁脚本和锁定脚本。

如果付款人要花费之前已有的资金，也就是要花费一个未花费的交易输出，那么他需要在另一笔交易里面构造一笔交易输入，这个交易里面会设置签名字段和公钥，以此证明他有资格花费未花费的交易输出。这种情况叫作解锁脚本，即把之前已有的资金打开进行使用。这个过程中，解锁脚本有两个字段数据，<sig>和<PubK>，这两个字段分别代表签名和公钥，两者由使用者提供。锁定脚本里也有一些字段和操作码，DUP是复制操作码；HASH160是哈希操作码；<PubKHash>是字段；EQUALIVERIFY是验证操作码；CHECKSIG也是验证操作码。其整体流程是：执行复制;进行哈希;执行字段;验证是否相等;校验签名是否正确。在验证签名的环节，系统就会调用ECDSA里面的验签算法。

图4-12给出了脚本语言堆栈处理过程。这里会将两个脚本语言进行堆栈处理，即所有字段和操作码全部罗列在右边，解锁脚本有2个，锁定脚本有5个。

第一步，将第一个字段<sig>解锁，置于堆栈的底部；

第二步，向右移动一格，将<PubK>解锁推送至堆栈顶部，置于<sig>的上面；

第三步，DUP是复制操作码，它会将刚刚解锁脚本里面的<PubK>（公钥）进行复制，此时会有两个<PubK>，结果被推送至堆栈顶部；



图4-12 脚本语言堆栈处理过程（1）

第四步，HASH160将复制的公钥进行哈希运算，使之变成<PubKHash>（公钥哈希）；

第五步，EQUALVERIFY操作码将PubKHash和用户的PubKHash对比，如果一致，则都被移除，然乎继续执行。这里是为了对公钥进行认证，来表明公钥和地址是否康铜；

如果第五步匹配的话，继续执行第六步验证签名，这里就用到了ECDSA，也是交易中最关键的一步。

在图4-13中会看到，在执行最后一个操作码的时候，堆栈里面有两个字段。此时CHECKISIG操作码会核查签名<sig>是否与公钥的<sig>匹配，如果匹配，则会在顶部显示TRUE。如果返回值是TRUE的话，此时发送者也表明了其对钥发送的比特币有所有权和使用权。

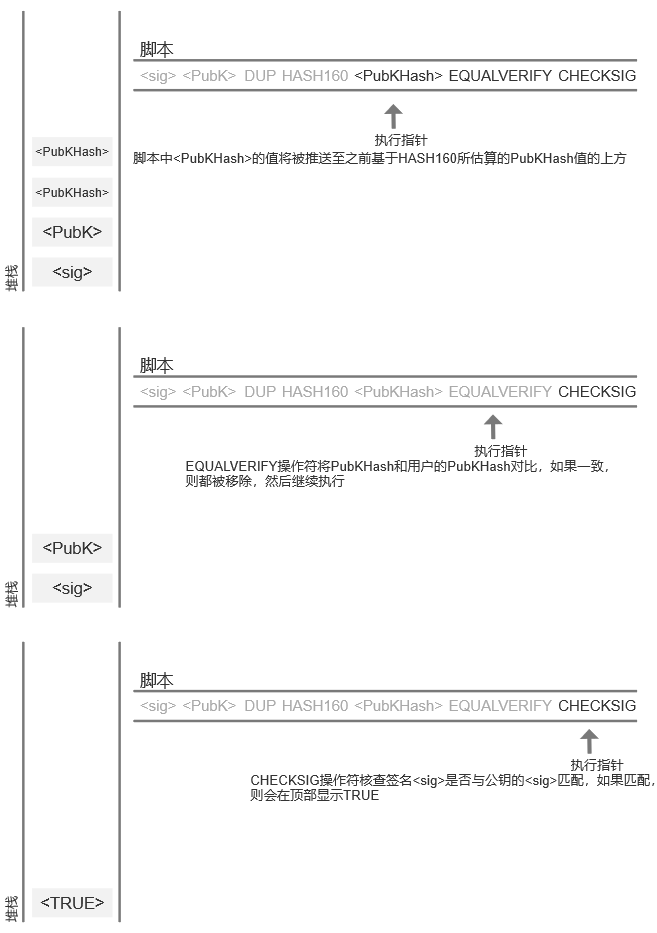


图4-13 脚本语言处理堆栈过程（2）

以上就是椭圆曲线签名ECDSA在区块链中的主要应用环节和场景。

4.3.2.3 EDDSA签名算法

在ECC算法基础上构建的数字签名算法被称为ECDSA算法，经过美国国家变准宇技术标准所批准的曲线有多条，例如secp256r1、secp256k1等，但现有的ECC算法中的曲线被指出存在后门。

EDDSA签名算法是由Schnorr签名发展变化而来。EDDSA从某种意义上来说也属于椭圆曲线密码学，不同的是它采用扭曲爱德华兹曲线（Twisted Edwards curves）作为椭圆曲线，同时采用的签名机制也不同于ECDSA算法。由曲线和参数的选择不同又可以划分为Ed25519呵Ed448等算法，其中Ed25519算法使用较多，由著名密码学家Daniel J。Bernstein等人设计实现，采用的曲线参数完全公开，并说明了参数选取的意义，保证曲线中并未内置后门，它具有运算速度快、密钥较短、安全性高等优点。

Ed25519公私钥生成宇签名验证的方式如下：

1. 公私钥生成
   1. 随机选取一个长度为b比特的二进制数作为私钥k；
   2. 对k进行哈希计算，产生一个长度为2b比特的值，取哈希值h的前b比特，并置为0，置为1，将置位后的结果作为x，用于产生公钥A；后b比特的值作为随机数y，用作签名流程的计算；
   3. 计算，A即为公钥。
2. 签名流程
   1. 进行签名时，需要持有私钥k，执行上述公私钥生成算法后得到公钥A与随机数y。
   2. 计算随机数；
   3. 计算随机点；
   4. 计算签名;
   5. 其中得到的便是数字签名。
3. 验证流程

进行验证时，验证这只需要知道公钥A、签名和消息M即可验证签名过程是否正确。即计算是否成立。观察整个签名过程，我们不难发现，当对同一个消息M进行签名时R呵s都是固定的，所以说EDDSA是一种确定性的签名算法，不像ECDSA那样每次签名都根据所选取的随机数的变化而不同，所以EDDSA的安全性也就不依赖于随机数生成器。

4.3.2.4 EDDSA在区块链中的应用

Ed25519系列曲线自2006年发表以来，在学术界外无人问津，直到2013年爱德华·斯诺登曝光棱镜计划后，该算法突然大火，大量软件如OpenSSH都迅速增加了对Ed25519系列的支持。美国国际标准与技术研究所选定的椭圆曲线可能迟早要退出历史舞台，而Ed25519则有可能是大势所趋。

与目前广泛使用的椭圆曲线相比，Ed25519曲线算法各参数的选择直截了当，非常明确，没有任何可疑之处。而且Ed25519系列的椭圆曲线经过特别设计，尽可能将吃错的概率降到了最低，可以说是在实践上最安全的加密算法，且理论安全性极高，等价于RSA的3000比特密钥的安全性。

同时Ed25519系列曲线也是目前最快的椭圆曲线加密算法。前文中演示过的Ed25519公私钥生成、签名呵验证的性能都极高，一个4核2。4GHz的Westmere CPU，每秒可以验证71000个签名，性能远超美国国家标准与技术研究所推荐的系列椭圆曲线算法，且该签名过程不依赖于随机数生成器，不依赖哈希函数的防碰撞性，没有时间通道攻击的问题，并且签名（64字节）和公钥（32字节）都很小。相比之下，基于椭圆曲线secp256k1的ECDSA由于在Bitcoin中的部署，逐渐成为区块链项目中默认的签名机制。然而在Bitcoin诞生的初期，工程项目中更多采用的基于椭圆曲线secp256r1的ECDSA签名机制。Bitcoin系统最终采用secp256k1的真正原因已不可知，我们只能进行揣测，也许是secp256k1支持的自同态映射能够加速签名验证过程。但是仍需指出，在区块链场景中应用基于secp256k1的ECDSA存在诸多方面的挑战，稍有不慎就可能在区块链网络中引发安全问题，甚至导致数字货币资产的损失。

因此，为规避诸多安全问题，以及获得更高的性能，中国领先的超融合架构产品与企业云解决方案提供商smartX在开发过程中几经考量决定抛弃secp256k1而采用安全性呵性能表现更好的Ed25519。除此之外，著名的匿名货币monero（门罗币），首个使用零知识证明机制的区块链系统Zcash，基于区块链网络的支付系统Stellar，致力于提供一个高速、注重隐私、用于主权身份认证的分布式公共账本的Evernym，基于区块链的分布式云存储平台Sia等系统与平台，均使用了EDDSA25519算法进行数字签名。

4.3.3 国密标准的数字签名方案

区块链具有去中心化，可信，可追溯的分布式特性，不断被应用于金融、物流、能源等领域。值得注意的是，区块链所具备的可信是建立在密码学原理的基础之上，包括私有链及联盟链的准入机制，链上资产的确权，节点身份的认证及权限管理，所以我们在使用区块链的同时必须关注区块链底层所应用的密码协议。据此，国家密码管理局发布的区块链密码应用技术要求明确指出：区块链中配置和使用的密码算法，应采用国家密码管理主管部门批准的算法，具体包括分组密码算法、非对称密码算法、密码杂凑函数等。中国软件行业协会发布的区块链平台基础技术要求中提到：区块链平台符合国家密码管理相关规范。

目前，国密算法SM2、SM3的可证明安全性已经达到了密码算法的最高安全级别，其实现效率相当或略高于高级标准的密码算法，相较于比特币、超级账本等现有区块链架构选用的国际通用密码算法ECC、SHA-256，拥有安全、稳定、高效等优势。目前各主流区块链公司都有尝试将国密应用在区块链上，这些工作主要集中在：

1. 用SM3代替SHA-256等作为密码杂凑Hash算法；
2. 用SM2替代ECDSA签名算法；
3. 用SM2证书代替RSA、ECDSA证书。

4.3.3.1 国密算法

国密算法和商用算法一样，可分为对称密码算法、非对称密码算法、哈希密码算法三类，具体如表1所示：



表1 国密算法类型

1）SM1 对称密码算法

SM1算法是分组密码算法，分组长度为128位，密钥长度为128比特，算法安全保密强度及相关软硬件实现性能与AES相当，算法不公开，仅以IP核的形式存在于芯片中。采用该算法已经研制了系列芯片、智能IC卡、智能密钥钥匙、加密卡等安全产品，广泛应用与电子政务、电子商务及国民经济的各个应用领域。

2）SM2椭圆曲线公钥密码算法

3）SM3密码杂凑算法

SM3密码杂凑算法给出了杂凑函数算法的计算方法，此算法适用于商用密码应用中的数字签名和验证，消息认证码的生成与验证和随机数的生成，可满足多种密码应用的安全需求，在SM2，SM9标准中使用。SM3算法采用Merkle-Damgard结构，消息长度为512位，摘要值长度为256位。SM3算法的压缩函数与SHA-256的压缩函数具有相似的结构，但是SM3算法的设计更加复杂，比如压缩函数的每一轮都使用2个消息字，SM3算法的安全性高于SHA-256。

4）SM4对称算法

SM4对称算法是我国无线局域网标准WAPI中所采用的分组密码标准，随后被我国商用密码标准采用，作为我国商用密码得分组密码标准，预计SM4在国内的敏感但非机密的应用领域会逐渐取代3DES、AES等国外分组密码标准，用于通信加密、数据加密等应用场合。

5）SM7对称算法

SM7算法是一种分组密码算法，分组长度为128比特，密钥长度为128比特。SM7适用于非接触式IC卡，应用包括身份识别类应用，票务类应用，支付与通卡类应用。

6）SM9标识密码算法

SM9标识密码算法是国密标准中两个数字签名算法标准之一，也是国密SSL协议中几个密码套件采用的签名算法，因此可以预计会成为一个应用较为广泛的重要数字签名算法。

1. ZUC祖冲之算法

祖冲之算法是中国自主研究的刘密码算法，是运用于移动通信4G网络中的国际标准密码算法。该算法包括祖冲之算法(ZUC)、加密算法(128-EEA3)和完整性算法(128-ELA3)三个部分，目前已有对ZUC算法的优化实现，有专门的针对128-EEA3和128-ELA3的硬件实现与优化。

4.3.3.2 SM2/SM3算法

SM2椭圆曲线公钥密码算法（elliptic curve cryptography，ECC）是我国公钥密码算法标准，SM2算法的主要内容包括3部分：数字签名算法；密钥交换协议和公钥加密算法（以下分别称为SM2签名算法；SM2密钥交换协议和SM2加密算法）。ECC所基于的椭圆曲线具有以下性质：1）有限域上椭圆曲线在点加运算下构成有限交换群，且其阶与基域规模近似；2）类似于有限域乘法群中的乘幂运算，椭圆曲线多倍点运算构成一个单向函数。在多倍点运算(也称点乘运算)中，已知多倍点与基点，求解倍数的问题称为椭圆曲线离散对数问题(elliptic curve discrete logarithm problem，ECDLP)。对于一般ECDLP，目前只存在指数级计算复杂度的求解方法。与大数分解问题( integerfactorization problem，IFP)及有限域上离散对数问题( discrete logarithm problem，DLP)相比ECDLP的求解难度要大得多。因此，在相同安全程度要求下，ECC 较其他公钥密码算法所需的密钥规模要小得多，表2列出了这几种公钥密码算法在同等安台强度下的私钥位长比较。

表2 RSA(或DSA)和ECC的私钥位长比较

由于在相同安全强度下ECC比 RSA的私钥位长及系统参数小得多，这意味着应用ECC所需的存储空间要小得多，传输所用的带宽要求更低，硬件实现 ECC所需逻辑电路的逻辑门数要较RSA少得多，功耗更低。ECC的优势使其成为了最具发展潜力和应用前景的公钥密码算法，至2000年国际上已有多个国家和行业组织将 ECC采纳为公钥密码算法标准。在此背景下，我国从2001年开始组织研究自主知识产权的ECC，通过运用国际密码学界公认的公钥密码算法设计及安全性分析理论和方法，在吸收国内外已有ECC研究成果的基础上，于2004研制完成了SM2算法。SM2算法于2010年12月首次公开发布，2012年3月成为中国商用密码标准(标准号为GM/T 0003-2012），2016年8月成为中国国家密码标准(标准号为GB/T 32918-2016)。

SM2算法基于求解离散对数问题，是我国自主研发的椭圆曲线密码算法，相较于ECC-256算法，SM2算法在解密正确性判断、明文编码问题、对待加密数据长度的限制及加密计算等方面具有更高的效率。在实际应用中，SM2算法具有速度快、损耗低的特点，比ECC-256算法更具有优势。数字签名算法由一个签名者对数据产生数字签名，并由一个验证者验证签名的可靠性。每个签名者有一个公钥和一个私钥，。公私钥用于产生签名，验证者用签名者的公钥验证签名。

SM2数字签名算法中，作为签名者的用户A的密钥对包括其私钥和公钥，用户A具有位长为的可辨别标识，记是由整数转换而成的2B数据，签名者和验证者都需要用密码杂凑算法求得用户A的杂凑值。SM2 数字签名算法规定为SM3密码杂凑算法，主要参数对应如表3所示。



表3 SM2参数对照表

算法主要分为4个步骤，实现消息签名及验证：

1. 密钥对生成
   1. 用随机数发生器生成私钥： 
   2. G为基点，生成公钥 ：
2. 用户哈希生成；
   1. 计算用户A的哈希值:
3. 数字签名生成

