



区块链与数字货币

浙江大学 杨小虎

2020年12月4日

教学安排（共8周）

- 数字货币和区块链技术原理
- 数字货币和区块链技术原理
- 以太坊技术原理与架构
- 智能合约与开发实践
- 以太坊应用开发
- HyperLedger Fabric技术原理与架构
- HyperLedger Fabric应用开发
- 数字货币和区块链发展趋势



课程考核

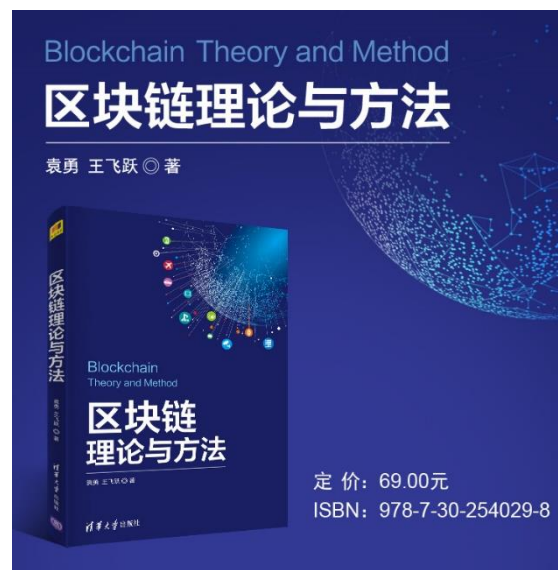
- 课程平时作业与课堂表现：45分
- 课程大作业：55分

| 作业 | 占总成绩比例 | 备注 |
|-----|--------|---------|
| 作业1 | 15% | 区块链技术原理 |
| 作业2 | 15% | 以太坊 |
| 作业3 | 15% | Fabric |
| 大作业 | 55% | 以太坊 |





毛德操著，
《区块链技术》，
浙江大学出版社，2019.



袁勇、王飞跃著，
《区块链理论与方法》，
清华大学出版社，2019.

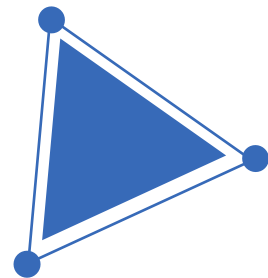
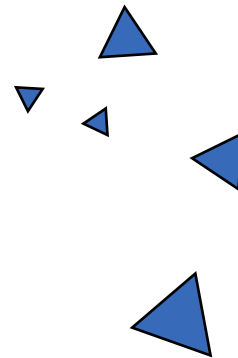
参考资料：比特币、以太坊、HyperLedger、Libra等网上资料

推荐网站：<http://book.8btc.com/>



01

基础知识



货币知识

- 货币的价值
 - 使用价值
 - 交换价值
- 货币的形式
 - 贵金属：金、银、铜
 - 纸币
 - 数字货币



交子，宋代出现，世界上最早的纸币

货币知识

- 纸币的本质：信用

- 金本位

- 国家信用

- 布雷顿森林体系

- 美元与黄金锚定，每一美元的“含金量”为0.888671克黄金
 - 其他国家的法定货币与美元挂钩，规定与美元间的汇率
 - 实行可调整的固定汇率。各国货币对美元的汇率只可在法定汇率 $\pm 1\%$ 的范围内波动。
 - 1944年启动，1971年结束



货币知识

• 纸币的全球化

▫ 国际货币基金组织特别提款权

- 特别提款权的价值最初确定为相当于0.888671克纯金，当时也相当于 1 美元。在布雷顿森林体系解体后，特别提款权价值被重新定义为一篮子货币。
- 执董会每五年或在必要时提前检查特别提款权篮子，以确保特别提款权能反映各组成货币在世界贸易和金融体系中的相对重要性。
- 特别提款权的价值：根据伦敦时间中午左右观察到的即期汇率，每日确定特别提款权的美元价值，并在基金组织网站上公布。

| 货币 | 2015年检查中确定的权重 | 货币单位的固定数量，为期五年，自2016年10月1日起 |
|-----|---------------|-----------------------------|
| 美元 | 41.73 | 0.58252 |
| 欧元 | 30.93 | 0.38671 |
| 人民币 | 8.33 | 1.0174 |
| 日元 | 8.09 | 11.900 |
| 英镑 | 10.92 | 0.085946 |



货币知识

- 纸币的全球化
 - 美元、欧元、英镑、日元……
 - 国际货币基金组织特别提款权
 - 特别提款权是国际货币基金组织为了补充国际储备资产的不足，创设的一种新的国际储备资产和记账单位，是IMF按照成员国缴存的基金份额分配给各成员国的，分配后就成为该成员国的资产，并与黄金和外汇一起构成该成员国的国际储备资产。
 - 当成员国发生国际收支逆差时，可将特别提款权划给另一个成员国，换取可兑换货币来密闭逆差，特别提款权还可用来偿还基金组织的贷款，但它不能兑换黄金，不能直接用于国际贸易或非贸易的支付，所以，特别提款权只是成员国在国际货币基金组织的特别提款权账户的一种账面资产。

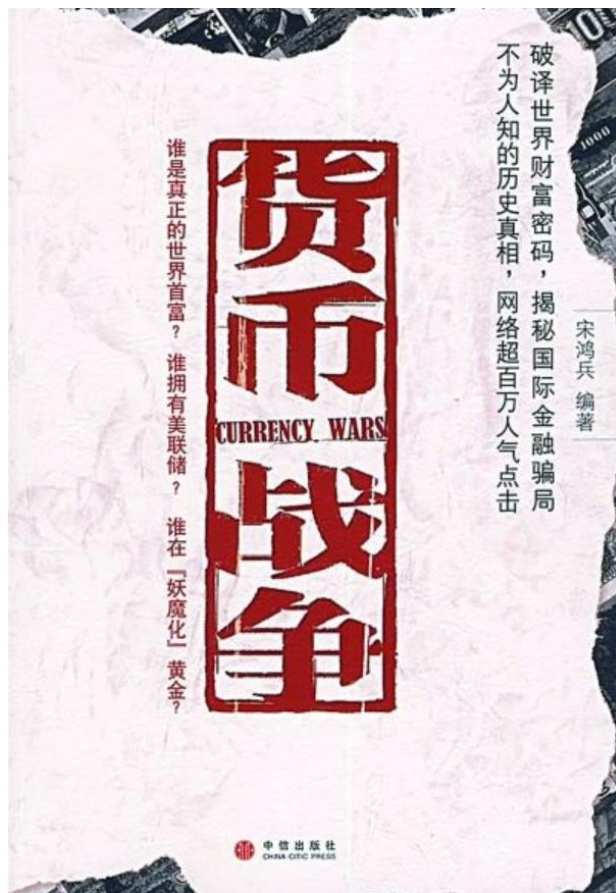


货币理论

- 马克思政治经济学
 - 商品的使用价值与交换价值
- 凯恩斯经济学
 - 市场这只“看不见的手”有时候会失灵，需要政府干预
- 哈耶克经济学
 - 自由市场、自由经营、自由竞争、自动调节、自动均衡
 - 货币非国有化
- 现代货币理论MMT
 - 货币的本质就是欠条，即IOU (I Owe You)



货币战争？



Hash算法

- 把任意长度的输入通过hash算法变换成固定长度的输出，该输出就是hash值。
 - 输入域（空间）大于输出域（空间）。
 - 两个不同的输入可能会产生两个相同的输出，称为碰撞。
 - 从输出无法推导出输入。



最简单的hash算法

$x \bmod y = z$ (整数取模运算)

x , y , z 均为整数, 其中 y 是质数, y 不能为0

例:

$$y=7$$

$$9 \bmod 7 = 2; 10 \bmod 7 = 3; 50 \bmod 7 = 1, 100 \bmod 7 = 2 \dots\dots$$

寻找一个足够大的质数 y , 可以实现长度为 y 的哈希表



Hash算法

- Hash实际上是一种思想，包含很多算法。
- 应用：
 - 快速定位（数据库中散列表）
 - 错误校验
 - 唯一性验证
- 常见算法：MD4、MD5、SHA1、SHA2



hash算法的安全性

- 逆向困难：难以从输出推导出输入。
 - MD、SHA算法都不可逆。
- 抗碰撞：很难找到两个不同的输入，可以产生相同的输出。
 - MD4、MD5已经被王小云教授攻破。
 - 王小云的研究成果是给定消息 M_1 ，能够计算获取 M_2 ，使得 M_2 产生的散列值与 M_1 产生的散列值相同。



安全哈希算法SHA256

- SHA-2, Secure Hash Algorithm 2的缩写, 一种安全hash算法标准, 由美国国家安全局National Security Agency研发。
- SHA256是SHA-2下细分出的一种算法, 对于任意长度的输入数据, SHA256都会产生一个256bit长的哈希值, 通常用一个长度为64的十六进制字符串来表示
 - 例如: A7FCFC6B5269BDCCE571798D618EA219A68B96CB87A0E21080C2E758D23E4CE9
- SHA256的安全性
 - 输出为256位, 输出空间是 2^{256} , 是“亿亿亿亿亿亿亿亿”。
 - 目前尚未有破解方法。



安全哈希算法SHA256

```

1 Note: All variables are unsigned 32 bits and wrap modulo 2^32 when calculating
2
3
4 Initialize variables
5 (first 32 bits of the fractional parts of the square roots of the first 8 primes 2..19):
6 h0 := 0x6a09e667
7 h1 := 0xbb67ae85
8 h2 := 0x3c6ef372
9 h3 := 0xa54ff53a
10 h4 := 0x510e527f
11 h5 := 0x9b05688c
12 h6 := 0x1f83d9ab
13 h7 := 0x5be0cd19
14
15
16 Initialize table of round constants
17 (first 32 bits of the fractional parts of the cube roots of the first 64 primes 2..311):
18 k[0..63] :=
19 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
20 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
21 0xe49b96c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
22 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
23 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
24 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
25 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
26 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
27
28
29 Pre-processing:
30 append the bit '1' to the message
31 append k bits '0', where k is the minimum number >= 0 such that the resulting message
32   length (in bits) is congruent to 448(mod 512)
33 append length of message (before pre-processing), in bits, as 64-bit big-endian integer
34
35
36 Process the message in successive 512-bit chunks:
37 break message into 512-bit chunks
38 for each chunk
39   break chunk into sixteen 32-bit big-endian words w[0..15]
40