数字签名分为消息预处理，签名计算两部分。设待签名的消息为M，为了获取消息M的数字签名，具体签名过程如图4-14所示。

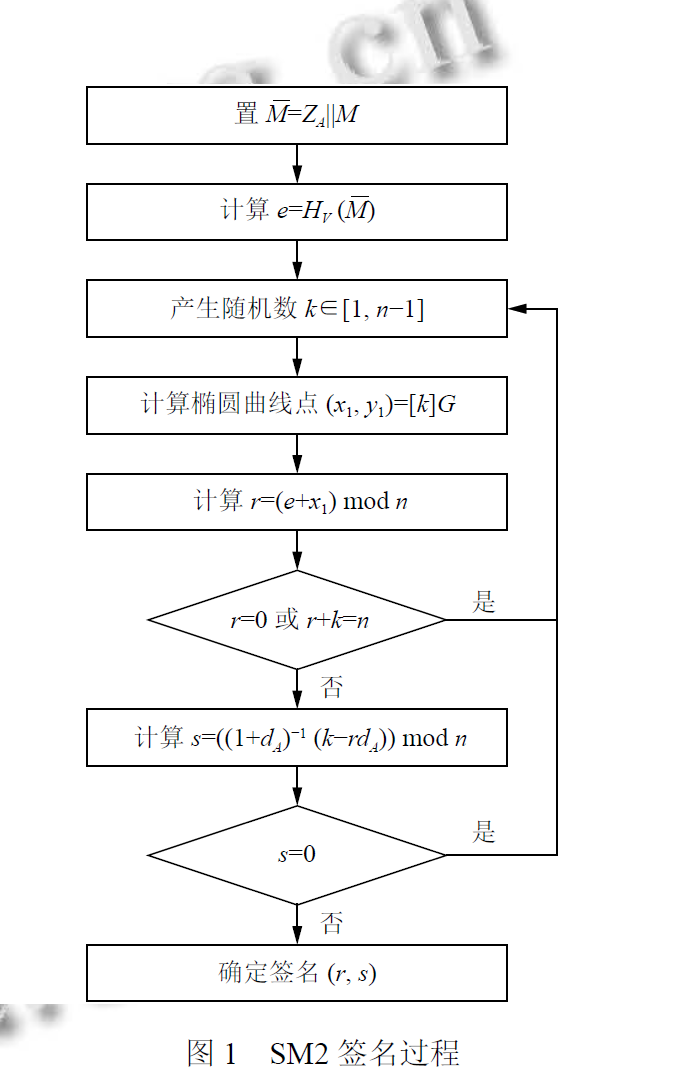
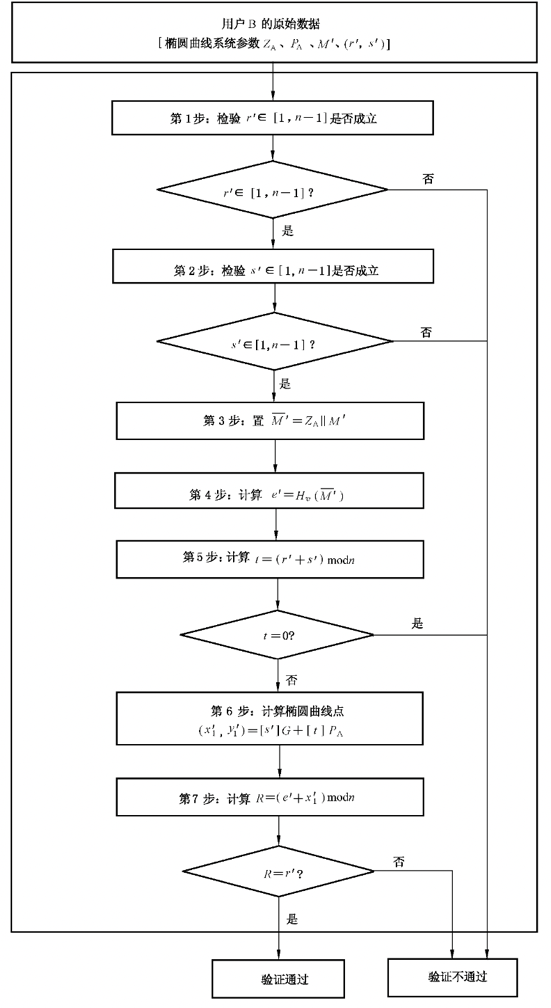


图4-14 SM2签名过程 图4-15 SM2验签过程

* 1. 置；
  2. 计算；
  3. 用随机数发生器产生随机数；
  4. 计算椭圆曲线点，将的数据类型转换为整数；
  5. 计算，若，则返回；
  6. 计算，若，则返回；
  7. 将的数据类型转换为字节串，消息M的签名为。

4）数字签名认证

如图4-15所示，SM2延签算法通过用户哈希、用户公钥对数字签名进行验签计算，已确定签名是否通过，完成签名的验证过程。

4.3.3.3 安全性及性能分析

##### **1 安全性**

针对数字签名算法的最强攻击行为是自主选择消息攻击( adaptively chosen-message attacks)，攻击者可以访问签名预言机( signing oracle)，除攻击者要伪造签名的消息外，他可以任意选择消息进行签名而获得有效的消息/签名对。攻击者如果达到以下目标之一，则称数字签名算法被攻破：

1. 完全攻破( total break):攻击者获得签名私钥，可以对任意消息伪造签名，这是最严重的攻破;
2. 一般性伪造(universal forgery):攻击者建立一个有效的算法来模仿签名，模仿签名的成功率足够高;
3. 存在性伪造(existential forgery):也称随机消息签名伪造，攻击者利用已有的消息/签名对，可以生成新的消息/签名对，新的消息与原有消息/签名对具有相关性，攻击者不能自主选择。

上述3个攻击目标中，存在性伪造是最低的，对于一个数字签名算法，如果攻击者采用最强的攻击行为，仍然不能达到最低的攻击目标，则该数字签名算法是安全的。Goldwasser等人[12-13]提出的自主选择消息攻击下存在性不可伪造(EUF-CMA)，已经成为评估数字签名算法安全性的一个标准概念。

Menezes和 Smart提出了针对数字签名算法的密钥替换攻击，攻击者拥有公钥以及该公钥对应的消息/签名对，试图生成另一个公钥，使得用验证仍然是有效的。有多项研究已经表明，EUF-CMA安全的数字签名算法，仍然可能被成功实施密钥替换攻击。SM2数字签名算法属于广义ELGamal数字签名算法范围。针对密钥替换攻击，SM2数字签名算法采取的防御方法是将签名者ID、公钥和源消息一起Hash，在 Hash算法安全的前提下，可以抵抗密钥替换攻击。

##### **2 实现效率**

ECC运算过程中最耗时的运算是椭圆曲线点乘，无论是软件、FPGA还是集成电路实现，点乘运算占据整个算法运算时间的比例一般超过80%。此外，算法中如果存在有限域上元素的求逆运算，也将占用不可忽略的时间比例。故一般通过计算点乘运算和求逆运算的次数，就能大致评估ECC的实现效率。

基于上述判断，可以将SM2算法与国际上同类的ECC算法进行实现效率对比分析: 1)SM2 数字签名算法的实现效率与ECDSA相当，但 SM2数字签名算法中求逆运算是可以预计算的(生成签名私钥时，应检查不为0， ECDSA中的求逆运算则不能，所以通过预计算可以使SM2数字签名算法实现效率略高于ECDSA;2)SM2密钥交换协议的实现效率与ECMQV相当;3)SM2公钥加密算法的实现效率与ECIES相当。

ECC较RSA在实现效率方面有明显优势，尤其适合实现到智能IC卡芯片等资源受限的环境中。作者在一款智能C卡芯片中，验证了这一判断;采用相同的硬件协处理器来实现 SM2数字签名算法和RSA 签名算法，该协处理器的核心运算部件是64b×64 b的乘法器，运算数据暂存于双端口SDRAM中，运行于30MHz时钟频率下的实测性能为:SM2签名118次/s;RSA-1024签名、非CRT模式32 次/s，CRT模式76 次/s:RSA-2048签名、非CRT模式4次/s，CRT模式13次/s。需要指出的是SM2算法的安全强度是高于RSA-2048的，大致与RSA-3072相当，如果在IC卡中运行RSA-3072，将是相当耗时的运算，而SM2算法所需的运算负载显然要小很多。SM2算法的实现效率可以满足从服务器端到客户端的不同种类的信息安全设备的需求。

4.3.3.4 国密化的应用和挑战

目前各主流区块链公司都有尝试将国密应用在区块链上，例如旺链的VoneBaas平台的国密改造等。这些工作主要集中在：（1）用SM3代替SHA-256等作为默认的密码杂凑算法；（2）用SM2替代ECDSA签名算法；（3）用SM2证书代替RSA、ECDSA证书。对于上层的协议如SSL则涉及较少，与区块链相关的硬件设施（如硬件钱包）也很少采用国密标准进行设计。此外区块链中比较热门的密码学协议如：零知识证明、多方安全计算等，因为国密中没有针对性的标准，很难将其国密化。

因为国密算法的开发及优化涉及大量的专业知识且工作量巨大，多数公司还是选择在开源的国密平台实现（如GmSSL）在区块链系统上进行集成，由于这些区块链在设计之初并没有考虑国密的兼容性问题，集成的工作量仍然很大，为了解决这个问题，HyperLedger成立了URSA项目，旨在为fabric等底层区块链系统打造一个支持多数密码算法（包含国密）的密码库。

随着区块链领域对共识算法、签名方案、隐私保护、数据安全共享等需求的不断发展变化，国产密码体系涵盖基础密码已无法满足区块链的应用需求，需要能够和现有的国产密码安全参数保持兼容的一系列新型密码方案。区块链领域的特殊性使得密钥、密码消息有关的数据格式和传统PKI体系有一定差异，因此也需要制定有针对性的规范。区块链，特别是公有链，通常需要在全球范围内部署，因此国产密码体系需要国际化，不仅是算法和标准的公开，还需要设计原理和安全参数选择的公开透明。

一个成功的区块链系统的部署运行生命周期完全可能超过某个密码算法标准的生命周期，目前国产密码标准中的算法的安全性为128比特，不包含支持抗量子计算机的算法，也不包含支持适用于物联网轻量级的密码算法。区块链应用国产密码算法的同时，也应注意密码算法的灵活性区块链系统在设计上，需要满足无需硬分叉的条件下替换和更新密码算法，以满足应用场景、合规性、性能或者安全性的要求。

4.3.4 哈希函数

4.3.4.1 哈希函数概述

哈希函数，又称单向散列函数、杂凑函数，其思想是接收一段明文，然后以一种不可逆的方式将它转换成一段密文。哈希函数是一类数学函数，可以在有限合理的时间内，将任意长度的消息压缩为固定长度的输出值，并且是不可逆的。其输出值称为哈希值，也称为散列值、信息摘要。以哈希函数为基础构造的哈希算法，在现代密码学中扮演着重要的角色。哈希函数可用于数字签名、消息的完整性检测、消息起源的认证检测等。其数学表达式如下：

一个优秀的Hash函数具有以下四个特性：

1. 正向快速。给定明文和Hash算法，在多项式时间和有限资源内能计算得到Hash值。
2. 逆向困难。给定（若干）Hash值，在多项式时间内很难或基本不可能逆推出明文。
3. 输入敏感。原始输入信息发生任何改变，新产生的Hash值都会出现很大不同。
4. 冲突避免。很难找到两段内容不同的明文，使得它们的Hash值一致。

在实际应用中，Hahs函数是基于压缩函数进行计算的。如图4-16所示，给定任意长度的消息，Hash函数均为输出长度为m的哈希值。压缩函数的输入为对应的消息分组和压缩函数上一级计算输出。

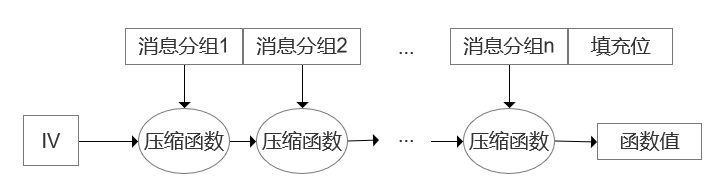


图4-16 哈希函数的工作模式

4.3.4.2 典型Hash算法

目前主要的Hash函数由MD、SHA系列算法和SM3国密算法，具体分类如图4-17所示。

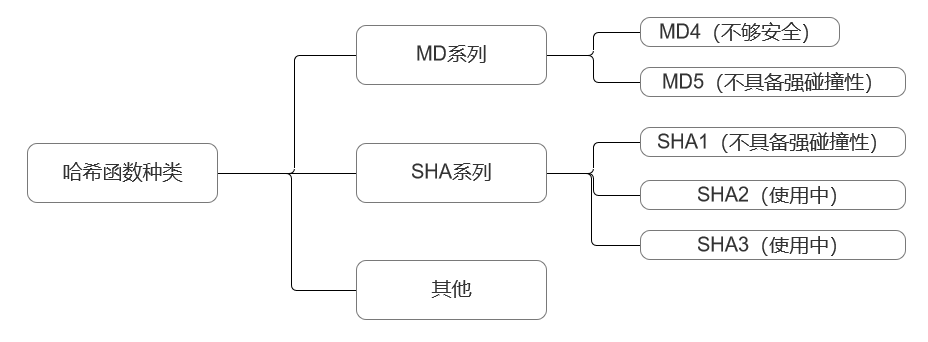


图4-17 主要哈希函数种类

##### 1 MD5算法

MD5（单向散列算法）输入为任意长度额信息，经过处理，输出为128位的信息。不同的输入得到不同的输出，根据128位的输出结果不能反推出输入的信息。MD5算法具有防止被篡改、防止直接看到明文、防止抵赖等用途。MD5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组。经过填充、记录信息长度、装入标准幻数和四轮循环运算后，得到的输出输出由4个32位分组组成，将这4个32位分组级联后就生成了128位的散列值。

##### 2 SHA256算法

2002年，美国国家安全标准与技术研究所推出SHA-2系列Hash算法，其输出长度可以取224位、256位、384位、512位。其中SHA-256是最常用的算法，其输入是最大长度小于位的消息，输出是256位的消息摘要，输入以512位的分组为单位进行处理。比特币和以太坊系统采用的是SHA256，Fabric 2。3采用的是BLAKE2b，BlAKE2b在保持高度安全的同时拥有极高的效率，经测试，在一个现代CPU上计算BLAKE2b实际上比SHA-3要快三倍。

##### 3 SHA-3算法

SHA-3 算法是第三代标准的哈希函数，基于 Keccak 算法实现，可建立从任意长度输入到任意长度输出的映射。与之前的哈希算法有所不同，SHA-3 算法是基于置换的加密函数，整体采用Sponge结构，分为吸收和榨取两个阶段。SHA-3的核心置换f作用在5\*5\*64的三维矩阵上，整个f共有24轮，每轮包括5个环节。Keccak算法的立体加密思想和海绵结构，使SHA-3优于SHA-2。

##### 4 SM3算法

SM3算法也是一种哈希算法，由中国国家密码管理局在2010年发布。该算法是我国自主设计的密码杂凑算法，适用于商用密码应用中的数字签名呵消息认证码的生成与验证以及随机数的生成，可满足多种密码应用的安全需求。SM3算法的输出长度为256比特，从输出长度来看，SM3算法的安全性要高于MD5算法和SHA-1算法。

如图4-18所示，SM3的处理过程如下：

1. 填充。使填充完的数据的长度是512的整数倍，先在数据尾部加一个“1”；然后把原始数据的长度用64比特表示，放在最后面；“1”和原始数据长度的64比特之间的空余比特位全部填充“0”。
2. 分组。把填充后的信息按照512比特进行分组，如果分成N组，就是。
3. 迭代压缩，得到最后的杂凑值（哈希值）。

如果信息分组分为N组，那么就是最后得到的杂凑值。对长度为比特的消息m，SM3杂凑算法经过填充和迭代压缩，生成杂凑值，杂凑值长度为256比特。

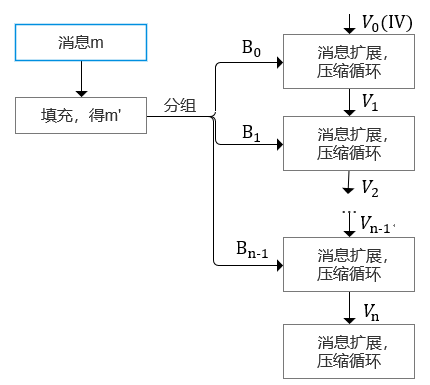


图4-18 SM3算法流程

SM3的压缩函数与SHA-256的压缩函数具有相似的结构，但SM3设计更加复杂，如SHA-256压缩函数的每一轮使用都使用2个消息字。SM3在避免高概率局部碰撞方面效果显著，能够有效抵抗强碰撞性“差分分析”、碰撞性“线性分析”等密码分析方法。

##### 5 BLAKE2b算法

BLAKE算法是SHA-3候选会议中进入第三轮的5个候选算法之一，综合考虑算法的安全性和软硬件实现性能，虽然美国国家标准与技术研究所最终选取了Keccak作为SHA-3标准，但不可否认BLAKE算法在某些方面依然保持优越。BLAKE2b算法是BLAKE算法在64位CPU上的高度优化优化，经测试，BLAKE2b i常见的MD5，SHA-2，SHA-3更快，同时提供不低于SHA-3的安全性。使用Blake2b算法中最具有代表性的币种就是siacoin，这是一个去中心化云存储的区块链项目。此外，Fabric 2。3也采用了BLAKE2b作为Hash算法。