```

```

41 Extend the sixteen 32-bit words into sixty-four 32-bit words:
42 for i from 16 to 63
43   s0 := (w[i-15] rightrotate 7) xor (w[i-15] rightrotate 18) xor (w[i-15] rightshift 3)
44   s1 := (w[i-2] rightrotate 17) xor (w[i-2] rightrotate 19) xor (w[i-2] rightshift 10)
45   w[i] := w[i-16] + s0 + w[i-7] + s1
46
47 Initialize hash value for this chunk:
48 a := h0
49 b := h1
50 c := h2
51 d := h3
52 e := h4
53 f := h5
54 g := h6
55 h := h7
56
57 Main loop:
58 for i from 0 to 63
59   s0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
60   maj := (a and b) xor (a and c) xor (b and c)
61   t2 := s0 + maj
62   s1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
63   ch := (e and f) xor ((not e) and g)
64   t1 := h + s1 + ch + k[i] + w[i]
65   h := g
66   g := f
67   f := e
68   e := d + t1
69   d := c
70   c := b
71   b := a
72   a := t1 + t2
73
74 Add this chunk's hash to result so far:
75 h0 := h0 + a
76 h1 := h1 + b
77 h2 := h2 + c
78 h3 := h3 + d
79 h4 := h4 + e
80 h5 := h5 + f
81 h6 := h6 + g
82 h7 := h7 + h
83
84 Produce the final hash value (big-endian):
85 digest = hash = h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7

```

<https://blog.csdn.net/u011583927/article/details/80905740>



加密算法

最简单的加密算法：凯撒密码

- 明文字母表：ABCDEFGHIJKLMNOPQRSTUVWXYZ
- 密文字母表：DEFGHIJKLMNOPQRSTUVWXYZABC
- 明文：THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
- 密文：WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ



加密算法

M是明文， C是密文

E是加密函数， D是解密函数， k是密钥

$$E_k(M)=C$$

$$D_k(C)=M$$

$$D_k(E_k(M))=M$$



加密算法

M是明文， C是密文

E是加密函数， D是解密函数， k_1, k_2 是密钥

$$E_{k_1}(M)=C$$

$$D_{k_2}(C)=M$$

$$D_{k_2}(E_{k_1}(M))=M$$

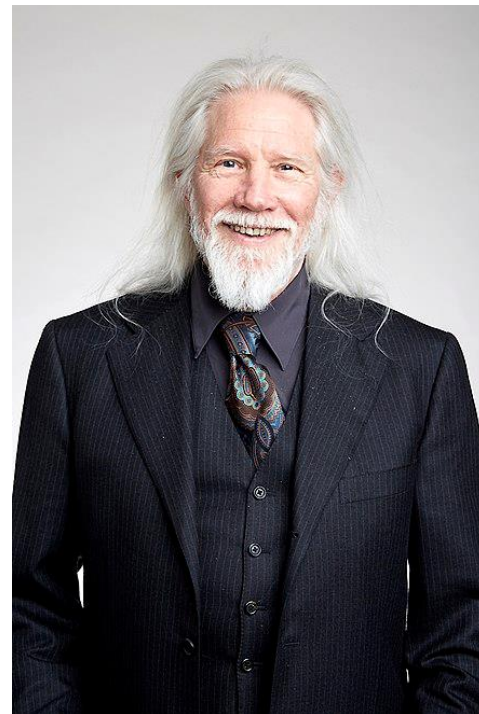
$K_1 = k_2$, 对称加密算法

$K_1 \neq k_2$ 非对称加密算法

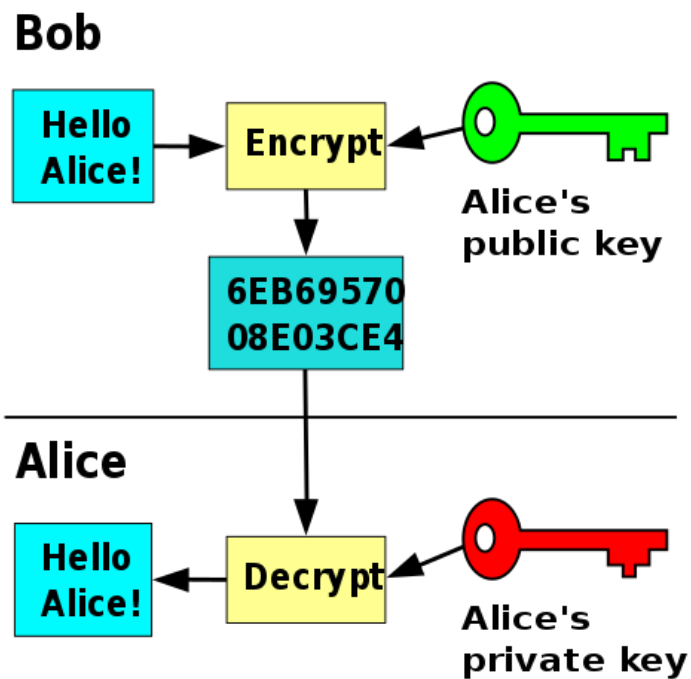
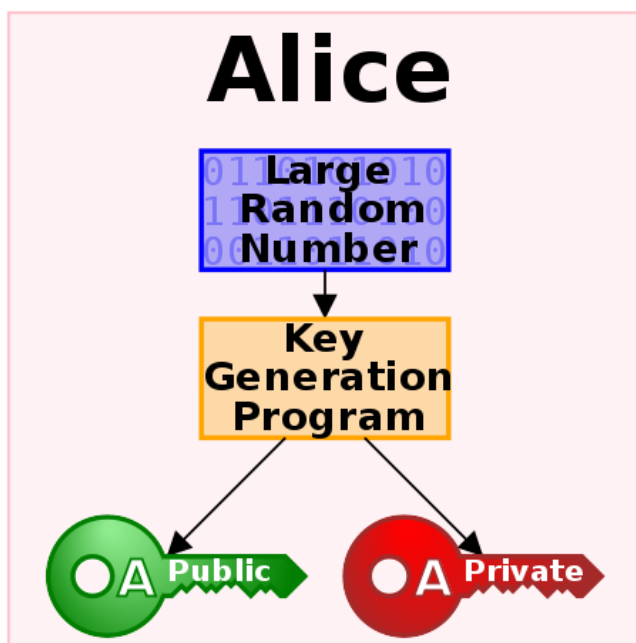


非对称加密算法

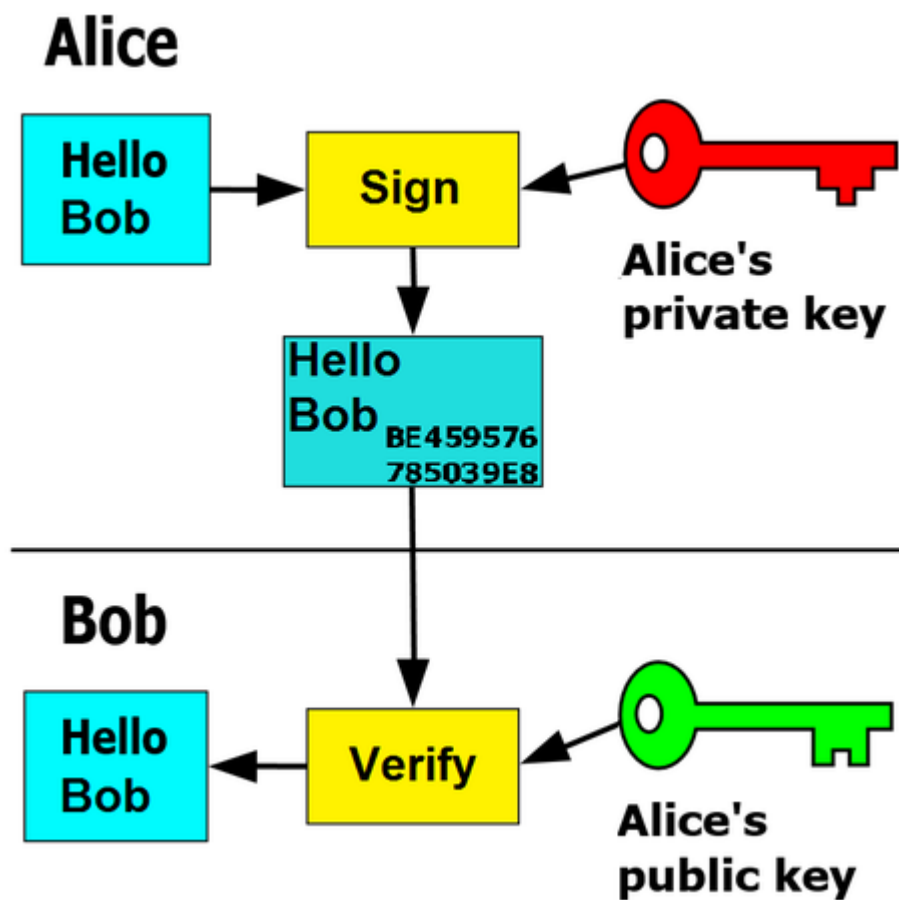
- 1970年，英国学者James H. Ellis提出设想
- 1976年，美国学者Whitfield Diffie and Martin Hellman首次提出一种可实现的非对称加密算法——
exponentiation in a finite field



非对称加密算法



数字签名



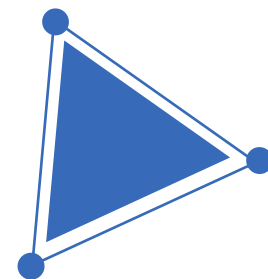
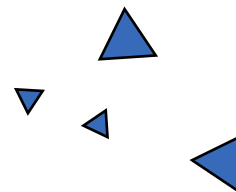
非对称加密算法

- 基于大素数分解难题：
 - RSA
- 基于离散对数难解问题：
 - 椭圆曲线加密
 - Diffie-Hellman 加密



02

区块链技术原理



起源

- 中本聪 (Satoshi Nakamoto), 2008
- 比特币: 一种点对点的电子现金系统



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

总共将发行2100万个比特币
目前已生成1800万个, 目前市值超10000美元/个, 总市场规模超过1800亿美元。

迄今最成功的区块链应用:
10多年来没有出现过任何一次服务暂停
任何交易均可被追溯

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.



数字货币面临的2个问题

➤ 问题1：虚拟货币

- 解决方案：数字签名（非对称加密技术）

➤ 问题2：多重支付

- 解决方案：分布式账本



分布式账本技术原理

- 将交易向全网所有节点进行广播
- 众多记账节点对记账权达成共识，由共识确认的记账节点把记账区块发布给全网
- 所有账本数据完整存储于区块链网络的每个节点
- 所有节点都对账本数据的合法性和完整性进行验证



分布式账本技术原理

➤ 两个核心技术：

- 以链式区块组织账本数据实现账本数据的不可篡改
- 分布式的可信记账机制



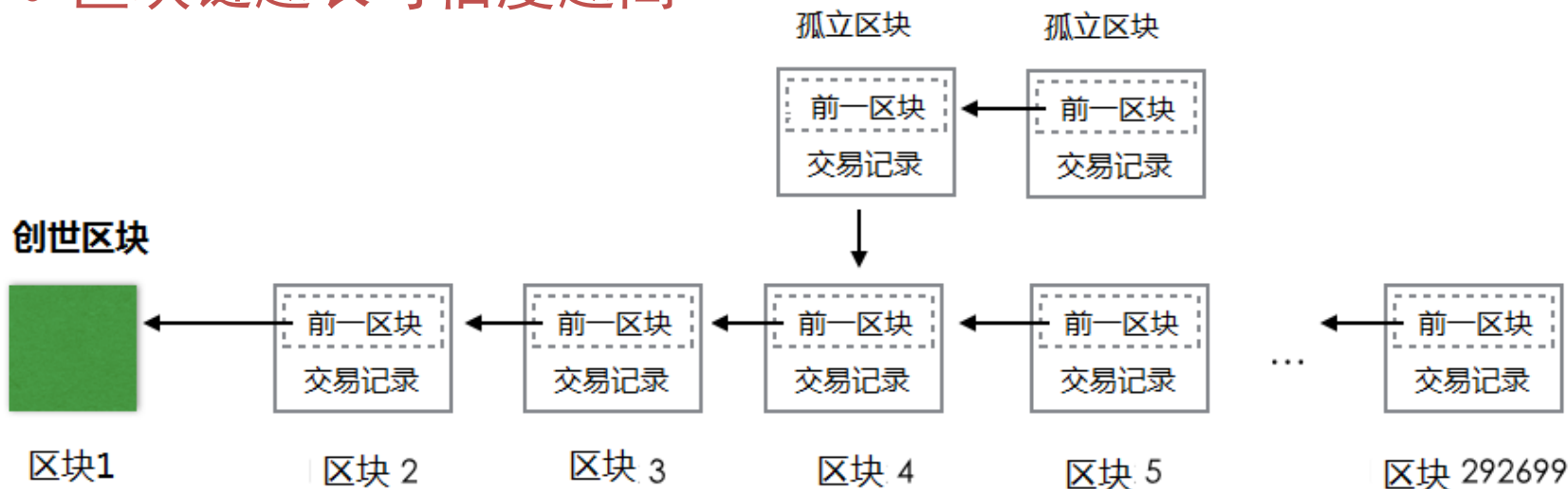
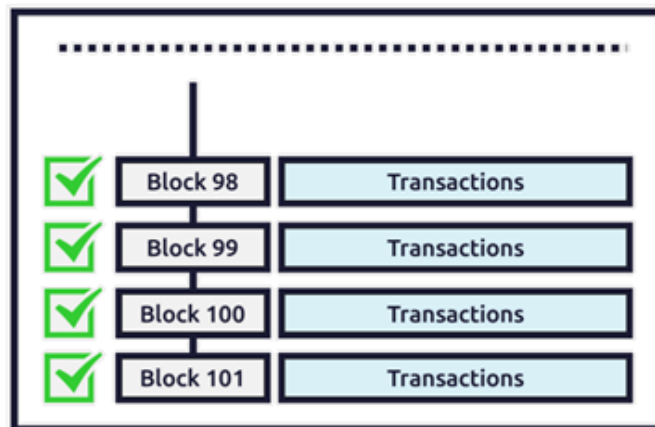
区块链宏观结构

➤ 区块链

- 基于哈希值进行链接

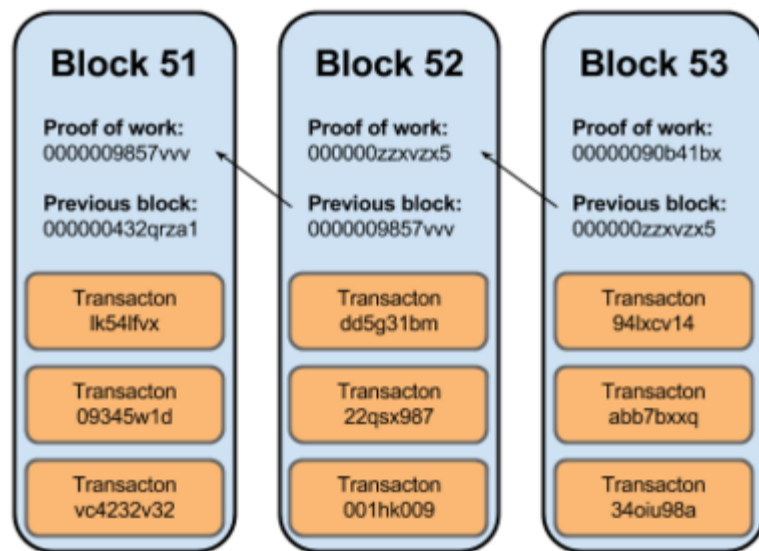
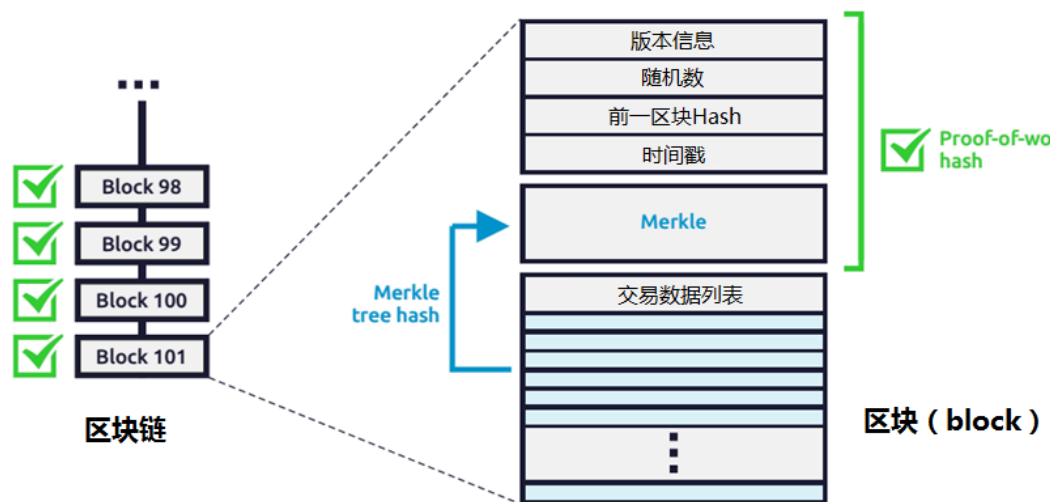
➤ 特点

- 区块链中数据无法篡改或删除
- 区块链越长可信度越高



区块的微观结构

- 每个区块包括区块头和交易数据两个部分
 - 区块头由当前区块的元数据和前一区块的Hash值构成
 - Merkle树用于对交易数据列表进行快速寻址



区块的微观结构

区块的结构

| 大小 | 字段 | 描述 |
|--------------|-------|------------------|
| 4字节 | 区块大小 | 用字节表示的该字段之后的区块大小 |
| 80字节 | 区块头 | 组成区块头的几个字段 |
| 1-9 字节（可变整数） | 交易计数器 | 交易的数量 |
| 可变的 | 交易 | 记录在区块里的交易信息 |

区块头的结构

| 大小 | 字段 | 描述 |
|------|---------|--------------------------|
| 4字节 | 版本 | 版本号，用于跟踪软件/协议的更新 |
| 32字节 | 父区块哈希值 | 引用区块链中父区块的哈希值 |
| 32字节 | Merkle根 | 该区块中交易的merkle树根的哈希值 |
| 4字节 | 时间戳 | 该区块产生的近似时间（精确到秒的Unix时间戳） |
| 4字节 | 难度目标 | 该区块工作量证明算法的难度目标 |
| 4字节 | Nonce | 用于工作量证明算法的计数器 |



区块头代码

```
class CBlockHeader {} //这是一个“类”，或数据机构，内含下面这些字段：

] int32_t nVersion      //所采用Bitcoin协议的版本号

] uint256 hashPrevBlock //上一个块的（块头）Hash值。

] uint256 hashMerkleRoot //所记载交易记录的TxID，
                          //即其Hash值所构成Merkle树的根

] uint32_t nTime        //本块的发布时间

] uint32_t nBits //为挖矿过程设置的难度，Hash值中须有的前导零的位数。

] uint32_t nNonce      //在挖矿过程中使Hash值达到nBits字段所
                          //要求前导零位数的试凑值。
```



区块的微观结构

区块标识符：区块头哈希值和区块高度

- **区块主标识符是它的加密哈希值**，一个通过SHA256算法对区块头进行二次哈希计算而得到的数字指纹。例如
:000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f是第一个比特币区块的区块哈希值。
- 区块哈希值实际上并不包含在区块的数据结构
- 第二种识别区块的方式是通过该区块在区块链中的位置，即“**区块高度 (block height)**”。例如：高度为0的区块就是创世区块。
- 和区块哈希值不同的是，区块高度并不是唯一的标识符，因为有可能出现区块链分叉。