4.3.4.3 哈希函数在区块链中的应用

Hash函数在区块系统中得到了广泛应用。区块链通过哈希算法对一个交易区块中的交易信息进行运算，并把信息压缩成由一串数字和字母组成的散列字符串。区块链的哈希值能够唯一而准确地标识一个区块链，区块链中任何节点通过简单的哈希计算都可以获得这个区块的哈希值，计算出的哈希值没有变化也就意味着区块中的信息没有被篡改。Hash函数主要应用方式为通过hash函数在区块链中构建区块链和默克尔树（Merkle Tree）。构建区块的区块头和区块体都要应用Hash函数。

当哈希算法被集成到区块链协议中的时候，比特币系统选择了SHA256算法，而以太坊选择了改良版的SHA-3算法，Fabric1。1中使用的也是SHA-3算法，但在最近的Fabric2。3中选择了BLAKE2b算法。Hash函数在区块链中有多种应用，在比特币系统中具体有三种功能作用：

1. 对交易信息进行压缩和验证；
2. 用于工作量证明，形成共识；
3. 用于生成比特币钱包地址。

##### 1 构建区块链链表

区块链是一个基于Hash指针构建的有序的，反向链接的交易块链表，也就是说在区块链中每个区块都通过Hash指针连接到前一个区块上。与普通指针不同的是，Hash函数的值是通过数据计算出来的且指向数据所在位置，所以Hash函数可以告诉我们数据存储位置及数据的Hash值。通过Hash函数，我们可以很容易判断出数据是否被篡改。区块链的结构就是由创世区块开始，之后的每个区块通过Hash函数进行连接，每一个区块中都包含了前一个区块的Hash函数值，这样后面区块不仅可以查找到前面所有区块，也可以验证前面区块数具有没有被篡改，从而保证了区块链不易篡改的特性。

##### 2 构建Merkle Tree

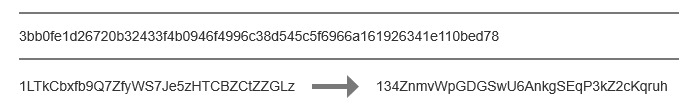
Merkle Tree，也称Hash Tree，顾名思义，就是存储Hash的树，其是区块链的基本组成部分。Merkle树的叶子是数据块的Hash值，非叶节点是其对应子节点串联字符串的Hash值。Merkle Tree在区块链中用于组织和记录存储在区块中的交易，以便高效地验证某个交易是否在区块中。Merkle Tree的各个节点使用Hash指针进行构建，通过不断的递归计算节点的Hash值，直到最后算出一个最终的Hash值。这样做的好处是，由于区块链要处理的信息量非常大，将每个区块内的所有数据直接以序列的方式存储会非常低效且耗时，但是利用哈希函数可以对信息进行压缩和验证。使用Merkle树可以很快验证某笔交易是否属于某个区块。

##### 3 构建基于工作量证明的共识机制

在区块链系统中，由于交易数据是全网广播的，网络上任何一个节点但都可以对交易打包形成区块，但存在的问题是如何对不同节点所打包的区块形成共识。比特币系统中采用了PoW的工作量证明机制，以确定谁将获得最终的区块打包权。PoW是指，在对交易打包前，网络节点必须先计算一个工作量证明函数，当这个函数得到符合条件的解之后才能对交易打包形成区块。这个工作量证明函数，通常是某一种哈希函数，因为哈希函数具备输出值长度固定、输入和输出近似一一对应、无法逆推等特点，非常适合工作量证明。

##### 4 比特币钱包

在比特币交易中，如图4-19所示的信息是大家都能看到的信息，左上角是交易号码，箭头连接的两个字母和数字组成的字符串是比特币地址，表明比特币在两个地址之间实现了转移。而这个地址的生成是由钱包的公钥经过哈希函数转换而生成的，其中公钥是由随机数字构成的私钥通过非对称加密形成的。交易时公钥和比特币地址都需要公开发布，以便区块链系统能够验证付款交易的有效性。

图4-19 比特币转账示意图

4.4 认证方案对比

认证方案主要应用于交易的签名验签，技术底层为数字签名算法。本节从我国自主研发的SM系列密码算法和国际标准算法的安全性，性能，时间空间开销等方面，展开对比分析，对比对象选取为SM3与SHA256哈希算法，SM2和ECC-256非对称加密算法。

4.4.1 SM3与SHA-256

SM3与SHA-256都是哈希算法，具有类似的结构与流程。例如:两者数据最大输入均为264位，输出为256位的哈希值;消息分组均为512位，每个分组采用64轮循环迭代，且压缩函数具有相同的结构计算流程基本一致。 SM3的优势在于将消息扩展为132个字，相较于SHA-256 算法将消息扩展为64个字，位之间相关性更强，拥有更优良的抗碰撞性能。

4.4.2 SM2与ECC-256

SM2是ECC椭圆曲线算法的一种，其私钥长度为256位。主要分为签名验签、加密解密、密钥交换3个功能模块。表4显示了SM2与ECC-256签名算法的不同之处。

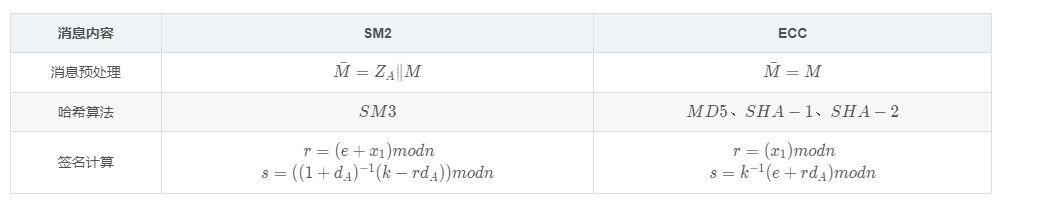


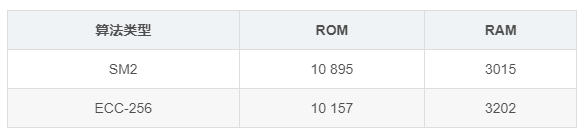
表4 SM2余ECC-256算法差异

可以看出，SM2与ECC-256的签名算法主要差别是计算消息M的哈希值e时，需要额外计算一个值。Z长度为256位，是根据签名者自己的、椭圆曲线参数、自身公钥等信息的哈希值，使得签名拥有更全面的用户信息，签名值拥有更强的效力，最终提高签名的可信性。

4.4.3 性能差异

选取长度为32、64、128字节的字符串，分别通过SM2、ECC算法完成签名验签操作，重复10000次，计算签名验签的时间开销与内存开销。

表5 SM2和ECC-256签名验签时间开销（单位：S）

表6 SM2和ECC-256签名验签内存开销（单位：Byte）

由表5可看出，SM2签名验签算法具有更快的执行速度，相较于比特币中采用的secp256k1曲线，时间节约了大概20%。由表6可知，SM2+SM3实现签名验签比ECC-256+SHA-256算法占用的内存更少，节约内存大约10%。

综上，SM2签名验签算法相对比特币中采用的ECC-256在同等安全性下，具有更高的效率，更低的运行开销。结合网络货运千万物流数据场景，探索轻量级认证算法对于项目的应用有必要的。此外，Hyperledger Fabric模块化的设计，可帮助我们后期扩展SM国密系列以符合国家区块链服务应用要求，并灵活接入其他轻量级算法。

# 五、高性能共识技术

5.1 共识机制概述

在人类社会发展的任何时期，共识理论都是一个涉及多个学科的复杂理论体系。从范围来看， 共识可以划分为国家共识、民族共识、群体共识等诸 多层面；从内容来看，共识又可以被划分为政治共识、经济共识和文化共识等诸多领域。

共识机制就是共识的达成机制，是共识主体、客体和中介等的关系和运行方式，是推动内在对象达 成统一共识的主要推动。常见的共识机制可以分为 内在互动机制和外在引导机制。 内在互动机制的达成需要良好的共识环境和理 性的共识行为，只有在一个相对平等和可信赖的环 境中，多元主体才能就某一问题协商，从而达成一个相对客观公正的共识；外在引导机制可以通过利益 整合机制、教育引导机制和制度保障机制等方式实现，通过利益满足和调节、刚性的约束等外在方式促成共识的达成。

### 5.1.1 区块链中的共识机制

在区块链中、为了获得系统给予的奖励，区块链网络中的每个节点都试图向链上添加区块，然而如果每个节点都添加区块，那么一笔交易可能被包含在多个区块中，同时每个节点添加的区块也不完全相同，造成节点之间维护的区块链数据不一致，最终导致整个系统失效。为了解决这个问题，区块链系统使用各种共识算法使得网络中互不信任的节点一起工作，并对添加到链上的数据达成一致性意见。

我们把分布式系统中的通信实体叫做节点，共识算法的目标就是协调各节点的动作，使得他们能够对系统中的共享值达成一致的状态。在分布式系统中，每节点一开始处于未决（Undecided）状态，然后每个节点提议了一个值，接着节点之间互相通信，最终每个节点设置了一个决定值而处于决定（Decided）状态。共识算法则确保各节点进入决定状态时拥有相同的决定值，把共识算法放在区块链的场景下，那么其必须满足以下性质：

* 终止性：每个节点最终提交一个区块。
* 一致性：每个节点提交的区块相同。
* 有效性：每个最终提交的区块都是由节点合法创建的。

有时也用活性（Liveness）和安全性（Safety）来描述区块链中的共识算法。安全性是一致性和有效性的统称，是指各节点提交的区块是完全相同的并且该区块是由节点合法创建的，即每个节点都保存有相同的正确数据。在区块链的语义下，指的是不会分叉。活性等同于终止性，是指节点不会一直处于停滞状态而丧失了提交区块的能力，即节点提出的合法交易最终会被记录在区块内。

### 5.1.2 FLP定理和CAP定理

#### 5.1.2.1 FLP三角不可能定理

1985 年， 由 Michael Fisher、Nancy Lynch 和Michael Paterson 共同提出并证明了在分布式系统共识算法的设计中起到重要指导作用的“FLP 不可能定理”[11]。该定理指出: 在网络可靠的异步通信系统中， 当存在节点故障(即使只有一个)的情况下， 不存在协议能保证在有限时间内使系统达成一致。

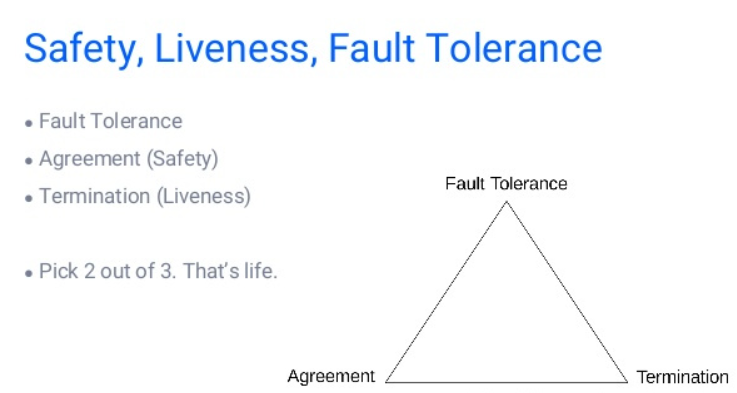


图5-1 FLP三角不可能定理

从另一个角度来理解，FLP如图5-1所示，实际上考虑了分布式系统的3个属性：安全(safety)、活性（liveness)、容错。

* 安全：系统内各个节点达成的值是一致的、有效的。safety其实是保证系统一致性运行的最低要求，其核心是cannot do something bad，即不能干坏事、不能做错事。
* 活性：系统内各个节点最终（在有限时间内）必须能够达成一致，即系统必须能够向前推进，不能永远处于达不成一致的状态。liveness其实是更高要求，意味着不能只是不干坏事，也不能一直不干事，you must do something good，即必须使得整个系统能良好运转下去。
* 容错：该协议在有节点故障的情况下也必须能有效。

“FLP 不可能定理”指出了在可能存在节点失效的分布式异步通信系统中， 理论上不存在能使系统在有限时间内达成一致的共识算法。根据FLP Impossibility原理，在asynchrony假定下，共识算法不可能同时满足safety和liveness。为了设计能够符合实际场景的共识算法，目前的BFT类共识算法多是基于partial synchrony假定，这在PBFT论文中被称为"weak synchrony"，同样在比特币系统中假定网络为弱同步性，即网络节点间可以快速同步。

#### 5.1.2.2 CAP不定理

2000 年， Eric Brewer 在一次研讨会的报告中提出了一个猜想: 分布式系统无法同时满足一致性 (Consistency)、可用性(Availability)和分区容忍性(Partition Tolerance)， 最多只能同时满足其中两个特性， 如下图5-2所示。该猜想于2002 年被 Seth Gilbert 和 Nancy Lynch 在异步网络型证明， 被命名为“CAP 定理”[12]。一致性是指分布式系统中所有节点在同一时刻持有同样的数据信息; 可用性是指系统处于服务状态， 当客户端发出请求， 服务端能在有效的时间内返回结果; 分区容忍性是指允许网络中部分节点不与其他节点通信， 即允许网络发生分区(不同区域之间的节点不能建立通信)。

CAP与FLP看起来有相似之处，其实二者并不尽相同，二者是从不同的维度思考问题，另外即使是很相似的概念，内涵也并不完全一样。比如：

* FLP面对的是分布式一致性问题，而CAP面对的是分布式网络中的数据同步与复制。
* FLP是说在异步网络模型中，三者不可能同时实现；而CAP是说在所有场景下，三者都不可能同时实现。
* FLP中的liveness强调的是一致性算法的内在属性；而CAP中的availability强调的是一致性算法对外呈现的外在属性。

“CAP 定理”指出即使可以设计出工程上可行的共识算法， 这个共识算法也无法完美地做到同时满足一致性、可用性和分区容忍性。该定理的出为共识算法的设计提供了指导性的原则， 使研究者不再追求能同时满足三个特性的共识算法。

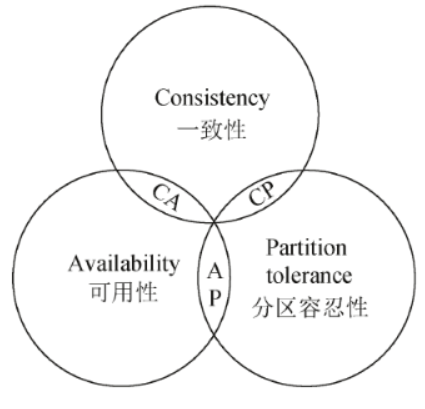


图5-2 CAP定理

### 5.1.3 共识机制中的拜占庭问题

拜占庭将军问题与共识拜占庭将军问题是分布式系统中节点达成共识的经典问题，最早由 Lapmport等在1982年提出。拜占庭将军问题描述为:在宽广的古代东罗马都城，需要多个将军守卫边境，然而将军之间需要通过信使来传递消息，从而达成某些一致的决定。但是将军中可能存在叛徒，其中叛徒可以向其他将军传递不同的消息，试图干扰一致性的达成。在此情况下，如何让忠诚的将军们能达成行动的一致。

若分布式系统中存在恶意的计算机节点，这些节点会选择性响应某些请求或 篡改系统中的数据，在不可靠的信道上，系统中所有非恶意的节点如何通过消息传递的方式达成共识。

消息在不可靠的信道上试图通过消息传递的方式达成共识是不可能的。进程间的通信是无法达到完全同步的，节点也无法保证持续在线。这为共识算法的设计提出了要求: 工程上，节点间须采用非对称加密算法对消息进行签名保证消息可靠传递，设置消息最大时延(即在一个最大时间限制 Tmax内，若消息的接收方依旧没有收到γ节点发送的ξ消息， 就判定γ节点失效且ξ 消息超时，系统将不再等待和处理ξ 消息来解决 FLP不可能问题。同时存在恶意节点的环境下要 达成共识，恶意(拜占庭/非拜占庭)计算机节点的比例不可超 过系统中计算机节点总数量的１/３，才能保证系统的安全性 (Safety)与活性(Liveness)。

通常，我们将选择性只响应某些节点请求、对系统中数据做出伪造或篡改等破坏性行为的计算机节点称为拜占庭错误 (ByzantineFault)节点，将发生宕机、网络异常导致消息延时、 丢失或重复等现象的计算机节点称为非拜占庭错误(Crash Fault)节点。

### 5.1.4 共识算法的分类方法

从数据一致性的角度，可以将共识算法分为强一致性共识算法、弱一致性(最终一致性)；共识算法强一致性指各节点在同一时间（段）处于一致状态，而弱一致性则允许各节点的状态出现短暂的不一致，但最终趋于一致。区块分叉就是短暂不一致所导致的结果，但最终各节点会保留唯一的主链，也就是最终趋于一致。典型的强一致性算法有：Raft、PBFT；典型的弱一致性算法有POW、POS。