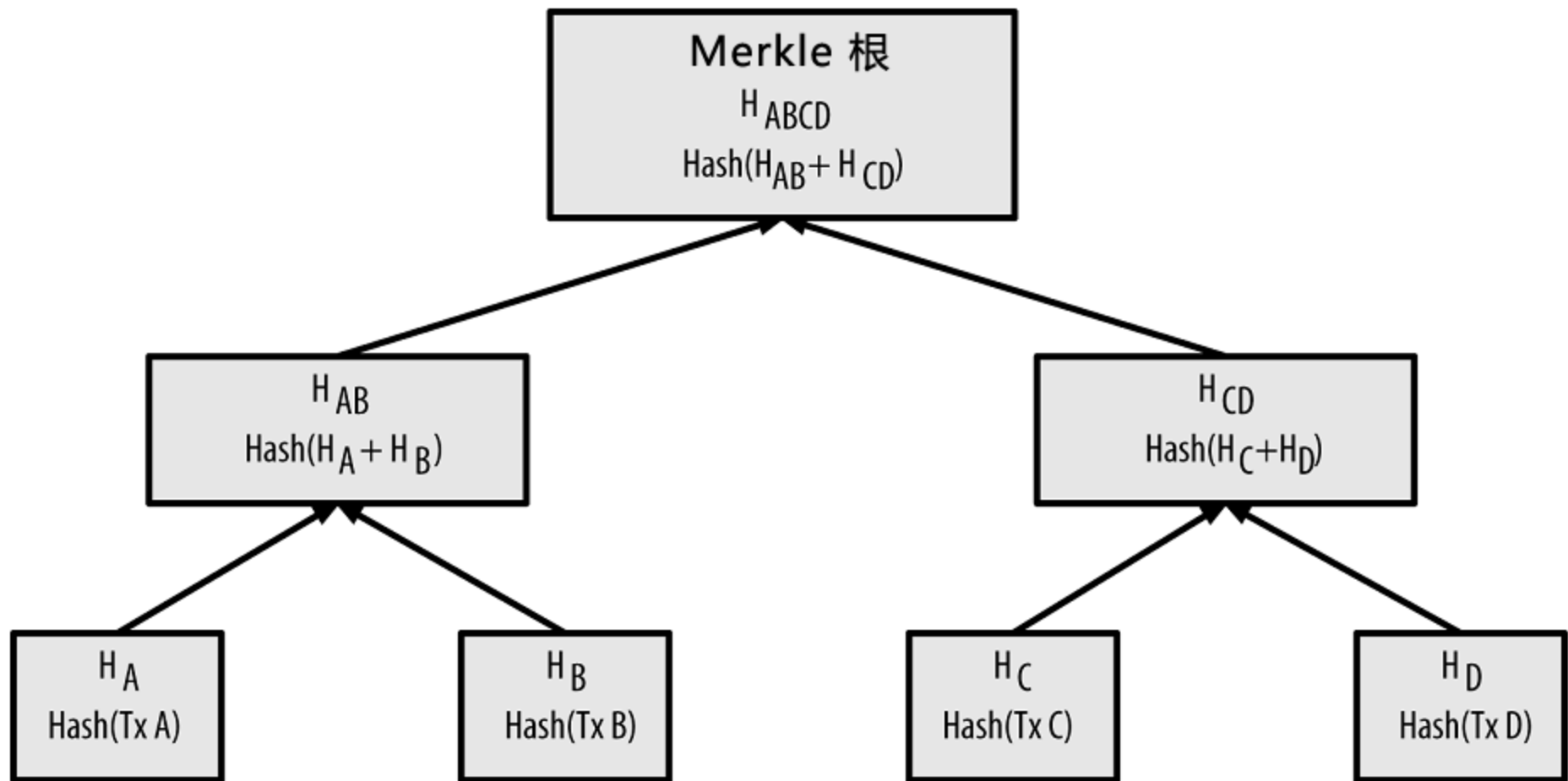


区块的微观结构：Merkle树

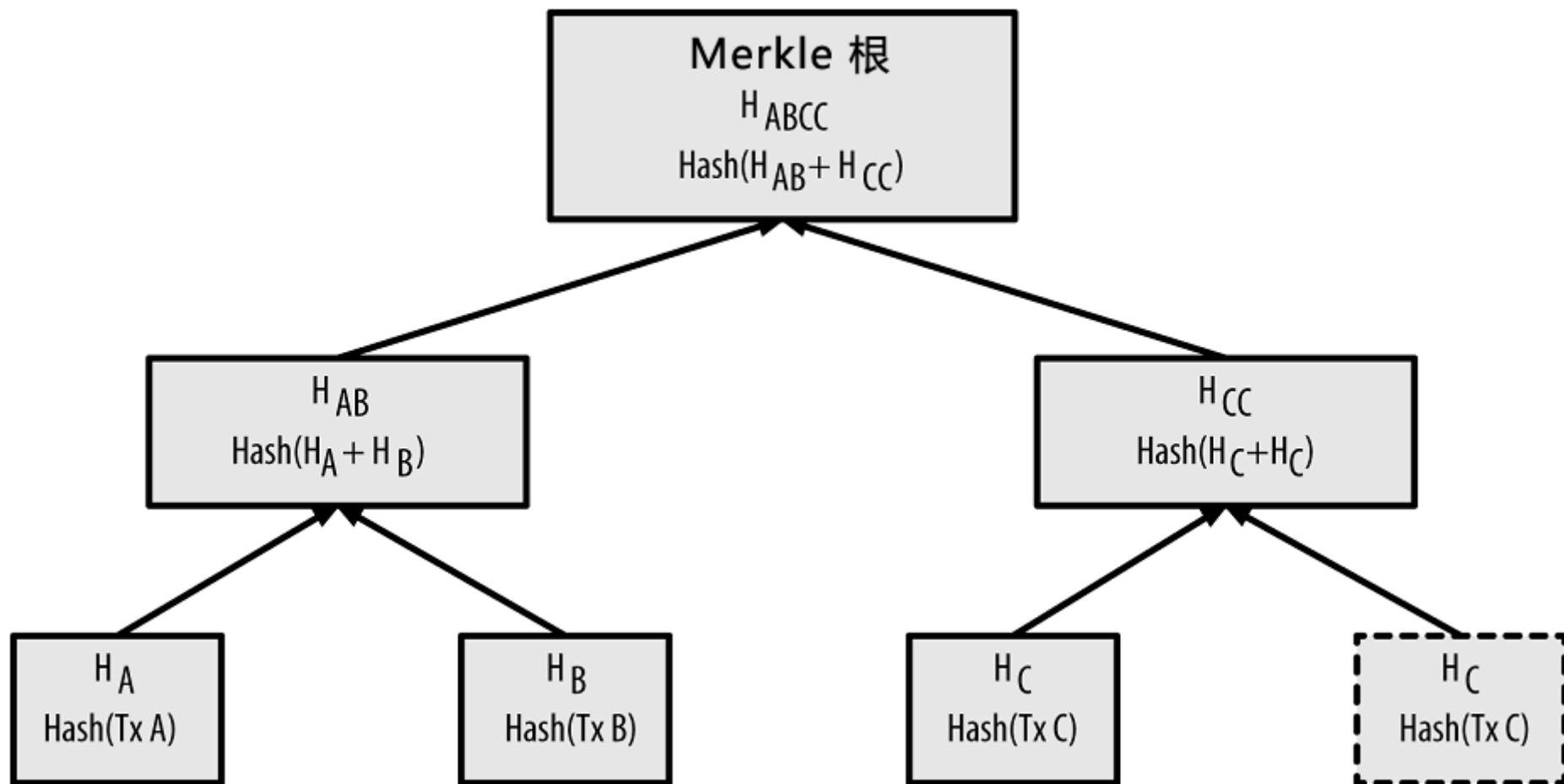
- Merkle树是一种哈希二叉树，它是一种用作快速归纳和校验大规模数据完整性的数据结构。这种二叉树包含加密哈希值。
 - 叶节点是数据块的哈希值。
 - 非叶节点的哈希值是根据它下面子节点的值哈希计算得到。
- 在比特币网络中，Merkle树被用来归纳一个区块中的所有交易，同时生成整个交易集合的数字指纹，且提供了一种校验区块是否存在某交易的高效途径。
- Merkle树中使用两次SHA256算法计算结点的哈希值。
 - $H^{\sim}A^{\sim} = \text{SHA256}(\text{SHA256}(\text{交易}A))$ //两次SHA256是为了提高安全强度
- 当N个数据元素经过加密后插入Merkle树时，你至多计算 $\log_2(N)$ 次就能检查出任意某数据元素是否在该树中，这使得该数据结构非常高效。



区块的微观结构：Merkle树



区块的微观结构：Merkle树



Merkle树

Ralph Merkle

Ralph Merkle was born in Berkeley, California, in 1952. He received his B.S. in computer science (1974) from UC Berkeley and an M.S. (1977) and Ph.D. in electrical engineering from Stanford University (1979).

As an undergraduate in 1974, Merkle discovered a general method of securing electronic communications using a system of cryptographic key exchange now known as Merkle's Puzzles. Unfortunately, the idea was met with disinterest by his professors, and languished until Merkle learned about Martin Hellman and Whitfield Diffie at Stanford.

Merkle joined the team at Stanford for a summer in 1976 and became a doctoral candidate under Hellman the following fall. Working with Diffie and Hellman, Merkle developed the world's earliest public key cryptographic system. Their insight underpins secure transactions on the Internet, enabling e-commerce and a host of other interactions in which secure electronic communications are required.

On graduation, Merkle worked for Elxsi, a small computer company in Silicon Valley. Since 1988, Merkle has been researching nanotechnology and, in 2003, became a distinguished professor at Georgia Tech before returning to California in 2006.

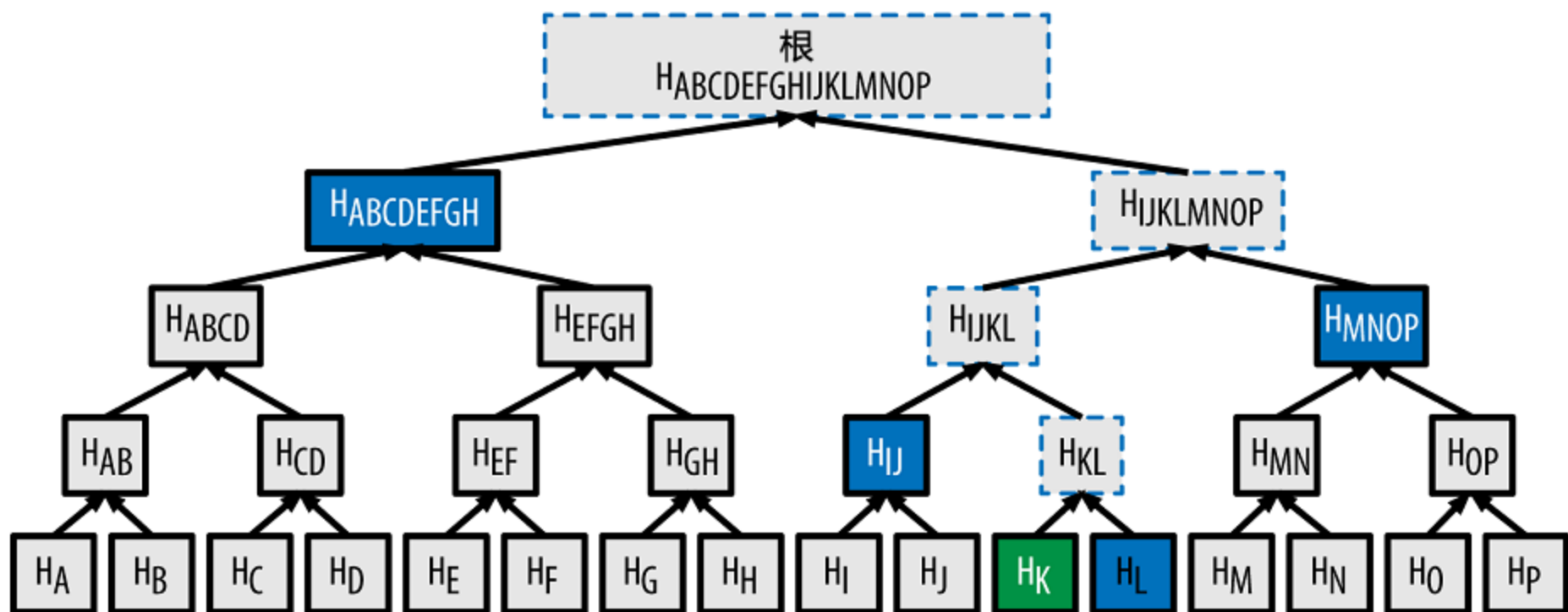
He has been awarded the RSA Award in Mathematics (2000) and the IEEE Richard W. Hamming Medal (2010).



- 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。
- 快速定位修改：如果其中某个交易（数据）发生修改，从根向下可以快速定位被修改的数据。
- 快速验证其中数据：要验证某个交易（数据）是否在Merkle树中，只需要提供一条从该交易的叶节点开始的路径，经过hash计算比对后即可验证。



区块的微观结构：Merkle树



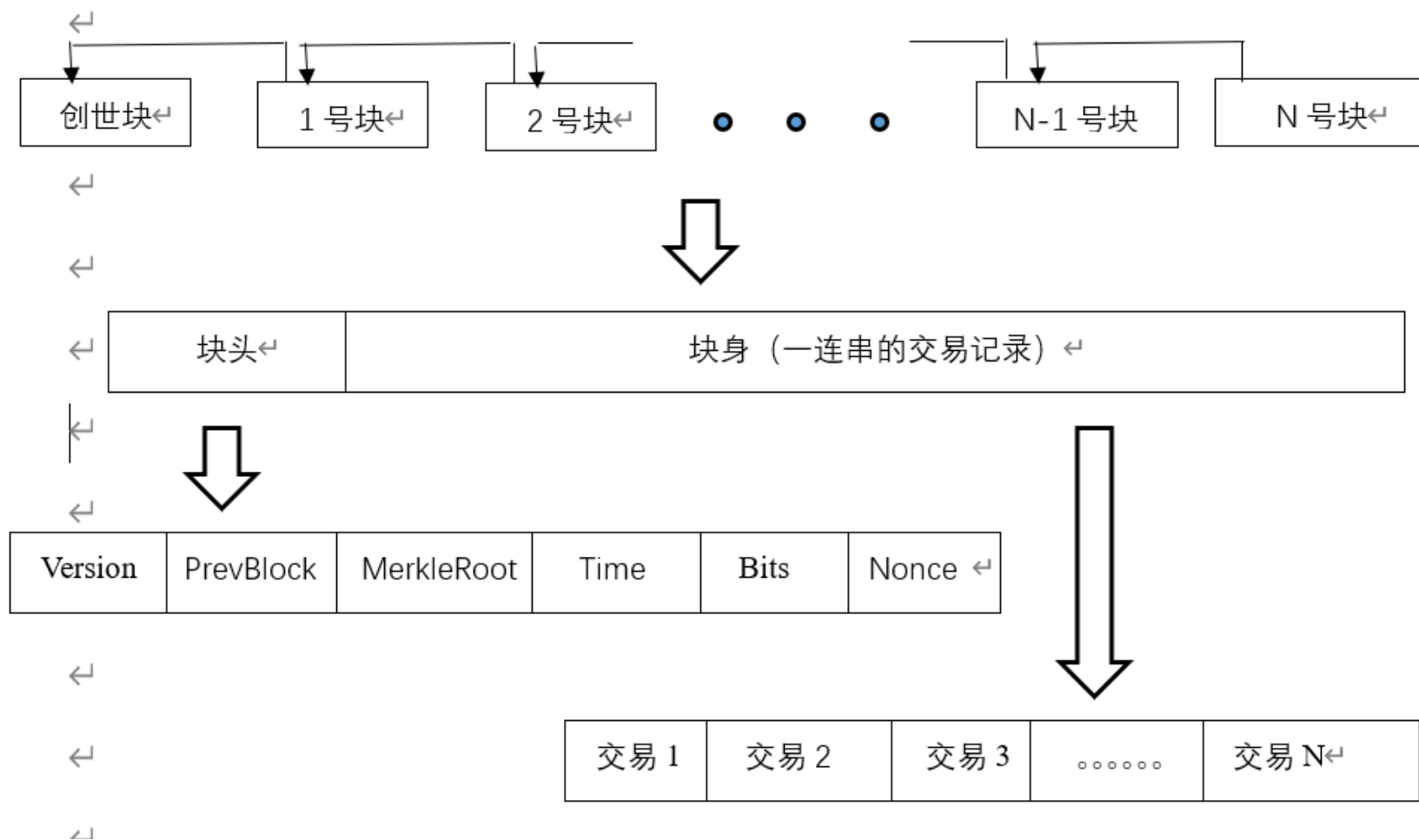
Merkle树的价值：简单支付验证

| 交易数量 | 区块的近似大小 | 路径大小（哈希数量） | 路径大小（字节） |
|-----------|---------|------------|----------|
| 16笔交易 | 4KB | 4个哈希 | 128字节 |
| 512笔交易 | 128KB | 9个哈希 | 288字节 |
| 2048笔交易 | 512KB | 11个哈希 | 352字节 |
| 65,535笔交易 | 16MB | 16个哈希 | 512字节 |

有了Merkle树，一个节点能够仅下载区块头（80字节/区块），然后通过从一个满节点回溯一条Merkle路径就能认证一笔交易的存在，而不需要存储或者传输区块链中大多数内容。

这种不需要维护一条完整区块链的节点，又被称作简单支付验证（SPV）节点，它不需要下载整个区块而通过Merkle路径去验证交易的存在。





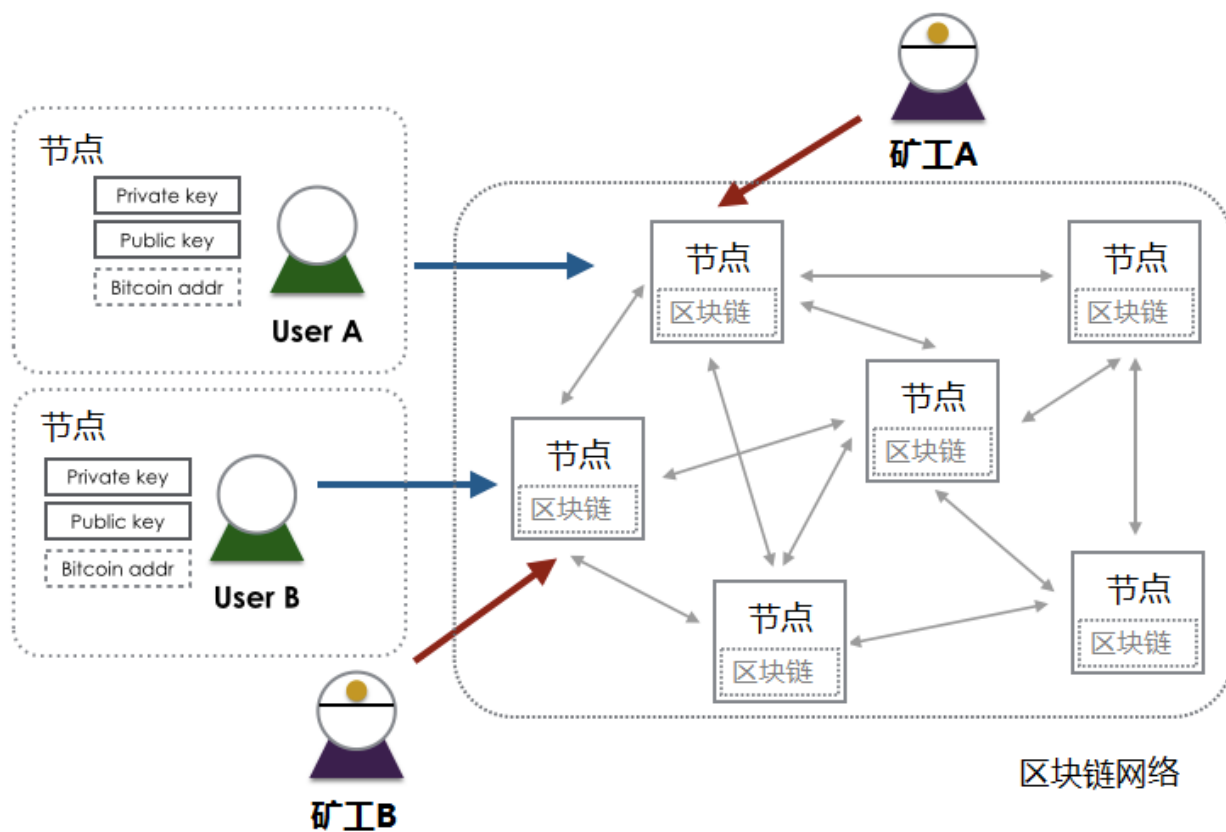
分布式账本技术原理

➤ 两个核心技术：

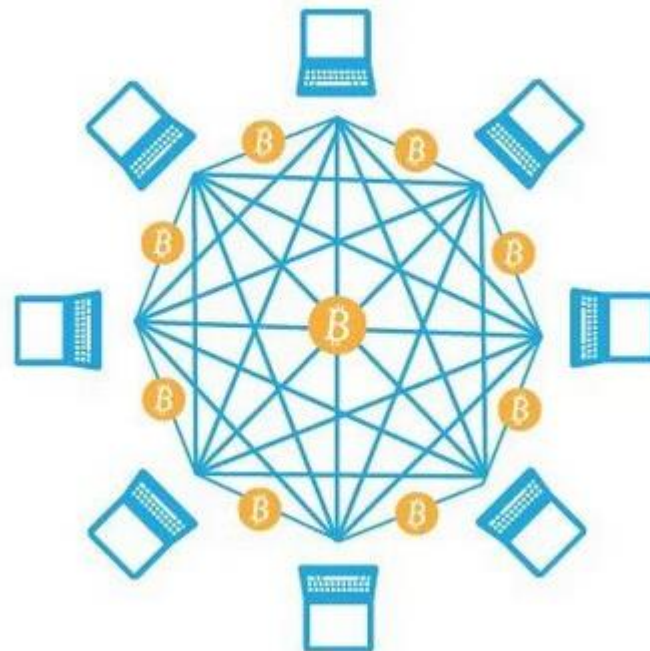
- 以链式区块组织账本数据实现账本数据的不可篡改
- 分布式的可信记账机制



共识机制：由谁记账



拜占庭将军问题（分布式一致性）



拜占庭将军问题（分布式一致性）

- 1982年**Leslie Lamport**（2013年的图灵奖得主）与Robert Shostak联名发表了《Byzantine Generals Problem》，对拜占庭将军算法作了完整的阐述。
- 1989年，Lamport又提出了一个“Paxos算法”，又称“Paxos规程（Paxos Protocol）”。
- 1998年，Lamport在一篇论文《The Part-Time Parliament》中对这个算法作了系统的阐述并提供了数学证明。
- 2001年，Lamport发表了《Paxos Made Simple》进一步加以解释，但人们认为还是太复杂。
- 1999年，Miguel Castro和**Barbara Liskov**（2008年图灵奖得主）联名发表了《实用拜占庭容错（Practical Byzantine Fault Tolerance）》一文，提出了一个**PBFT算法**。
- 2014年，斯坦福大学的Diego Ongaro & John Ousterhout又发表了《寻找可理解的共识算法（In Search of an Understandble Consensus Algorithm）》一文，提出了一个称为**RAFT的算法**。
- **PBFT**和**RAFT**算法都是在区块链中常用的共识算法，但是不是在彻底去中心化的公有链中。这两种算法有个共性，就是“委员会”中都有个类似于主持人那样的“领导者（Leader）”或“主导者（Prime）”的角色。