从拜占庭容错的角度，可以将共识算法分为拜占庭容错共识算法、非拜占庭容错共识算法（故障容错共识算法）。拜占庭节点是系统中出现的最坏节点类型，但并不是所有的系统中都有该类节点。在一些可信度较高的场景中，通常不考虑拜占庭节点的存在，而只考虑故障节点。典型的拜占庭类共识算法有：PBFT、Hot-stuff；典型的非拜占庭共识算法（故障容错共识算法）有：Paxos、Raft。

从应用角度，可以将共识算法分为适用于公链、联盟链、私链的共识算法。联盟链和私链属于授权网络中，其节点一般通过公钥基础设施(public key infrastructure， PKI)完成身份认证后，才能参与后续共识机制；公有链则是基于非授权网络。非授权网络中，节点随时加入和退岀，节点数量动态变化且不可预知。典型的公有链共识算法有：POW、POS;典型的联盟链和私有链的共识算法有：Raft、PBFT。

从共识过程出发， 按照主节点的产生方式可以将共识算法分为竞争类、选举类、随机类以及其他类型：竞争类: 所有参与共识的节点通过参与特定的竞争方式， 在竞争中获胜的节点成为主节点。竞争类共识算法以 PoW 为代表， 还有权益证明、空间证明等等。选举类: 所有节点通过投票的方式选择出部分节点参与共识过程， 这些被选择的节点依次轮流成为主节点。选举类共识算法以区块链中应用的传统 BFT类共识算法为代表， 比如 PBFT、DBFT 等。随机类: 通过设计特定的随机算法从参与共识的节点中随机选择节点作为主节点。随机类共识算法以 PoET、Algorand、Ouroboros 为代表，其他类：比如基于分片的共识算法（Elastico）、基于DAG有向无环图的共识算法。

### 5.1.5 共识算法的评判标准

共识算法的性能高低直接影响到区块链的上层应用，而评判一个共识算法的性能主要参考以下几点：

1. 容错率：区块链系统最多能够容忍多少故障节点，或者容忍多少算力（权益）属于故障节点。
2. 吞吐量：区块链系统处理交易的速度，用每秒交易数（Transaction Per Second，T PS）来衡量，通常指最大吞吐量。
3. 交易确认延迟：区块链系统中的交易从创建到最终安全确认的时间间隔。
4. 资源消耗
   1. 算力开销：共识过程中需要不停地进行哈希运算，主要指 POW 类共识。
   2. 通信开销：共识过程中各节点互相通信的次数。
   3. 计算开销：共识过程中各节点互相通信时对消息签名或者验签的次数。

5.2 常见基于公有链的共识算法

### 5.2.1 POW共识算法

PoW 共识算法中， 所有参与共识的节点通过消 耗算力解决数学难题来竞争成为主节点， 由最先解 决数学难题的节点成为主节点。在比特币系统中， 中本聪采用安全散列算法， 将数学难题设计为对区块 头信息的双 SHA256 哈希运算， 即:

F（BlockHeader | Nonce ）< T

其中 F 为双 SHA256 哈希运算函数[68]， T 是目标哈 希值， BlockHeader 为区块头信息， 其中包含 Nonce 字 段 ， 求解合适的 Nonce 字段使得对完整 BlockHeader 的哈希运算结果满足目标值要求。 共识过程：

第一步，全网节点接收并 转发交易信息， 将接收到的交易信息打包进新区块， 然后全网所有节点基于本节点打包的新区块设置区 块头中的 Nonce 字段， 计算整个区块头信息的哈希 值， 使其满足目标值的要求; 率先完成计算工作的 节点成为主节点， 并将自己的区块广播给其余节点。

第二步， 未竞争成功的其余节点在接收到新区块后， 放弃计算本地新区块， 并对接收到的新区块进行验证。

第三步， 节点在验证通过新区块后， 将其添加到 本地区块链， 所有节点达成对最新高度区块的共识。

### 5.2.2 POS共识算法

Peercoin 项目中实现的 PoS 共识算法是为了 降低算力的消耗。PoS 延续了 PoW 算法的竞争理念， 只不过相对于PoW中 Nonce 字段的大搜索空间而言， PoS 将搜索空间限制在一个计算量可接受的范围; 除此外， PoS 中还引入了“币龄”作为权益， 即:

Coinage = Coin \* Age

其中， Coinage 是“币龄”， Coin 为持有的货币， Age 为持续持有的时间。通过减小搜索空间以及引入“币 龄”， PoS 将数学问题设计为:

F（BlockHeader | Timestamp ）< T\*Weight

其中 F 为双 SHA256 哈希运算函数; BlockHeader 为 区块头信息， 其中包含Timestamp 字段， 取值范围是 上一个区块时间和当前时间之间， 远小于 PoW 中 Nonce 字段的搜索空间大小; Weight 是用于竞争所 消耗的“币龄”权重， T 是目标哈希值， 与 PoW 中的T 相同。PoS 的共识过程与 PoW 一致， 唯一不同是解决的数学问题不同，共识过程和POW一致。与 PoW 相比， PoS 的数学问题中自变量的搜索 空间减小， 同时不等式右侧引入“币龄”权重， 对于同一T ， 在每轮竞争中所投入的“币龄”越多， 权重越大， 竞争中获胜的概率也越大。

### 5.2.3 DPOS共识算法

以比特币为代表的区块链应用采用了 DPOS 共识算法，其完全避免了算力的消耗，并且极大地提高了共识效率。与 POW 和 POS 不同，DPOS 不属于证明类共识，而是属于选举类共识。在 DPOS 中，各节点基于自身拥有的权益（代币）产生投票，用于选举出若干名见证人，这部分见证人享有记账权，负责轮流生成新的区块[48]。具体共识流程如下：

1) 选举出块者。权益持有者投票选举见证人。见证人在系统中保持中立，每 24 小时更新一次，其收益由权益持有者投票决定。Bitshare 可以有任意数量的见证人，EOS每轮见证人的数量设置为21，票数最多的前 20 人自动当选见证人，剩余 1 人随机选出，见证人需要 100% 在线。

2) 提出区块。见证人产生区块并对区块签名和添加时间戳，再在系统中广播新产生的区块，如果见证人在某个时隙没有产生区块，则该时隙将被跳过，产生区块的权力交给下一个见证人。Bitshare 中每 2 秒产生一个区块，EOS 中每 3 秒产生一个区块。

3) 验证区块并更新区块链。其他见证人负责验证新产生区块的合法性。Bitshare 中 N 个见证人验证通过就可上链( N 个见证人要求代表的投票权之和大于 50% ) ，EOS 中半数以上的见证人验证通过就可以上链。

在 POW 和 POS 中，全网节点都可参与共识。而 DPOS 增加了一个选举功能，被选举出来的见证人代表其余节点完成整个系统的共识过程。记账权只属于被选举出来的一小部分见证人，他们只需要轮流出块而无需进行挖矿，因此系统的共识效率得到了较大提高并且彻底解决了算力消耗的问题。但 DPOS 使得权益越多的节点拥有更大的话语权，这在一定程度上脱离了去中心的思想，也削弱了系统的去中心化程度。

### 5.2.4 小结

公有链因为网络无权限控制，完全的开放、透明、民主，所以参与的节点数目很可能非常众多，节点作恶的可能性也很高。在这种无许可的网络中、节点数目众多、存在较大作恶可能的分布式网络环境中，显然传统的分布式一致算法并不适用，我们需要寻求新的解法。公有链的共识机制不再拘泥于分布式算法层面，而是包含了更多经济学、博弈论、概率论等思想。

其中POW和POS属于工作量证明类共识算法，DPOS属于权益选举类共识算法。DPOS算法满足“区块最终一致性”。也就是一旦区块被大多数节点验证成功，那么它就会在主链上不会出现分叉。再基于工作量证明的共识机制中，一般都不满足强共识性质。一旦一个矿工的区块被发布，它就被添加到区块链中，并且信息被广播。由于网络效应，会存在多个矿工区块同时发布的情况。也有可能，一些矿工在不同意最新的区块的情况下提出其他区块，这个时候会产出分叉的情况。

所以，工作量证明共识通过要求矿工在最长的分支上工作来实现最终的区块链共识，即要求以最长的区块链认定为主链。因此，即使每次已经验证了区块，我们也不能确定它之后是否仍然在主链上。在比特币中，用户通常等待6个区块的确认后才认为交易(即其区块)有效，即being6×10min= 1h。从这个意义上说，交易永远不会被绝对接受，只有随着链的增长而指数下降的逆转概率。

5.3 基于联盟链的共识算法

目前区块链共识可以分为两大类， —类是基于授权网络的共识机制，授权网络中节点一般通过公钥基础设施(public key infrastructure， PKI)完成身份认证后，才能参与后续共识机制；另一类是以比特币为代表的基于非授权网络中的共识机制。

在授权区块链中，存在角色的明确分离。矿工总是众所周知的，而且是预先确定的。在其他用户可能拥有的功能方面存在差异。对于某些系统，成员资格是开放的，但节点只能发送和接收事务。对于其他用户，每个用户必须收到成为节点的特殊邀请。在这样的系统中，所有用户都是事先知道和识别的。由于这些特点，此类系统在文献中被称为私有或联盟链。

根据共识算法是否能够处理拜占庭场景，授权网络下的共识机制又可以分为故障容错（CFT）类共识算法和 拜占庭容错类（BFT）共识算法。 前者能够处理网络故障、断电、宕机等错误，后者则能够处理拜占庭错误。 一般地，前者往往有着更强的性能，后者则有着更多的适应场景。

### 5.3.1 故障容错协议

#### 5.3.1.1 Paxos

Paxos不是一个单独的协议，而是一整套协议。基本系统假设如下：处理器以任意速度执行操作；在工作期间，他们可能会遇到碰撞故障；恢复后，处理器可以重新加入协议；在操作过程中，处理器不会试图以任何方式欺骗系统。在通信方面，一个处理器可以向任何其他处理器发送消息，这需要任意长的时间才能发送。消息可能会丢失，但不会被第三方修改。

Paxos在n = 2/+1模型中，能够容忍/个崩溃节点，实现了基于消息传递的一致性算法。Paxos中 提出了主节点、备份节点的概念，其主要过程如下：

1. 主节点向全网超过 1/2的备份节点发送准备(prepare)消息；
2. 备份节点验证消息合法性，通过后 向主节点返回承诺(promise)消息；
3. 主节点收集足够多的承诺消息，组成承诺凭证，主节点向备份节点 发送包含承诺凭证的接受(accept)消息；
4. 备份节点验证接受消息的合法性，通过后向主节点返回己接受 (accepted)消息。

Paxos允许多个主节点提议，并对主节点赋予不同的等级，等级高的主节点的提议能够打断等级低的 主节点提议，即使等级低的主节点提议己经得到备份节点的承诺消息。Paxos协议被用于分布式系统中数 据库的维护，只能对崩溃节点容错，不能对拜占庭节点实现容错。

#### 5.3.2.2 Raft

Raft是Ongaro& Ousterhout（2014）提出的具有碰撞故障恢复能力的共识协议。它是对Paxos算法的一种改进，是从多副本状态机的角度提出，通过管理多副本状态机的日志复制来实现共识。Raft 算法中有三种角色，分别是领导者( Leader) 、 跟从者( Follower) 、候选人( Candidate) 。其中领导 者主要用于传达客户端的信息，通知跟从者日志的 同步，在收到大部分跟从者确认后，提交日志。而跟从者则是接受并持久化日志，在领导者确认之后 进行提交。候选人就是节点从跟从者到领导者的 中间角色。该算法的容错能力与 Paxos 算法相同， 为 2f + 1 ，但是比 Paxos 更容易理解和实现。

Raft 算法的关键问题是领导者选举( Leader Election) 和日志同步( Log Ｒeplication) 。Ｒaft 通过心跳触发系统进行领导人的选举。选举过程分为两步: 首先跟从者将自己的任期期号加一，并将角 色转变为候选者; 其次向其他节点发送投票请求， 希望跟从者能给其投票。如果某个候选人得票超 过半数，就可成为下一任期的领导人。如一段时间内，没有节点获胜，即多个领导者的票数相同，则候 选人继续增加任期号，并重启选举。日志同步则是 由客户端发送请求给领导者，再由领导者进行发布 确保其和其它跟从者之间的状态是一致的。

### 5.3.2 拜占庭容错协议

#### 5.3.2.1 PBFT

PBFT（Practical Byzantine Fault Tolerance），即实用拜占庭容错算法，由Miguel Castro和 Barbara Liskov在1999年发表的论文《Practical Byzantine Fault Tolerance and Proactive Recovery》中提出。为了防止误传和混淆，每个节点都使用自己的密钥对消息进行签名。此外，每条消息都有一个身份验证码，发送时，使用哈希函数对其进行压缩。每个节点都与系统中的所有其他节点通信。节点可以根据签名相互识别，并检查消息在传输过程中是否发生了更改。

该共识算法从全网节点选择一个主节点负责创建区块后经过三阶段投票达成共识: 预准备阶段， 准备阶段， 提交阶段。主要过程如下:

1. 预准备阶段（pre-prepare）: 从全网选择一个主节点; 每个节点把客户端发来的交易信息向全网广播， 主节点收集所有交易信息， 创建新区块并向全网广播;
2. 准备阶段(prepare)： 每个节点在收到主节点发送的区块信息后， 从预准备阶段进入到准备阶段， 节点对区块进行验证， 验证通过后向其他节点广播一条准备消息;
3. 提交阶段(commit): 节点在向全网广播准备消息之后进入提交阶段， 如果节点收到超过 2/3 节点的准备消息， 就向全网广播一条提交消息; 如果一个节点收到超过 2/3 节点的提交消息， 即可提交新区块到本地区块链， 达成对最新高度区块的共识。

如果领导节点出现故障或者错误的时候，导致了数据无法处理，就需要启动视图转换。这个过程被称为视图转换( View－Change) ，每当需要在共识节点中切换Leader时，都需要大量的消息传递，复杂度为O(n2)。

#### 5.3.2.2 SBFT

可扩展拜占庭容错协议(scalable Byzantine fault tolerance， SBFT)由Golan-Gueta等人提出。 SBFT主要解决拜占庭容错协议在应用到区块链中的去中心化和扩容问题，SBFT的敌手模型n = 3/+1， 能够允许200多个节点同时完成共识。SBFT利用收集器(collector)作为信息、签名的收集者，采用门限签名降低通信复杂度。在PBFT中，每一轮的投票，节点需要向网络中其他节点广播投票，并且收集其他节点的投票， 而SBFT利用收集器(collector)收集每一轮投票，收集到的签名数量达到门限要求以后，收集器(collector)便能恢复总的门限签名，从而将通信复杂度从O(n2)降低为O(n)。视图更改协议依旧和PBFT一致。

注：门限签名：对于一个(k， n)-门限签名方案，假定存在一个公钥，而 n 个签名者每人都拥有自己的私钥（分片）。只要其中至少 k 个签名者对消息进行部分签名，那么由这 k 个部分签名可以导出对消息的完整签名，并且这个完整签名可以由公钥来验签。

#### 5.3.2.3 Tendermint

Tendermint由2014年提出，协议中的参与者叫「验证者」 (Validator），对于主节点的选择，每个节点都会保留一份Validator集合，每一轮会从集合中选择一个作为Proposer（出块节点）。对于区块的确定，Tendermint是基于round（回合）的机制来确定的。每个round又三个过程组成：propose（提议），prevote（预投票）和 precommit（预提交）。Tendermint的propose->pre-vote->pre-commit三个阶段，跟PBFT的三个阶段，pre-prepare， prepare， commit 三阶段是一一对应的。

Tendermint采用了锁定、解锁机制简化了PBFT的View Change阶段，Tendermint将视图切换的流程合并到了正常的投票流程，即“复用”了投票信息，用于判断系统是否要进行视图变更。proposer超时的时候新块是一个特殊的空块，切换proposer是通过提交commit空块来触发的，而PBFT是有一个单独的view change过程来触发primary轮换。 在主节点更换过程中，tendermint并没有传递prepared状态信息。为了保证safety特性，tendermint中新的轮次中，根据本地节点是否有锁定的信息来进行，而锁定的信息就是prepared状态。所以，tendermint也是在本节点上一轮prepared信息的基础上继续进行共识。

除了简化View Change这一点，Tendermint的所有信息都存储在blockchain里。因为PBFT是1999年提出来的，那时候还没有blockchain的概念(blockchain是2009年比特币出现之后才有的)，因此 PBFT的所有节点虽有有一致的数据，但数据是分散存放的。

#### 5.3.2.4 Hot-Stuff

HotStuff算法由 Abraham，Gueta 和 Malkhi 提出。首先，HotStuff引入了门限签名，每一轮的共识投票消息，都是各个共识节点发送给Leader节点，由Leader将这些消息签名组合成一个，再广播给大家。这样极大的较少了系统中消息量，从O(n2)减到了O(n)。

然后相比于PBFT和Tendermint的两轮投票，HotStuff采用了三轮投票，多了一轮投票，各个节点集齐投票就可以进入下一个共识，而不需要等待固定的时间。

最后该算法采用链式确认，将原先需要两轮通信才能达成共识写入链中的方式，改成了先将区块上链。只要一个区块后面产生了三个连续区块，那么就说 明该区块经过了三轮投票确认，就可以最终确认该 提前上链区块成功出块了。这种方式使得系统的 响应特性大幅度提高了。

#### 5.3.2.5 小结

上述四种协议都可以防御n=3f+1的拜占庭容错，同时都应用状态机复制技术进行共识。它们的性能有以下图5-3所示的区别：

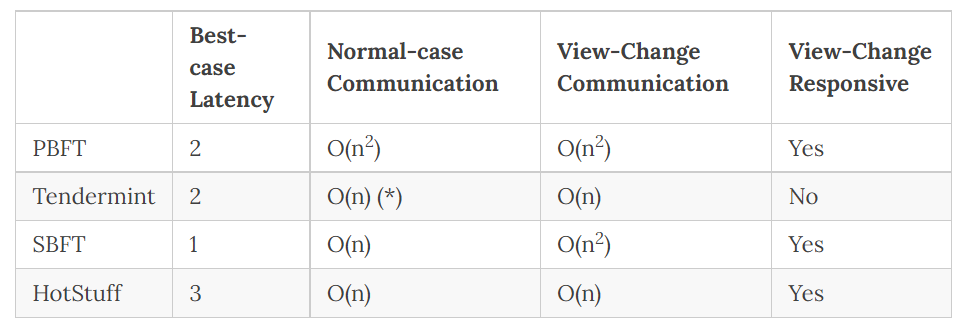


图5-3 拜占庭容错算法对比

5.4 联盟链主流共识算法详解

### 5.4.1 典型故障容错共识算法Raft

#### 5.4.1.1 共识过程

Raft的共识过程主要分为两个阶段，首先是leader选举过程，然后在选举出来的leader基础上进行正常操作，比如日志复制、记账等。

#### 5.4.1.2 领导者选举

在Raft集群中如下图5-4，每一个节点任意时刻都处于以下三个状态之一：leader （领导者）、follower（跟随者）、candidate（候选者）。正常状态下，集群中只有一个leader，其余全部都是follower。follower是被动的，它们不会主动发出消息，只是响应leader或candidate的消息，或者转发客户端的请求给leader。candidate是一种临时的状态。

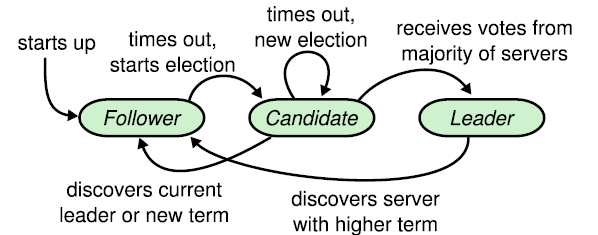


图5-4 Raft节点角色

根据 Raft 协议，一个应用 Raft 协议的集群在刚启动时，所有节点状态都是Follower态，由于没有Leader，Follower 无法与 Leader 保持心跳（heart beat），Follower等待心跳超时（每个Follower的心跳超时时间不一样），Followers 会认为 Leader 已经 down 掉。最先超时的Follower进而转为 Candidate 状态，然后，Candidate 将向集群中的其它节点请求投票，同意自己升级为 Leader，如果 Candidate 收到超过半数节点的投票（N/2+1），它将获胜成为 Leader。角色选举详细流程如下：

**第一阶段：都是 Follower 状态**

一个应用Raft协议的集群在刚开始启动时（或者 Leader 宕机重启时），所有的节点都是 Follower 状态，[初始任期](file:///E:\Home\我的大学\毕业设计\user_cancel)（Term，即某次选举的唯一标识）都是0。同时启动选举定时器，每个节点的选举定时器都不一致且都在100～500ms之间（避免同时发起选举）。

Term可以理解为任期，用Term作为一个周期，每个Term都是一个连续递增的编号，每一轮选举都是一个Term周期，在一个Term中只能产生一个Leader 同时，term也起到了系统中逻辑时钟的作用，每一个server都存储了当前term编号，在server之间进行交流的时候就会带有该编号，如果一个server的编号小于另一个的，那么它会将自己的编号更新为较大的那一个；如果leader或者candidate发现自己的编号不是最新的了，就会自动转变为follower；如果接收到的请求的term编号小于自己的当前term将会拒绝执行。

**第二阶段：从 Follower 状态转换为Candidate，并发起投票**

由于没有 Leader，Followers 无法与 Leader 保持心跳（heart beat），节点启动后在一个选举定时器周期内未收到心跳和投票请求，则状态转变为 Candidate 状态、Term 自增，并向集群中所有节点发送投票请求并且重置选举定时器。

注意：每个节点选举定时器超时时间都在 100 ～ 500 ms之内，且不一致。因此，可以避免所有的Follower同时转化为 Candidate状态，换言之，最先转为 Candidate 并发起投票请求的节点将具有成为 Leader 的先发优势。

**第三阶段：Follower进行投票**

Follower 节点收到投票请求后会根据以下情况决定是否接受投票请求：

请求节点的 Term 大于自己的 Term，且自己尚未投票给其它节点，则接受请求，把票投给 Candidate 节点；

请求节点的 Term 小于自己的 Term，且自己尚未投票，则拒绝请求，将票投给自己。

**第四阶段：Candidate 转换为 Leader**

经过一轮选举后，正常情况，会有一个 Candidate 节点收到超过半数（N/2+1）其它节点的投票，那么它将胜出并升级为 Leader 节点，然后定时发送心跳给其它节点，其它节点会转化为 Follower 节点并与 Leader 保持同步，如此，本轮选举结束。如果一轮选举中，Candidate 节点收到的投票没有超过半数，那么将进行下一轮选举。

#### 5.4.1.3 日志复制

一个 Raft 集群中只有 Leader 节点能够处理客户端的请求（如果客户端的请求发到了 Follower 节点，Follower 将会把请求重定向到 Leader），客户端的每一个请求都包含一条被复制到状态机执行的指令。Leader 把这条指令作为一条新的日志条目（Entry）附加到日志中去，然后并行的将附加条目发送给 Followers，让它们复制这条日志条目。当这条日志条目被 Followers 安全的复制，Leader 会应用这条日志条目到它的状态机中，然后把执行的结果返回给客户端。如果 Follower 崩溃或者运行缓慢，再或者是网络丢包，Leader 会不断的重复尝试附加日志条目（尽管已经回复了客户端）直到所有的 Follower 最终都存储了所有的日志条目，确保强一致性。日志复制详细流程如下：

**第一阶段：客户端请求提交到 Leader**

Leader 收到客户端请求：如存储一个数据：5；Leader 收到请求后，会将它作为日志条目（Entry）写入本地日志中。此时该 Entry 是未提交状态（uncommitted），Leader并不会更新本地数据，因此它是不可读的。

**第二阶段：Leader 将 Entry 发送到其它Follower**

Leader 与 Followers 之间保持心跳联系，跟心跳 Leader 将追加的 Entry（AppendEntries） 并行的发送到其它 Follower 节点，并让它们复制这条日志条目，这一过程我们称为：复制（Replication）。

**第三阶段：Leader 等待 Followers 回应**

Followers 接收到 Leader 发来的复制请求后，有两种可能的回应：

* 写入本地日志，返回 Success
* [一致性检查](file:///E:\Home\我的大学\毕业设计\user_cancel)失败，拒绝写入，返回 false。原因和解决办法上面已经详细说明。

注：此时该 Entry 的状态也是未提交（uncommitted）。完成上述步骤后，Followers 会向 Leader 发出回应 - success，当 Leader 收到大多数 Followers 的回应后，会将第一阶段写入的 Entry 标记为提交状态（committed），并把这条日志条目应用到它的状态机中。

**第四阶段：Leader回应客户端**

完成前三个阶段后，Leader 会回应客户端 - OK，写操作成功。

**第五阶段：Leader 通知 Followers Entry 已提交**

Leader 回应客户端后，将随着下一个心跳通知 Followers，Followers 收到通知后也会将 Entry 标记为提交状态。至此，Raft 集群超过半数节点已经达到一致状态，可以确保强一致性。需要注意的是，由于网络、性能、故障等各种原因导致的“反应慢”、“不一致”等问题的节点，也会最终与 Leader 达成一致。

#### 5.4.1.4 一致性安全保证

前面描述了 Raft 算法是如何选举 Leader 和 日志复制。然而，到目前为止描述的机制并不能充分保证每一个状态机会按照相同的顺序执行相同的指令。例如：一个 Follower 可能处于不可用的状态时， Leader 已经提交了若干的日志条目，然后这个 Follower 恢复（尚未与 Leader 达成一致）后被选举成为新的Leader时，就会出现问题：不同的状态机执行不同的指令序列。

鉴于此，在 Leader 选举的时候需要增加一些限制来完善 Raft 算法。这些限制可保证任何的 Leader 对于给定的任期号（Term），都拥有之前任期的所有被提交的日志条目（所谓 Leader 的完整特性）。

##### 1 选举安全性

选举安全性，即任一任期内最多一个leader被选出。这一点非常重要，在一个复制集中任何时刻只能有一个leader。系统中同时有多余一个leader，被称之为脑裂（brain split），这是非常严重的问题，会导致数据的覆盖丢失。因此，某一任期内一定只有一个leader。在raft中，两点保证了这个属性：

* 一个节点某一任期内最多只能投一票
* 只有获得大多数投票的节点才会成为leader

##### 2 日志匹配

当出现了leader与follower不一致的情况，leader强制follower复制自己的log。为了使得followers的log entries和leader的保持同步，leader必须找到其与每一个follower的日志条目相同的位置，并将follower在这个位置之后的所有日志条目删除掉，然后把自己在这个位置之后的所有日志条目发送给follower。所有这些操作发生在AppendEntires rpcs（追加日志请求）的一致性检查之后。

leader在接收客户端请求后，会发送AppendEntires rpcs给各个follower，其中包含的一致性检查具体为：Aleader为每一个follower维护一个nextindex，这个index是leader将要发送给follower的下一个日志条目的index。当一个leader第一次掌权时，它初始化所有的nextindex为自己的下一个log entry的index。如果一个follower的日志与leader的不符合，那么AE一致性检查将在下一个AE rpcs中返回失败。在rejection之后，leader将next index-1，然后重试AE rpc。最终nextindex将会退回到一个leader和follower的日志完全匹配的位置上。然后AE 将会成功，然后移除和leader有冲突的日志条目，并且追加leader的对应index的日志条目。一旦AE成功，follower的log将会与leader的保持一致，并且在这个term剩下的时间内保持这样的情况。

##### 3 领导者日志完整性

Raft 使用投票的方式来阻止一个没有包含所有已经提交的日志条目的 Candidate 赢得选举。Candiate 为了赢得选举必须联系集群中的大部分节点，这意味着每一个已经提交的日志条目都在这些服务器节点中肯定存在于至少一个节点上。也就是至少有一个节点有最新的日志记录。如果 Candidate 的日志至少和大多数的服务器节点一样新，那么它一定持有了所有已经提交的日志条目。所以投票请求的限制为：然后投票人会拒绝那些日志没有自己日志新的投票请求（投票请求中包含了 Candidate 的日志信息）。

判断日志是否最新的方法如下：Raft 通过比较两份日志中最后一条日志条目的索引值和任期号，确定谁的日志比较新。如果两份日志最后的条目和任期号不同，那么任期号大的日志更加新一些。如果两份日志最后的任期号相同，那么日志比较长的那个就更加新。

### 5.4.2 典型拜占庭容错共识算法PBFT

#### 5.4.2.1 共识流程

PBFT 算法在共识过程中了防止误传和混淆，每个节点都使用自己的密钥对消息进行签名。此外，每条消息都有一个身份验证码，发送时，使用哈希函数对其进行压缩。每个节点都与系统中的所有其他节点通信。节点可以根据签名相互识别，并检查消息在传输过程中是否发生了更改。基本流程主要有以下四步：

1. 客户端发送请求给主节点
2. 主节点广播请求给其它节点，节点执行PBFT算法的三阶段共识流程。
3. 节点处理完三阶段流程后，返回消息给客户端。
4. 客户端收到来自 f+1 个节点的相同消息后，代表共识已经正确完成

##### 1 初始选主过程

PBFT 共识机制中每个节点都有一个唯一性编号标识( i) ，编号逐渐递增。一次共识从开始到结束所使用的数据集合称为视图，每个视图都由一个唯一性编号 v 标识，每次视图变更编号加 1，每个视图中只存在一个主节点，其他节点都为备份节点。主节点通过视图编号以及节点数集合来确定，按照编号轮流做主节点。即：主节点 p = v mod |R|。v：视图编号，|R|：节点个数，p：主节点编号。

##### 2 一致性协议

一致性协议用于处理客户端发出的日志请求，其能确保系统中各节点按同样的顺序处理同样的日志请求，继而产生相同的最终状态。一致性协议的核心过程由预准备（Preprepare）、准备（Prepare）和提交（Commit）阶段构成，而请求（Request）和回复（Reply）阶段则是用于客户端发出日志请求和接收日志执行结果。

(1) Request：客户端向主节点发出 < Request， o， t， C > 消息，申请执行日志请求。 o: 请求的具体操作，t: 请求时客户端追加的时间戳，c：客户端标识。REQUEST: 包含消息内容m，以及消息摘要d(m)。客户端对请求进行签名。

(2) Pre-prepare：主节点收到客户端的请求，需要进行校验：客户端请求消息签名是否正确。校验不通过丢弃，校验通过则分配一个编号n，编号n主要用于对客户端的请求进行排序。然后广播一条<<PRE-PREPARE， v， n， d>，  m>消息给其他副本节点。v：视图编号，d：客户端消息摘要，m：消息内容。<PRE-PREPARE， v， n， d>进行主节点签名。n是要在某一个范围区间内的[h， H]。

(3) Prepare：副本节点 i 收到主节点的PRE-PREPARE消息，需要进行以下校验：

a。 主节点PRE-PREPARE消息签名是否正确。

b。 当前副本节点是否已经收到了一条在同一v下并且编号也是n，但是签名不同的PRE-PREPARE信息。

c。 d与m的摘要是否一致。

d。 n是否在区间[h， H]内。

检验不通过丢弃，校验通过则向其他副本节点i向其他节点包括主节点发送一条<PREPARE， v， n， d， i>消息， v， n， d， m与上述PRE-PREPARE消息内容相同，i是当前副本节点编号。<PREPARE， v， n， d， i>进行副本节点i的签名。记录PRE-PREPARE和PREPARE消息到log中，用于View Change过程中恢复未完成的请求操作。prepared状态是十分重要的，当涉及到view转换时，为了保证view切换前后的safety特性，需要将上一轮view的信息传递到新的view，而在pbft中就是将prepared状态信息传递到新的view。可以这么理解，新的view中需要在上一轮view的prepared信息基础上，继续进行共识。

(4) Commit：主节点和副本节点收到PREPARE消息，需要进行以下校验：

a。 副本节点PREPARE消息签名是否正确。

b。 当前副本节点是否已经收到了同一视图v下的n。

c。 n是否在区间[h， H]内。

d。 d是否和当前已收到PRE-PPREPARE中的d相同

检验不通过丢弃，如果一个副本节点i收到了2f+1个验证通过的PREPARE消息，则向其他节点包括主节点发送一条<COMMIT， v， n， d， i>消息，v， n， d，  i与上述PREPARE消息内容相同。<COMMIT， v， n， d， i>进行副本节点i的签名。记录COMMIT消息到日志中，用于View Change过程中恢复未完成的请求操作。记录其他副本节点发送的PREPARE消息到log中。

(5) Reply：主节点和副本节点收到COMMIT消息，需要进行以下校验：

a。 副本节点COMMIT消息签名是否正确。

b。 当前副本节点是否已经收到了同一视图v下的n。

c。 d与m的摘要是否一致。

d。 n是否在区间[h， H]内。

检验不通过丢弃。如果副本节点i收到了2f+1个验证通过的COMMIT消息，说明当前网络中的大部分节点已经达成共识，运行客户端的请求操作o，并返回<REPLY， v， t， c， i， r>给客户端，r：是请求操作结果，客户端如果收到f+1个相同的REPLY消息，说明客户端发起的请求已经达成全网共识，否则客户端需要判断是否重新发送请求给主节点。记录其他副本节点发送的COMMIT消息到log中。