共识机制：由谁记账

➤ 目的：

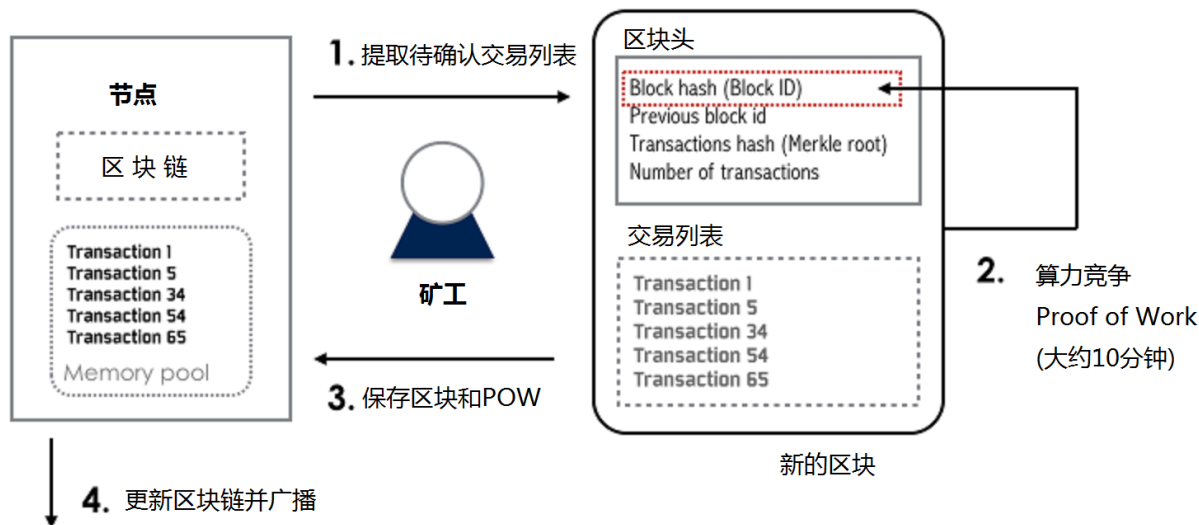
- 解决记账权

➤ 场景：

- 开放系统，节点海量

➤ 解决方案

- 工作量证明（proof of work）：高强度哈希计算（SHA256）进行算力竞争解决记账权，达成共识。



SHA256算力竞争

- I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
- I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
- I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
- I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
- I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
- I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
- I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
- I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
- I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
- I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
- I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
- I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
- I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
- I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
- I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
- I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
- I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
- I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
- I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
- I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...



区块头代码

class CBlockHeader {} //这是一个“类”，或数据机构，内含下面这些字段：

] int32_t nVersion //所采用Bitcoin协议的版本号

] uint256 hashPrevBlock //上一个块的（块头）Hash值。

] uint256 hashMerkleRoot //所记载交易记录的TxID，
//即其Hash值所构成Merkle树的根

] uint32_t nTime //本块的发布时间

] uint32_t nBits //为挖矿过程设置的难度，Hash值中须有的前导零的位数。

] uint32_t nNonce //在挖矿过程中使Hash值达到nBits字段所
//要求前导零位数的试凑值。



工作量证明的难度设定和调整

- $\text{New Difficulty} = \text{Old Difficulty} * (\text{Actual Time of Last 2016 Blocks} / 20160 \text{ minutes})$
- 例如:
 - `target`
`=0x00000000000000003A30C000`
- 在每个完整节点中独立自动发生的。每2,016个区块中的所有节点都会调整难度。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，即这些区块以10分钟一个速率所期望花费的时长）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（或变难或变易）。简单来说，如果网络发现区块产生速率比10分钟要快时会增加难度。如果发现比10分钟慢时则降低难度。



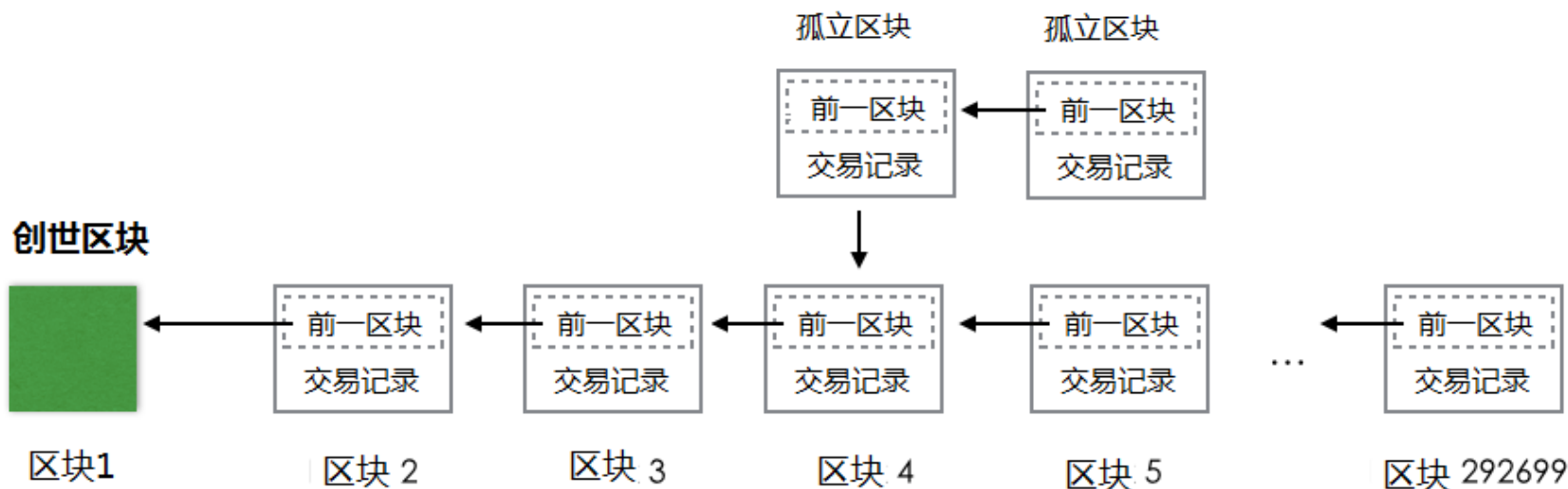
区块生成和组装

- 竞争胜出的节点创建区块并广播
- 其他独立节点校验新区块
- 区块链的组装和选择（根据父区块hash值查找父区块）
 - 连接到主链上
 - 分叉（备用链）
 - 孤立区块



区块链分叉解决方案

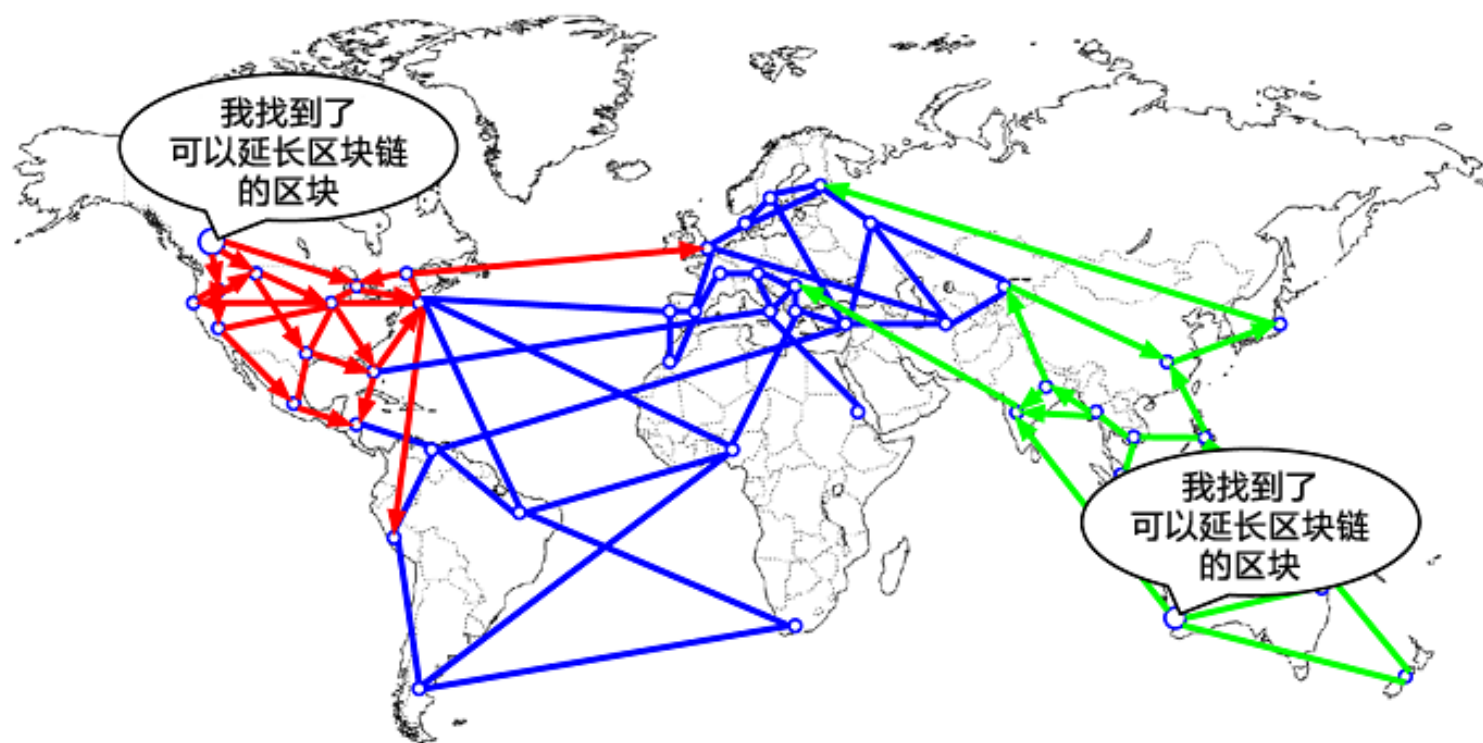
- 把当前同一个父区块下的若干有效子区块都记录，形成兄弟区块（产生分叉）
- 后续区块（第3代、第4代……）到达后，依次加在前序区块后，若没有其他竞争性区块，这一分支最长，成为主链。