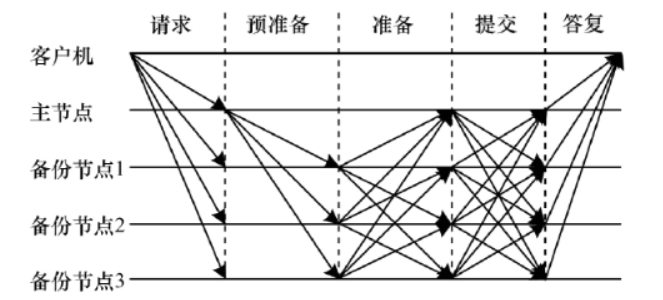


图5-5 PBFT一致性过程

#### 5.4.2.2 视图更改协议

若是主节点不是诚实节点，它可能会给不一样的请求编上相同的序号，或者不去分配序号，或者让相邻的序号不连续。备份节点应当有职责来主动检查这些序号的合法性。若是主节点掉线或者不广播客户端的请求，客户端设置超时机制，超时后客户端向全部副本节点广播请求消息。副本节点检测出主节点做恶或者下线，发起View Change。为了在进行视图更换的时候不破坏系统的安全性，视图更换协议必须满足以下两个条件：

(1) 在上一视图已经执行的日志请求不会被取消。

(2) 在上一视图未被全部执行的日志请求将会在新视图中继续被执行。

系统进行视图更换时，为了满足条件（1），各节点必须将稳定的检查点发送到新主节点；为了满足条件（2），各节点必须将序列号高于稳定检查点的日志请求发送到新主节点。视图更改协议由备份节点发起，编号最小( 要求此编号比当前发生故障的主节点编号大) 且没有发生故障的备份节点当选为新一轮的主节点。视图变更流程如图3-3 所示，具体步骤如下:

1) 发起视图变更。备份节点 i 向其他节点广播视图变更消息，视图变更消息格式为 Signi( View-change，v + 1，h，C，P，Q) ，其中，h 表示备份节点 i 存储的最新检查点编号，C 表示由备份节点 i 存储的所有检查点编号及对应凭证集合，P 表示由备份节点 i 收集的所有预准备消息集合( 对应请求还没有进入提交阶段) ，Q表示备份节点 i 接收到的预准备消息集合( 对应请求还没有进入准备阶段) 。

2) 确认视图变更。备份节点 i 收集 2f + 1 个( 包括其自身) 视图变更消息后，给视图 v + 1 对应的新主节点发送视图变更确认消息，视图变更确认消息格式为 Signi( View-change-ACK，v + 1，i，j，Hash( m) ) ，其中，j 表示发送视图变更消息的备份节点编号集合，m 表示视图变更消息。

3) 广播新视图。当新的主节点收到 2f 个其它节点的 view-change 消息，则证实有足够多的节点认为主节点有问题，新主节点根据存储的信息和日志更新节点数据，发送新视图消息，进入新一轮一致性协议。新视图消息格式为 Signi( New-view，v +1，u，χ) ，其中，u 表示主节点收集到的视图变更证据，χ表示主节点知道的最新检查点、检查点到当前视图中所有已经提交的请求、准备完成的请求和预准备完成的请求。

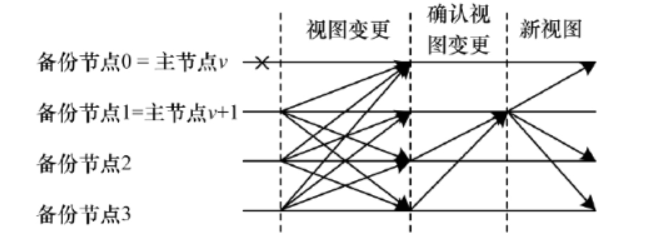


图5-6 PBFT视图更改协议

#### 5.4.2.3 检查点协议

在上述算法流程中，为了确保在View Change的过程中，能够恢复先前的请求，每一个副本节点都记录一些消息到本地的log中，每个节点应该记录下之前曾经共识过什么请求，但如果一直记录下去，数据会越来越大，所以应该有一个机制来实现对数据的删除。当执行请求后副本节点需要把之前该请求的记录消息，即共识过的记录清除掉。

最简单的做法是在Reply消息后，再执行一次当前状态的共识同步，这样做的成本比较高，因此可以在执行完多条请求K（例如：100条）后执行一次状态同步。这个状态同步消息就是CheckPoint消息（checkpoint 就是当前节点处理的最新请求序号。主节点收到请求会给请求记录编号的，这个请求的编号会一直携带）。

副本节点i发送<CheckPoint， n， d， i>给其他节点，n是当前节点所保留的最后一个视图请求编号，d是对当前状态的一个摘要，该CheckPoint消息记录到log中。如果副本节点i收到了2f+1个验证过的CheckPoint消息，那么该节点可以确定该消息对应的状态是正确的并且有足够多的诚实节点已经处于该状态，则清除先前日志中的消息，并以n作为当前一个stable checkpoint（stable checkpoint 就是大部分节点 （2f+1） 已经共识完成的最大请求序号）。

这是理想情况，实际上当副本节点i向其他节点发出CheckPoint消息后，某些节点还没有完成K条请求，所以不会立即对i的请求作出响应，它还会按照自己的节奏，向前行进。并且此时发出的CheckPoint并未形成stable，为了防止其他节点i的处理请求过快，设置一个上文提到的高低水位区间[h， H]来解决这个问题。低水位h等于上一个stable checkpoint的编号，高水位H = h + L，其中L是我们指定的数值，等于checkpoint周期处理请求数K的整数倍，可以设置为L = 2K。当副本节点i处理请求超过高水位H时，此时就会停止脚步，等待stable checkpoint发生变化，再继续前进。

#### 5.4.2.4 一致性保证

系统要完成诚实节点执行顺序的一致性，即当编号为n的消息m在正确节点i中得到了执行，那么该消息在正确节点j中也必须得到执行，且执行编号也为n。更重要的是，即使节点i在执行了消息m之后发生了主节点的切换，节点j还是必须以n为编号执行消息m

但是如果主节点崩溃发生换主，其中只有一个或几个（不是大多数）已经收到了足够的prepare，完成了prepared，其他节点因为网络原因没有收到本应该收到的足够多的prepare（异步网络环境没有任何通信保证，只有最终一定会收到的保证），这个时候如果换主，新主发起新一轮共识，假设新主几点是恶意节点或者不知道前面的消息编号n，新的主节点发起对应于编号n的另一个消息m'。我们如何保证，大部分诚实节点执行的区块顺序是一致的，即他们的消息和对应的编号n一致。

这个时候结合视图更换协议，所有节点在广播更换主节点的申请的时候，需要同时广播自己完成prepared状态的信息，同时给出自己处于这个状态的证据（例如，如果处于prepared状态，就提交2f+1条来自不同节点的prepare信息）。然后，新的出块节点根据收到的2f+1条信息中的状态决定来决定如何对新消息m'进行编号，是否需要重新发送之前的消息m再次共识，并且其他节点可以对主节点进行监督。这个时候无非有两种情况：

1）如果没有任何一个节点提交编号为n的消息m。系统考虑就舍弃上个主节点的消息m，因为各节点判定该消息不是这次的view是无法验证通过的。主节点换主判定依据是：不会有2f+1条commit消息，即不会有2f+1个节点处于prepared状态，考虑作恶节点则不会有f+1个诚实节点处于prepared状态。于是，这个时候，新的主节点会将自己的新消息m’与之前的m设置为同一个编号n，替换掉m。然后，广播消息m’的同时，广播收到的所有轮换申请——向所有节点说明这种情况。

2）如果有其中一个节点提交了编号为n的消息m。由上可知，说明2f+1个节点已经prepared了，至少有f+1个诚实节点已经收到消息m并且处于prepared状态。证明上个主节点的消息m还是可靠的，需要其他节点也同步完成消息m的共识，即其他节点也完成编号为n的消息m的提交。此时主节点会去搜集要重新进行共识的pre-prepare（2f+1个节点的，必然存在一个诚实节点并且有对应的pre-prepare），因此会把编号为n的消息m将其中的view改为自己再重发一次，也就是说需要先达成上一个节点遗留下来的m的共识。

注意：这里有可能会有这个疑问，我们不要commit阶段行不行，视图更换时改成需要2f+1个prepare数据进行证明行不行，这里是不行的，prepare是没有经过一轮共识的，主节点发送pre-prepare之后，副节点就发送prepare了，假设上一个主节点是恶意节点，会直接被误导，也就是说prepare数据是没有经过共识过的。Prepared数据是经历过“简单共识”过的，也就不会受到恶意主节点的误导，从而保证了在View v中执行了的消息也一定还是会在View v+1中执行。这也是为什么我们需要三阶段过程的重要原因。

综上，Commit阶段主要是用来与View Change机制进行配合，从而保证在上一个View中以编号n执行了的消息交易m，在新的View中也会以相同的编号n执行。简而言之，prepare锁定同一个view下的sequence，commit锁定sequence。

5.5 高性能共识机制技术研究

共识过程是区块链实现分布式一致性的核心环节， 区块链系统的共识既要保证安全性，又要保障效率。 共识过程包括主节点选举、区块生成、区块验证投票和区块上链4个步骤。其中对共识速度影响最大的是 主节点选举和区块验证投票这两个步骤。主节点的选 举过程决定了哪个节点来生成区块，如PoW依赖算力 竞争来决定哪个节点生成区块，算力竞争虽然保证了随机性和去中心化特性，却也导致了算力浪费和耗时 过长的问题。主节点的选举频率也是影响系统共识的 一个重要方面。区块验证投票主要包含两个问题，一 是投票共识组如何选择，二是如何对提出的区块形成 一致性意见。共识组的大小决定了共识网络规模，如 果共识组太大，可能会减慢共识速度。共识算法决定 了共识流程的复杂度和节点交换消息的数量，也决定 了达成分布式一致性的代价和时间。

同时，限制区块链吞吐率的主要因素是其链状的结构，因为只有被记录到区块链上的交易才 是有效交易。区块链单条链的结构使得打包交易过程只能够串行进行，串行的模式导致难以通过并行提升效率。 在串行模式下，吞吐率的上界被单节点的网络速度 和处理性能从物理上被限制，且随着网络规模的增大，系统的性能不能同比上升，甚至会有所下降。

所以为了能高效快速并且安全的实现共识机制，我们主要研究了以下四个技术：

* 有向无环图（DAG），改变区块的链式存储结构，变成区块DAG。在区块打包时间不变的情况下，网络中可以并行打包多个区块。
* 借助分片技术， 拆分系统中的资源，各个分片单独处理各自的交易。
* 引入可验证随机函数（VRF），用可验证随机函数抽签，来降低参与共识节点或者某种角色的数量的数量。
* 状态缓存优化技术，允许节点在块创建阶段缓存事务的执行结果，并使用缓存的结果来跳过节点在块验证阶段重新执行事务。

### 5.5.1 有向无环图（DAG）结构

#### 5.5.1.1 技术背景

传统区块链的链式结构有一些先天性的局限。比如，这种结构会有一个类中心化的动作——“打包区块”，整个区块链在任意时刻，都是由记账者单点写入，记账者通过全网POW共识机制，算出nonce随机数并获得区块写入权力，并得挖矿奖励。 这种单点写入区块链的局限性：无法处理高并发请求；因为区块链是否分叉和出块速度以及广播速度有关。当出块速度超过广播速度的时候，会出现多个区块同时在广播的情况，分叉也就产生了，分叉越多安全性越差。比特币为了减少分叉，在性能和安全性中找到的平衡点为：每十分钟出一个块。

有向无环图同传统链式结构的主要区别在于，每个区块中不仅包含一个父区块的 Hash 值。有向无环图应当包含多个区块的 Hash 值，从而构成一个有向无环图的结构，且其包含的区块的 Hash 值应为在其本地视图下最新的没有子区块的区块。DAG 的结构如下图5-7所示。

简单来说，我们可以把DAG看成一种允许分叉的网络结构，意味着出块的速度可以超过广播速度。这一方面导致单位时间内打包的交易变多了；另一方面当一个区块A在被全网广播的时候，另一个分叉区块B也在被全网广播，最后一些节点只会确认A，另一些节点只会确认B，所以DAG允许网络中的节点在同一时间记录的不一样的信息。这两方面综合起来就使DAG呈现出高并发，弱同步的特点。DAG是一种异步记账，这种记账方式可以极大的提高网络处理信息的速度，即TPS。而区块链是一种强同步记账的网络，它要求网络中的每个节点在同一时间记录相同的信息。然而这一要求往往限制住了区块链网络处理信息的能力，使TPS比较低。

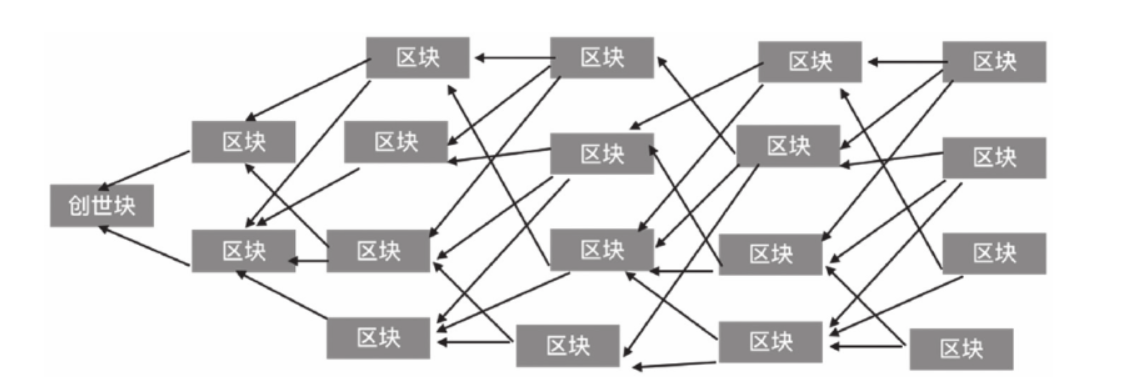


图5-7 DAG结构

#### 5.5.1.2 技术应用

以 Conflux的共识为例，区块之间由多条连接组成，分别是一个父链接和多个引用链接，从而在多个区块之间形成DAG 结构。 其中，父链接代表了投票关系，引用链接代表了先后顺序。 不同于比特币中的最长链原则，Conflux基于GHOST（GHOST算法：即选择拥有最多子区块的区块作为枢轴链（Pivot Chain）上的区块，然后再采用同样的算法将它之后的区块纳入枢轴链），定义了枢纽链并使得区块之间得以形成全局序。

如图5-8 所示。 虽然 Conflux等底层的数据结构不再是传统的区块链的简单链式结构，但是交易仍然被打包形成区块，并且区块之间仍然保持了全局顺序，因此相当程度上保留了区块链特征。 通过DAG结构，系统的并发性能和安全性得以提高，Conflux宣称其测试系统网络下TPS可以达到 6400，并且相较于比特币系统有着更低的延迟和更好的安全性。

需要注意的是，在狭义的共识机制上Conflux 仍然采用了前文中所描述的POW共识机制，在这点上并没有显著的创新，而是通过降低挖矿难度及数据结构的调整从而缩短挖矿间隔、提高吞吐率及降低延迟等，所以仍然不可避免地需要耗费大量的能源进行挖矿。

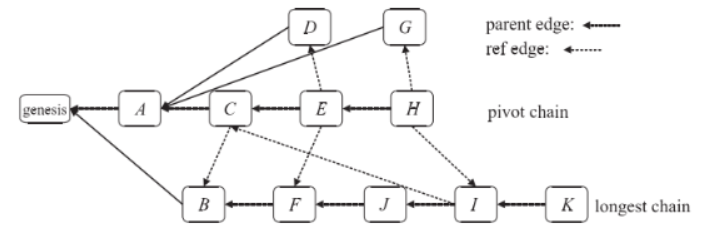


图5-8 Conflux结构

### 5.5.2 执行分片技术

#### 5.5.2.1 技术背景

传统数据库解决数据存储扩展的有效方式是分片，将数据库分割为多个部分并分别存放在不同的服务器，从而达到横向扩展能力，区块链也可将其视为一种数据库，于是也可以采取这种分而治之的思想模式。在分片系统中的节点和相应的交易都会被划分到不同分片中从而提升了扩展性和处理性能，因为每个节点无需处理全部交易，只需要处理它分配到的一部分交易，并且可以和在其他分片中的节点并行执行多线程处理交易。由于对网络进行分片后达到了并行处理的效果，节点无需排队等待验证，而是可以同时验证多个区块，极大地提高了吞吐量和效率。目前分片有以下几种含义：

1. 网络分片，网络分片是指将全网分为不同的片区，每个片区由一个对应的委员会处理，每个委员会内部成员大部分时间只需内部通信，每个片区内部的其他客户端、节点大部分时间可以通过与该分片内委员会通信获得 目前区块链的状态。
2. 交易分片，交易分片是指每个分片委员会只负责处理其对应的交易，如根据交易的ID判断其对应的分片，交易 ID最末位数字如果是i，则由i号分片委员会处理该交易，对交易运行委员会内分布式一致性算法，验证该交易的合法性，决定该交易是否被添加到区块链中。 计算分片使不同的交易以并行的形式被不同的委员会处理，当网络中节点数量增多时，可以增加更多 的委员会，这样不同的交易能够以并行的形式被不同的委员会同时处理，交易处理性能随着网络中节点数 量的增多而增加，进而实现了交易处理的可扩展性。
3. 状态分片，状态分片是指不同分片委员会将处理后的交易分片存储，每个分片委员会只负责处理本分片对应的交易，将交易放到本分片专属的交易区块链上。交易区块链用于存储本分片产生的交易历史或当前分片的未花费交易池信息。存储分片将整个区块链系统的交易数据或未花费的交易输岀(unspent transaction output， UTXO)数据分片存储，降低了节点的存储负担。

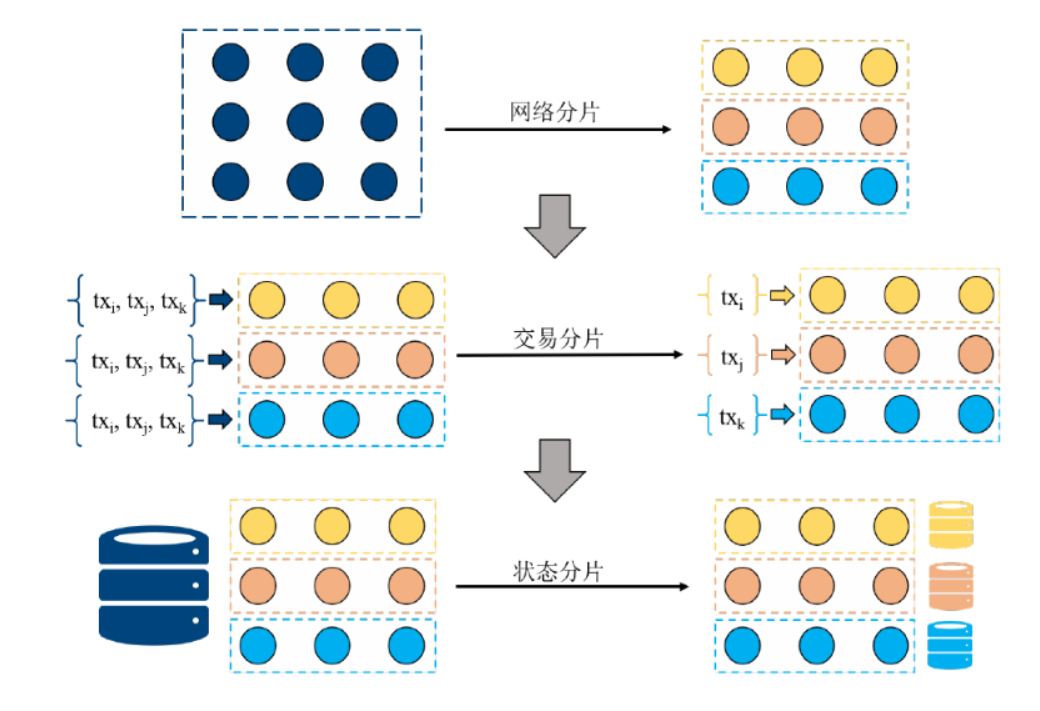


图5-9 执行分片技术

#### 5.5.2.2 技术应用

可以将每个分片想象成个子链，所有的子链构成了完整的区块链。区块链中，分片的技术难点在于保证跨分片交易的原子性。一笔交易如果被接受，在所有的分片中都会被接受；如果不被接受，所有的分片都会拒绝这笔交易。即状态的改变是原子性的，全网中的状态是相同的。

##### 1 Elastico

Elastico 周期性地共识区块， 每个周期内， 节点执行以下操作:

1. 身份获取和委员会分配: 每个节点产生一个身份信息， 身份信息由公钥、IP 地址和一个 PoW 难题解组成， PoW 难题解是为了防止恶意节点伪造多个身份信息， 之后根据身份信息将节点划分到不同的委员会
2. 委员会内部节点发现: Elastico 通过建立“目录服务委员会”， 节点可以高效地与同一委员会内部的其他成员建立通信连接;
3. 委员会内部共识: 委员会内部运行 PBFT 共识算法对交易集达成共识， 将共识结果发送到“共识委员会”;
4. 最终共识: “共识委员会”接受其他委员会的共识结果并重新打包交易， 运行 PBFT 共识算法达成最终共识并向全网广播;
5. 产生随机数: “共识委员会”产生一组随机数， 用于下一周期中节点重新构建身份信息。 分片技术通过并行处理提升系统性能， 使其随着全网节点数量的增多而线性提升; 另外每个节点只存储部分数据， 减少了节点数据存储压力。Elastico中也存在一些问题， 比如没有解决原子性交换问题， 当分片拒绝某笔跨片交易后， 原分片中的交易资产被永远锁定

##### 2 RapidChain

在Rapidchain系统中主要包含三个重要阶段，分别是Bootstrapping phase（引导启动阶段）、Consensus phase（共识阶段）和Reconfiguration phase（重构阶段）。Bootstrapping阶段只会在在Rapidchain系统开始时运行一次，这个阶段是为了创建一个初始随机源，并随机选出一个特殊的委员会，叫做参考委员会（reference committee）。再由这个参考委员会的成员对节点进行随机分配，构成一个个分片委员会。三阶段的执行如下：

1. 引导启动阶段：。RapidChain无需可信启动，首先在所有参与节点中，利用随机图(random graph)选举出 根群组(root group)，然后利用根群组运行分布式随机数生成协议，生成参考委员会。参考委员会负责生 成其他k个普通委员会。
2. 共识阶段：协议分时期运行，每个时期分为多个轮，每一轮中每个委员会使用时代随机源（epoch randomness）选出一个领导者（leader）；运行实用拜占庭协议达成一致性共识。
3. 重构阶段：在每个时期结束前，参考委员会都会生成一个时期随机数（epoch randomness）用于下个时期。参考委员会的成员还会运行DRG protocol（distribute random generation protocol）针对一个无偏差随机值达成共识。并定期对各个委员会进行洗牌，避免分区攻击的发生。

### 5.5.3 可验证随机函数(VRF)

#### 5.5.3.1 技术背景

VRF 最初于1999年被 Mical等人提出，其全称为Verifiable Random Function（可验证随机函数），可以看作是一个随机预言机，意思是说我可以通过任意的一个输入，获得一个随机数输出。但是可验证随机函数比随机预言机多了一个非交互的零知识证明，可以用来该随机数输出的正确性，表明这个随机数的确是某个人生成的。VRF一般包括以下三个部分：

1. 密钥生成keygen。 这一步生成一对密钥对，分别是私钥sk和验证公钥vk。
2. 评估算法evalute。 针对一个消息ｍ，使用私钥sk生成一个伪随机字符串 r 和证据 p ，p是零知识证明算出来的证据。
3. 验证函数veritfy。验证函数将验证公钥vk、消息ｍ、伪随机字符串ｒ和证据 p 作为输入。 如果r 确实是评估算法基于私钥 sk 和消息 m 产生的输出，则验证函数返回真。

VRF为在开放网络中去中心化的选举验证节点提供了一种可能性。 借助VRF机制，第一是可以不被预测的随机找到一个节点进行区块打包，第二区块链系统可以去中心化地在一个开放的网络中选出一批节点作为委员会来参与一段时期的共识。 基于VRF的共识协议的区块链系统有 Algorand、Dfinity、VBFT等。

#### 5.5.3.2 技术应用

##### 1 Algorand

2016 年 ，图灵奖得 Micali 主导提出了AlgoRand 算法。该算法的本质是通过随机算法（VRF）从一群大范围的验证者中选出一部分验证者运行拜占庭容错算法。AlgoRand 算法的共识过程分为两个步骤：随机选取区块生成者生成区块；BA\*算法达成共识。

随机生成区块时，算法提出的可验证随机函数（Verifiable Random Function，VRF）能够产生一组可验证的随机数。区块链中任何节点都可以通过验证函数检验 VRF 随机生成数的真实性。每一轮的区块生产者与验证委员，通过兑奖的方式查看自己是否是下一轮的生产者，而不是使用“公开选举”的方式。AlgoRand 算法利用可验证随机函数掩盖生产者选举的步骤，一般的拜占庭容错算法会在每轮开始前公开选举出下一区块的生产者以及委员会成员，AlgoRand 算法则是在每一轮开始时公布中奖号码，在中奖者表明自己的身份之前，没有人可以预测下一轮的生产者以及委员会人选。

AlgoRand 的拜占庭协议 BA\*类似于两个阶段的投票过程。在第一阶段，选出“验证者”共识最多的候选区块；在第二阶段，“验证者”对候选区块运行二元拜占庭协议，要么接受此候选区块，要么接受空区块。

Algorand 共识机制随机选举出块节点和验证节点，能够抵抗恶意攻击，去中心化程度和安全性高，吞吐量高于 Bitcoin，交易确认时间低于1min。BA\* 共识机制类似 PBFT 共识机制，资源消耗低，每个步骤都要求拜占庭节点数少于 f =(Ｒ－ 1) /3，算法时间复杂度为 O( n2) ，但其至少需进行 6 轮通信，通信开销高于 PBFT 共识机制。

##### 2 VBFT

在 Ontology 项目中，结合 PoS、BFT 和 VＲF 提出基于可验证随机函数的拜占庭共识机制( VBFT)实现了网络的快速共识。每个区块的 VＲF 值根据前一个区块计算得到，具体过程为提取前一个区块中的交易事务，计算 1024 bit 的哈希值并将该哈希值作为输出。系统中将网络分为共识网络和公共网络。用户通过质押代币后参与到共识网络，共识合约自动更新共识节点列表和相应权益( PoS 表) ，每产生一个区块更新一次。共识节点分为出块者、验证者、确认者，所有验证者组成验证者集合，所有确认者组成确认者集合。VBFT 共识机制相当于选举制度，合法的选民都具有投票权、被投票权，由可验证随机函数验证选民身份的合法性。VBFT 共识机制运行时依据 VＲF 值作为索引，从 PoS 表中选举共识节点，具体流程如下:

1. 选举出块者和提出区块。每一轮依据 VＲF值从共识网络中选举潜在出块者集合，每个潜在出块者提出一个区块
2. 验证区块。每一轮依据 VRF值从共识网络中选举出验证者集合，每个验证者从共识网 络中收集被提出的区块进行验证，并投票给具有最高优先级的区块，广播投票消息。
3. 确认区块。每一轮依据 VＲF 值从共识网络中选举出确认者集合，确认者统计验证者的投票，最后确认达成共识的区块并广播确认消息。
4. 更新区块链。共识网络中所有的节点复制确认者认可的区块，只有结束一轮共识，才开启新一轮共识。

每一个区块决定了 VＲF 函数的一个输出，由VＲF 函数确定共识节点序列，根据节点序列分配优先级，然后通过节点优先级加权决定区块优先级，最后投票给优先级最高的区块，从而解决分叉问题。VBFT 共识机制中随机选择出块节点、验证节点、确认节点，能够抵抗恶意攻击，去中心化程度和安全性高，吞吐量达到 3 000 TPS，交易确认时间为 5 s ～10 s。另外，VBFT 共识机制由共识节点执行 BFT 共识机制，资源消耗低，可以容忍的最大拜占庭节点数为 f =(Ｒ - 1) /3，时间复杂度为 O( n2) ，扩展性随共识节点的增加而降低。

### 5.5.4 状态缓存技术

#### 5.5.4.1 技术背景

事务是由用户在一段时间内生成的，事务表示的是一个个交易信息，一般来说事务首先经过所有节点的验证，注意此时验证通过的事务并没有确定交易成功，验证完进入一个事务池，等池子里的事务多了，节点们就可以进行将事务打包，将事务打包并发布区块。

节点会在事务池中选择一个子集，并在构建新块时对它们排序。同时执行这些事务，以确保其块只包含有效的事务，这被称为块创建阶段。由共识协议，这个“获胜”的块由创建它的节点广播，所有其他节点执行这个块的事务，以确保它只包含有效的事务并更新状态。这被称为验证阶段。然后重复创建下一个块的整个过程。

通常，节点在创建和验证阶段执行相同的事务。这将导致在块验证阶段执行大量冗余事务。因此，核心思想是允许节点在块创建阶段缓存事务的执行结果，并使用相同的方法来避免(在任何可能的情况下)在块验证阶段重新执行事务。

5.5.4.2 技术应用

Nitin Awathare等人提出了RENOIR，它在块创建阶段缓存事务执行的状态，并在验证阶段重用它，使节点在块验证期间跳过(重新)执行这些事务。每个块包含一个有序的事务集，执行时更新状态机。

RENOIR以以下方式解决这些问题。对于每一个高度为i的新块，在创建阶段，节点 r 会构建一个事务有序列表T' 并执行它们。对于有序列表T' 中的每个事务，在本地缓存节点对T'有序列表中每个事务的读和写操作。并且，按照每次读取事务的顺序，来维护每个读操作的key。

在其他验证节点接收到高度为 i 的包含有序事务列表T 的新块时，验证节点使用缓存的信息来处理T中的事务。特别是对于接收到的块中的每个事务tx，如果在块创建阶段块创建节点没有执行tx，例如，tx不属于T'，则其他节点执行tx。否则，如果接收块中tx之前的事务以与块创建阶段相同的顺序访问(写入对应的值)tx的读操作中的key，验证节点跳过执行tx。其思想是，如果在块创建和验证阶段访问一个事务的读操作集中的key的事务的顺序是相同的，那么事务的执行结果在块验证阶段仍然是可重用的。

在实现中使用两个表，即事务表和键表来维护所需的信息。事务表维护在块创建阶段执行的每个事务的读和写的操作集。类似地，键表维护访问每个事务的读和写操作集中的key的事务的顺序。此外，键表还维护在块创建阶段由事务写入的值。在块验证阶段，节点使用这两个表只跳过那些访问顺序与块创建阶段访问顺序相同的事务。此外，一旦节点决定跳过某个事务，它将使用存储在键表中的值来更新该事务触及的状态。更新后的状态将由未来的事务使用。

5.6 联盟链共识协议小结

联盟链共有拜占庭共识算法和故障容错共识算法两大类进行选择。拜占庭共识算法虽然可以抵御拜占庭攻击，但PBFT共识算法过程中采用两次全节点通信，需要比较大的网络传输和通信开销。Hot-stuff 虽然将PBFT的视图更改消息复杂度减少到O (n)，但在正常执行时，其消息复杂度与PBFT相当。SBFT 是一种使用称为c和e收集器的协议，它们是随机选择的，并在准备阶段和确认阶段负责收集签名。这些收集器有助于避免全网节点相互广播。在理想情况下，可以实现O (c\*n)线性开销。但是如果收集器是恶意的，这将导致SBFT性能的下降，并导致O (n2)的通信开销。Tendermint算法通过锁定、解锁机制来提高吞吐量，但是在交易处理的效率和达成共识的时延等方面需要进一步提升，并且在共识流程、区块同步和节点管理方面存在问题。

而故障容错算法经典的为Raft和Paxos，其中Raft算法在工程领域应用广泛，相比于Paxos易于理解和实现。Raft虽然无法抵御拜占庭攻击，但是其算法通信复杂度在任何情况下只有O (n)。如果集群总节点数为 n，对于日志记录阶段，通信次数为n-1，对于提交数据阶段，通信次数也为n-1，总通信次数为2n-2，因此通信复杂度为O（n）。此外，在平均共识时延上Raft只有604.5/ms相对于PBFT的2119.8/ms，同时平均吞吐量可以达到189.95/ps相对于PBFT的46.25/ps。

综上所述，根据中交智运庞大的货运体量，在基于高吞吐量和高低时延的要求下，建议采用Raft共识协议，同时可以辅以高性能的共识技术来进一步提高共识效率。

# 六、敏捷应用技术

6.1 与云计算深度融合的区块链快速应用技术

区块链技术的诸多优势都基于大量的节点资源，这些节点资源分散繁多，管理比较困难，因此，将云计算技术和区块链技术进行深度融合，实现对区块链节点、框架、网络等虚拟化的表达，实现节点资源动态扩容、远程运维，最终实现了区块链系统的快速部署、快捷运维和敏捷开发等目标。通过构建虚拟全节点，在不影响系统正常运行的基础上，通过配置监管策略，自动完成系统安全性合规性的穿透式柔性监管，为区块链行业应用提供安全和技术保障，为区块链技术的发展提供强有力的援助和支持。实现区块链快速应用的主要关键技术包括：