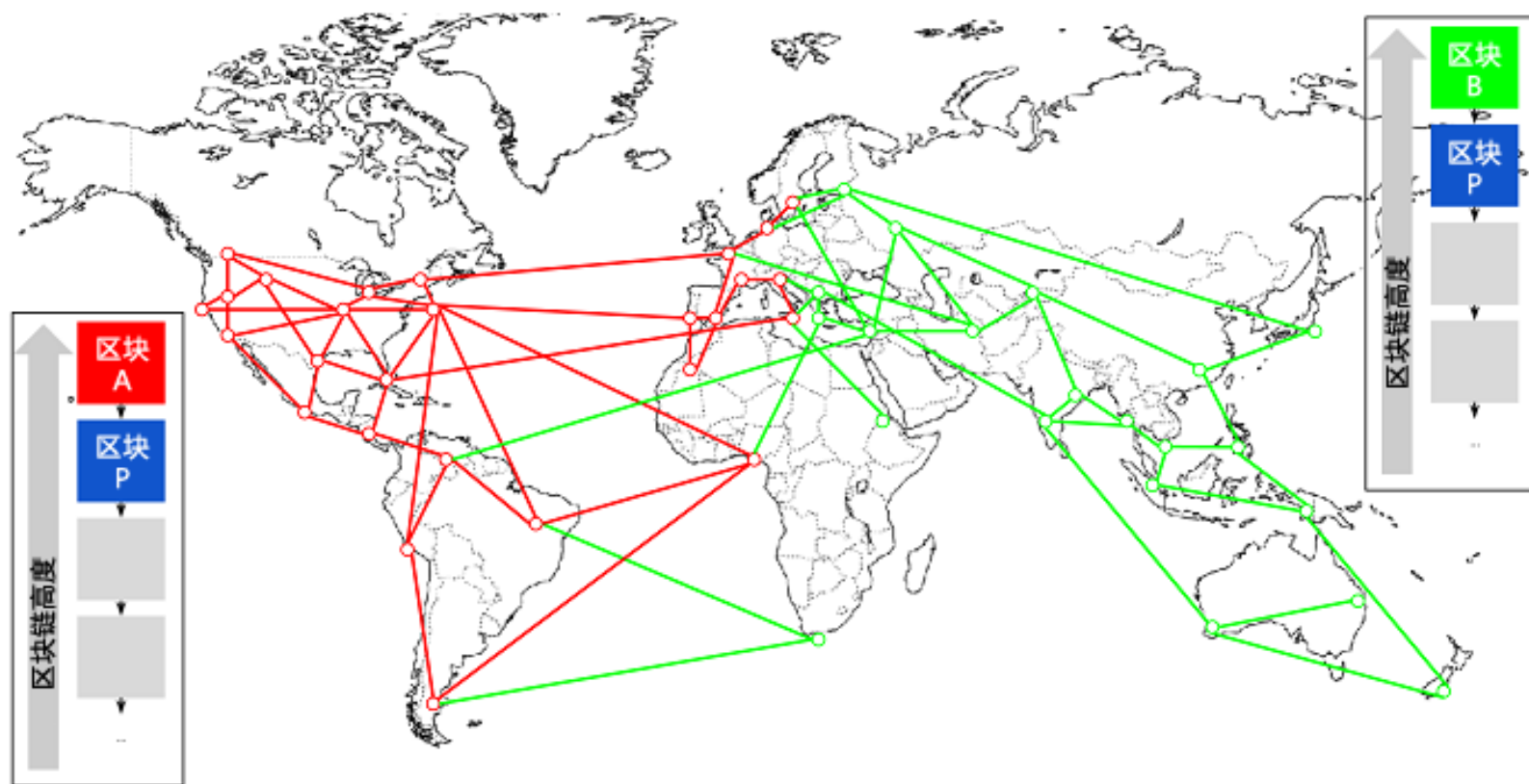
区块链分叉



区块链分叉



区块链分叉

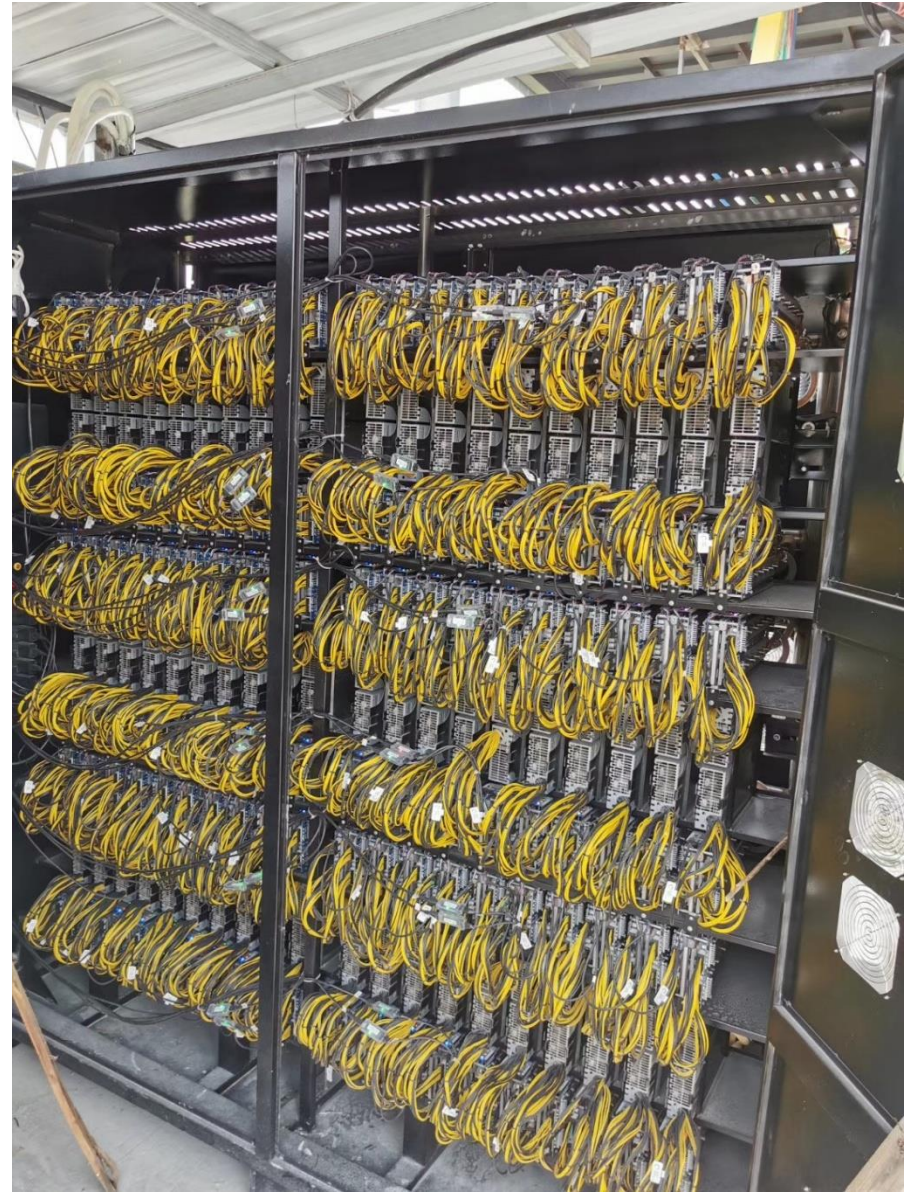


挖矿：比特币的产生

- 挖矿节点必须有钱包功能
 - 有自己的160位密码地址、私钥
- 打包生成区块时，区块中额外加一个交易coinbase
 - 生成一个UTXO，包含当前奖励数量的比特币（现在是6.25）
 - 这个UTXO的招领地址是自己的地址
 - 如果记账成功，这个coinbase交易就生效，否则不在链上。



挖矿



挖矿：算力竞争

- 2009 0.5 MH/sec – 8 MH/sec (16× growth)
- 2010 8 MH/sec – 116 GH/sec (14,500× growth)
- 2011 16 GH/sec – 9 TH/sec (562× growth)
- 2012 9 TH/sec – 23 TH/sec (2.5× growth)
- 2013 23 TH/sec – 10 PH/sec (450× growth)
- 2014 10 PH/sec – 300 PH/sec (3000× growth)
- 2015 300 PH/sec–800 PH/sec (266× growth)
- 2016 800 PH/sec–2.5 EH/sec (312× growth))



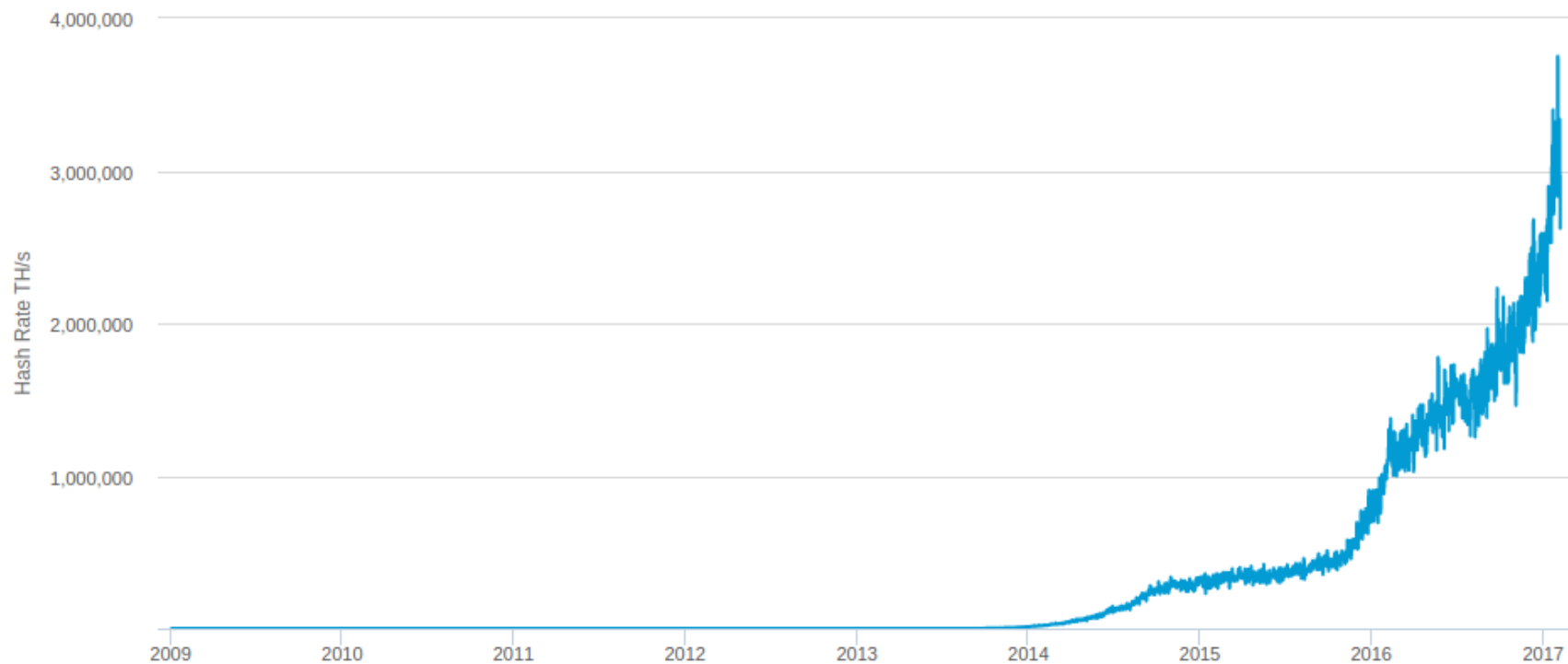
挖矿：算力竞争

Hash Rate

The estimated number of tera hashes per second (trillions of hashes per second) the Bitcoin network is performing.