（1）将区块链技术与云计算技术深度融合，支持以VM、Docker、Kubernetes集群等方式实现区块链节点、框架、网络的全虚拟化表达，支持动态扩容、远程运维，从而达成快速部署、快捷运维和敏捷开发的目标。对于货运物流业来说，在货运物流平台上有多少个参与方，就会有多少个节点，这些节点加入到货运物流区块链网络中，对整个网络都是一个非常大的挑战，我们需要花费大量的时间成本去对每一个节点进行监控和维护。此外，这些节点在区块链网络中，会慢慢的产生大量的数据和日志，并且数据的增量是不可预计的，在这样的情况下，本地资源永远是有限的，只有云资源才可以不断的扩展，将区块链与云计算融合，借助云平台的容错可扩展、网络的多链路负载、计算资源的动态调整，我们可以针对区块链中的节点能够更好的实现节点的服务，从一定角度来讲也节省了节点的运行成本，提高了整个系统之间交互的效率。

（2）支持区块链节点的追踪与可视化：该项技术研究可以为货运物流平台的运维人员提供非常便利且高效的区块链网络和节点运维服务，它将各项监控指标和追踪内容抽离成标准的监管模块，之后将标准监管模块放置于每一个区块链节点中，标准监管模块每隔一段时间自动搜集指标数据，搜集完全后通过WebAPI服务提供给前端可视化界面进行展示，运维人员能够根据每一节点的指标变化对该节点的运行现状作出评判。

（3）支持联盟链穿透式监管：联盟链穿透式监管的主要原则是将联盟链网络中的每一个环节产生的真实可信的数据串联起来实时的上传到监管节点，以便于提高监管节点的效率，降低监管节点的成本。联盟链网络中每一个环节产生的数据都是大量的，随着时间的流逝，这些数据会变得异常庞大，因此，结合货运物流的行业实际进行区块链式的分析，研究什么数据需要上传到监管节点，要以怎样的方式上传到监管节点。

（4）实现区块链环境的一键式快速部署与应用，且安全、合规地部署区块链应用，简化区块链部署、运维和管理等环节：研究区块链环境的一键式部署是实现区块链商业化，促进区块链落地应用发展的重要内容之一，基于Docker容器技术的发展与Shell脚本编程，将区块链网络的各节点抽离成标准节点模板，之后将标准的节点模板镜像化容器化，放置于安全的可控的仓库，当需要部署网络时，只需要配置脚本需求，就可以安全高效的部署区块链网络。

6.2 基于 Docker 的区块链实现

6.2.1 Docker

Docker 是一个开源的应用容器引擎，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。开发者在本地编译测试通过的容器可以批量地在生产环境中部署，包括VMs（虚拟机）、bare metal、OpenStack 集群和其他的基础应用平台。

简单的理解，Docker类似于集装箱，各式各样的货物，经过集装箱的标准化进行托管，而集装箱和集装箱之间没有影响。也就是说，Docker平台就是一个软件集装箱化平台，这就意味着我们自己可以构建应用程序，将其依赖关系一起打包到一个容器中，然后这容器就很容易运送到其他的机器上进行运行，而且非常易于装载、复制、移除，非常适合软件弹性架构。因此，就像船只、火车或卡车运输集装箱而不论其内部的货物一样，软件容器充当软件部署的标准单元，其中可以包含不同的代码和依赖项。 按照这种方式容器化软件，开发人员和 IT 专业人员只需进行极少修改或不修改，即可将其部署到不同的环境。

总而言之，Docker 是一个开放平台，使开发人员和管理员可以在称为容器的松散隔离的环境中构建镜像、交付和运行分布式应用程序。以便在开发、QA 和生产环境之间进行高效的应用程序生命周期管理。

6.2.2 Fabric在Docker中实现

Hyperledger Fabric 区块链系统采用容器式运行的方式，其系统中的智能合约称为Chaincode。Chaincode 分为特殊的系统Chaincode 和交易Chaincode，前者负责区块链的管理，后者负责保存状态和账本数据，并执行交易。

Chaincode 生命周期分为 5 个阶段：打包、安装、实例化、升级、删除。Chaincode 通过API 与区块链中的各种节点进行交互，同时也可以通过API 对 Chaincode 的生命周期进行管理。

打包过程包括创建包和包的签名，源码按照部署规范格式打包，签名主要用于检查和确认Chaincode 所有者，可以在创建包的同时进行签名，一次签名的包用执行install 交易，多次签名包围多个所有者一次签名。安装过程需指定 CDS 包的路径，发送一条 SignedProposal 消息给 peer 节点的生命周期系统节点调用 LSCC 上的 install 方法完成 Chaincode 安装。实例化过程调用LSCC，在 channel（Ledger 上包含特定peer 节点的私有链）上启动一个 Chaincode 容器，实现 Chaincode 与 channel 的绑定。实例化交易执行过程中，验证 Chaincode 的实例化策略，以确保实例化交易执行的合法性。实例化成功后，Chaincode 即处于激活状态，时刻监听并接受交易请求。升级过程类似于实例化过程，即修改新的 Chaincode 版本并与 channel 绑定。为保证升级该Chaincode 的合法性，该过程须验证当前版本的实例化策略。删除过程只需删除对应的容器，同时删除每个安装的 Chaincode 的背书节点上的 Singed CDS。

编写Chaincode 智能合约时需要实现Chiancode 接口，以响应传来的交易消息， Init 和Invoke 是两个必须借口，分别实现智能合约的部署（实例化、升级等）和交易调用。

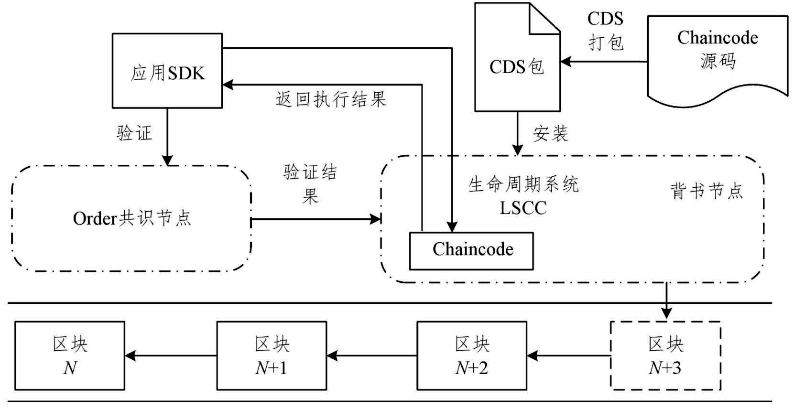


图6-1: Chaincode 的部署和执行流程

Chaincode 在超级账本上安装部署及运行流程如图[6-1](#_bookmark4)所示，具体过程如下：

1. Chaincode 源码以 CDS 规范签名生成CDS 包；
2. 通过生命周期系统 Chaincode，将 CDS 包安装在同一通道内的背书节点上，生成运行在节点上的 Chaincode；
3. 应用程序通过SDK 发送请求到背书节点；
4. 节点通过 Chaincode 执行交易并将执行结果返回给应用程序；
5. 应用程序搜集结果，将结果发送给Order 共识服务节点；
6. 共识服务节点执行共识过程并生成区块验证结果；
7. 背书节点各自验证交易并提交到超级账本区块中。

以太坊和超级账本是两个应用最广泛的智能合约平台，但两者在架构设计上 面向不同的应用领域。前者定位为完全独立于任何特定应用领域的通用平台，具有对应的账户和代币功能，允许智能合约作为特殊的账户部署在区块链节点上，应 用程序通过API 调用节点上的智能合约来产生交易，变更区块状态。而后者为模块 化和可扩展的架构，不具有自身的代币，共识机制采用 PBFT 而非以太坊的 PoW， 具有较高的共识效率，而且共识服务从背书节点分离，独立形成可插拔模块，可扩展性强。其主要作为联盟链，面向银行、医疗保健和供应链等行业。因此，基于两个平台的智能合约也具有各自的特点。不同智能合约的特性对比如表[6-1](#_bookmark5)所示：



表 6-1: 以太坊和超级账本智能合约对比

6.3 基于 VM 的区块链实现方式

6.3.1 Ethereum在VM中实现

以太坊的部署书基于虚拟机的方式实现的，采用 VM 方式，实现区块链的节点、框架、网络的全虚拟化表达。以太坊包含一个以太坊虚拟机 EVM，它是一个完全独立的沙盒，合约代码在 EVM 内部运行并且对外隔离。Vitalik Buterin 提出的支持图灵完备语言的以太坊智能合约平台，为智能合约提供了专用的开发语言，主要Serpent 和Solidity 两种。Serpent 类似于Python 语言，具备简洁的特性；Solidity 在语法上类似于 JavaScript，是以太坊官方推荐的智能合约语言，它具有详细的开发文档。下面介绍 EVM 的主要机制。

6.3.2 gas 计费机制

在以太坊系统中，为了防止区块链网络资源滥用或由图灵完备引起的无限循 环故障，任何可编程的计算都受计费限制。该计费机制以 gas为单位，创建智能合约、调用消息、访问账户存储的数据，并且虚拟机上的运行操作都对应一定的gas 计费标准。gas计费的引入为智能合约的运行提供了机制上的安全保障，一旦gas 超过计费限制gasLimit，整个交易将会被回滚，以保证数据的完整性和安全性。

6.3.3 EVM 虚拟机

EVM虚拟机被部署在执行智能合约的各个节点之上。负责对只能合约进行指令解码，并按照堆栈完全顺序执行代码。其结构不同于标准的冯诺依曼模型，程序 代码并非保存在通用内存和永久存储，而是被置于特殊的交互式ROM中。其内存模型和存储模型分别为基于简单字地址的字节数组和字数组，并有可变和不可变之分。虚拟机提供简单栈式结构，为了与Keccak-256哈希算法和椭圆曲线算法相匹配，栈的元素大小被设计为 256 位。

EVM本身运行一个状态函数，也称状态机，用于持续监测状态。当新的进程触发时，EVM运行代码并将一定数据写入内存或永久存储，每一个新状态都是基于上一个状态进行改变。

6.3.4 合约的创建与运行过程

以太坊的系统中，创建合约可看作一种特殊的交易过程，合约创建函数利用 一系列固定参数实现新合约的创建，并产生一组新的状态。过程如下

(*δ′ ， g′ ， A*) *≡* Λ(*δ， s， o， g， p， v， i， e*) (1)

其中，σ 为系统状态，s 为交易发送者，δ 为交易账户主体，g 为可用gas，p 为gas 价格，v 为账户金额，i 为初始化EVM 代码，e 为创建的智能合约栈的深度，δ′为系统新状态，g′ 为剩余 gas，A 为子状态。

最终通过执行初始化 EVM 代码，创建新的合约账户，产生账户地址、存储空间以及账户的主题代码。该过程中，出去发生交易所消耗的 gas，代码创建的 gas 消耗量与所创建的合约代码量成正比。然而，一旦 gas 剩余量小宇代码创建所需gas，则会产生gas 异常（Out of Gas，OOG），并且 gas 剩余量将被置为零，也不再创建新的合约。

合约运行模型则描述了在接受一系列字节码和环境数据元组之后，系统状态 的转变方式。在实际运行中，该模型由全系统状态和虚拟机状态的迭代过程构成。 迭代器不停地运行迭代函数，知道虚拟机出现状态异常（如OOG）而暂停或产生正常结果数据而暂停。

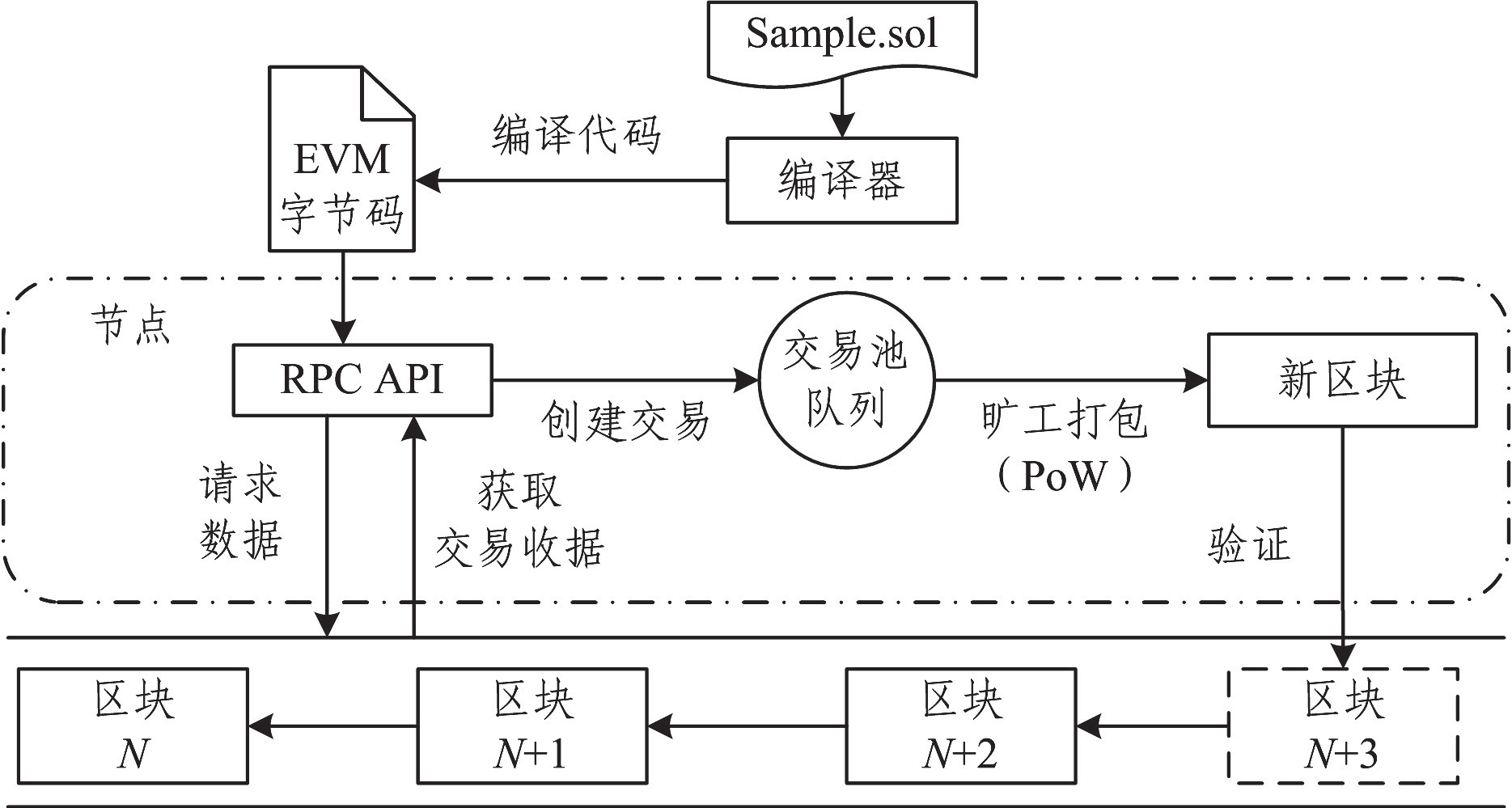


图6-2 智能合约部署

智能合约部署的流程图如图6-2所示，部署流程为：

1. 编写智能合约代码， 形成合约代码文件（如SampleContract。sol）
2. 通过智能合约编译器对代码文件进行编译，将其转换成可以在 EVM 虚拟机执行的字节码；
3. 向区块链节点 RPC API 发送创建交易（部署合约）请求，交易被验证合法后， 识别为创建交易，检查输入数据，进入交易池；
4. 旷工打包该交易，生成新的区块，并广播到P2P 网络；节点接收到区块后对交易进行验证和处理，为合约创建 EVM 环境，生成智能合约账户地址，并将区块入链；
5. API 获取智能合约创建交易的收据，得到智能合约账户地址，部署完成。

智能合约的调用流程与部署流程类似，也是通过 RPC API 创建交易，并由验证节点对交易进行处理，调用 EVM 实例，进行状态变更。

七、结论

本项目以区块链技术在网络货运领域的应用为核心，深入调研了区块链应用过程中的关键技术，包括但不限于主子链多层跨链技术、链上数据的隐私保护技术、高可信的身份及权限管理技术、高性能的共识算法及可拔插的密钥生成算法技术及区块链现实部署的全虚拟化表达技术。项目研究成果从技术角度为后续的平台建设奠定了基础，推动了我司平台建设的核心信息技术革新。

项目以Hyperledger Fabric为重点调研目标，深入研究区块链与交通物流经济的深度融合，可实现数据加密、数据溯源、数据共享、数据自动化处理的智慧服务，提高区块链科技创新应用价值，为“区块链+网络货运”的发展规划指明了方向，对中交智运区块链平台的建设具有十分重要的意义。