Source: blockchain.info

Export ▾



共识记账机制小结

- 每个全节点依据综合标准对每个交易进行独立验证
- 通过完成工作量证明算法的验算，挖矿节点将交易记录独立打包进新区块，
- 每个节点独立地对新区块进行校验并组装进区块链
- 每个节点对区块链进行独立选择，在工作量证明机制下选择累计工作量最大的区块链



UTXO模型

- 每个比特币用户有一个160位（20字节）长度的地址，产生的过程：
 - 用户生产一对非对称密钥
 - 公钥经hash计算产生160位的地址
 - 私钥自己保存，用于数字签名
- 这个160位地址，是用户在比特币网络中交易的唯一标识。
- 与一般的银行账户模型不同，比特币不维护每个账户的资金余额，而是采用一种称为UTXO的模型。

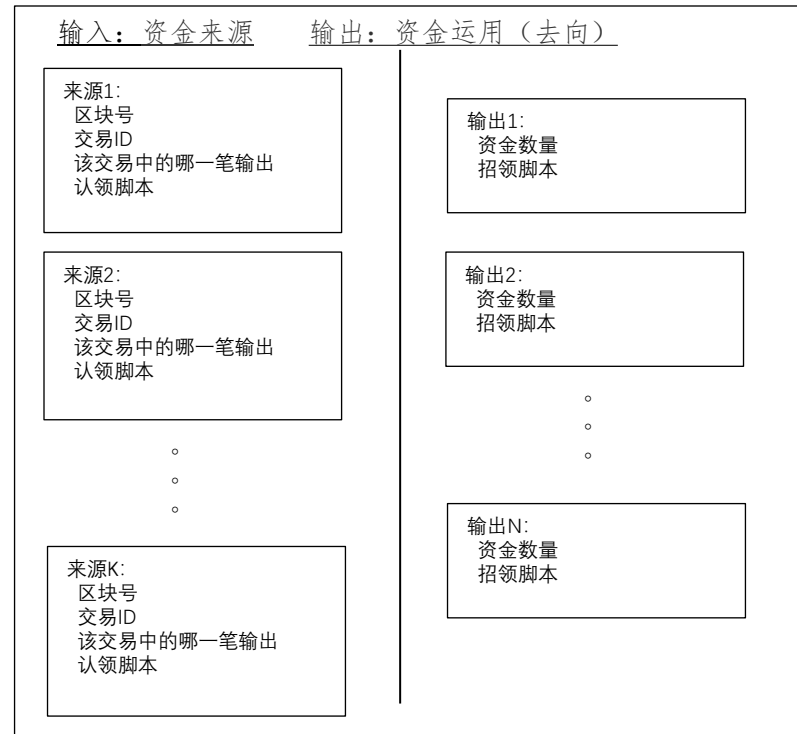


UTXO模型

- UTXO (Unspent Transaction Output), 未花费的交易输出
- 比特币中有两类交易：
 - 常规交易, 有交易输入 (支付者地址和金额)、交易输出 (收入者地址和金额)
 - 挖矿交易 (Coinbase), 产生比特币, 只有交易输出 (挖矿者地址和金额)
- 每个地址的资金余额就是散布在账本中所有UTXO的总和, 使用时把自己名下的UTXO作为交易输入, 可能需要拼凑找零。



UTXO模型



交易的数据结构

```

class CTransaction {} //一个交易请求或交易记录

] const std::vector<CTxIn> vin           //本Tx的输入UTXO序列，即资金来源。
] const std::vector<CTxOut> vout        //本Tx的输出UTXO序列，即资金去向。
] const int32_t nVersion                 // CURRENT_VERSION=2
] const uint32_t nLockTime               //锁定时间，时间未到点之前本交易不入块
。

] const uint256 hash                    //本Tx的Hash值，只存储在内存中，
                                        //不作永久存储也不发送。

```



CTxOut的结构

```
class CTxOut {}  
] CAmount nValue           //本项输出即该UTXO所承载的（比特币）价值  
] CScript scriptPubKey     //招领脚本，虽然名为scriptPubKey，  
                           //却未必只是以公钥为条件的脚本
```

| | |
|-------------|--------------------------|
| Value（支付金额） | scriptPubKey（招领脚本，即领用条件） |
|-------------|--------------------------|

P2PK

P2PKH

P2SH

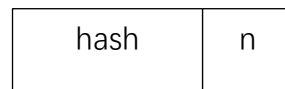
P2WPKH

P2WSH



CTxIn的结构

```
class CTxIn {}
] COutPoint prevout //Previous Output。指向具体的资金来源，
                    //说明来自那一项交易的第几项输出，展开如下：
]] uint256 hash      //资金来源所在Tx的Hash值，唯一地确定了一个Tx。
]] uint32_t n        //本项资金来源是该Tx中的第几项输出。
] CScript scriptSig  //为认领该项资金而提供的“签名脚本”（认领脚本）。
] uint32_t nSequence //具有特殊作用。
] CScriptWitness scriptWitness; //见证脚本的数据，0x00表示无。
```



本项资金来源所在交易记录的Hash值



本项资金来源是该交易记录中的第几项输出



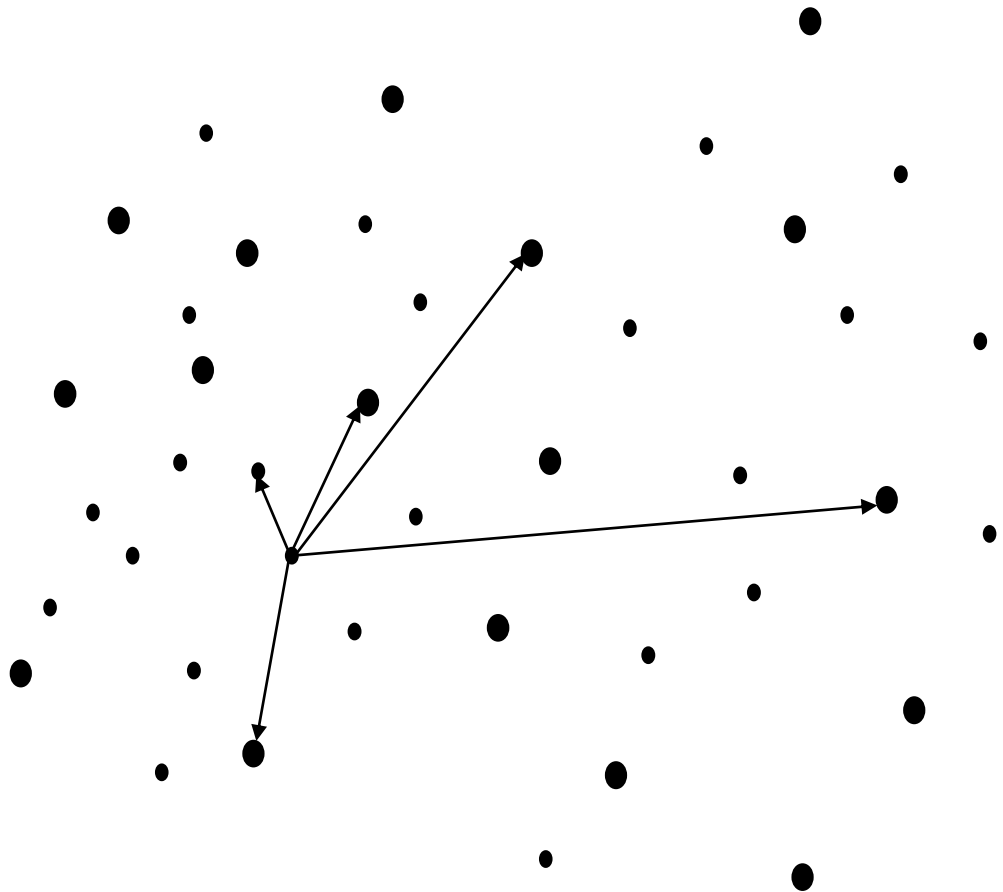
认领脚本scriptPubKey

- P2PK, “Pay to Public Key”, 付给公钥。付给给定256位公钥的主人。
- P2PKH, “Pay to Public Key Hash”, 付给公钥的Hash值。这里所谓的Hash, 是特指对于256位公钥的160位Hash, 那就是对方的“地址”。
- P2SH, “Pay to Script Hash”, 付给脚本的Hash值。给定一个脚本的Hash值, 付给能提供这个脚本的对象, 所提供脚本的Hash值必须与给定的Hash值相同。这个脚本是收付双方预先约定的, 是双方在“链外”约定的。
- 需要多方签名并采用SegWitness脚本的支付, 具体又有两种:
 - P2WPKH, “Pay to SegWitness Public Key Hash”, 付给SegWitness形式的多个公钥Hash值。
 - P2WSH, “Pay to SegWitness script Hash”, 付给SegWitness形式的脚本Hash值。



比特币区块链的网络通信

- P2P通信
 - 种子节点
 - 伙伴列表
 - 发现伙伴



比特币区块链中的节点

- **全节点 full client:** 存储着整个区块链，承担对交易请求进行验证和执行，可以通过挖矿争取发布区块，还承担着应别的节点之请向其发送区块和相关交易信息的义务，同时也承担转发交易请求和区块的义务。
 - **矿工节点**
 - **非矿工节点**
- **轻节点 light client**
 - **简单支付验证 (SPV) 节点**
 - 只存储区块头，不存储区块块体，仍可以对到来的交易请求进行验证。
 - **钱包**
 - 一个连接区块链的应用软件(app)，记录与所有者有关的信息：区块链地址、私钥、账户余额、UTXO等，不存储账本。



SPV（钱包）验证过程

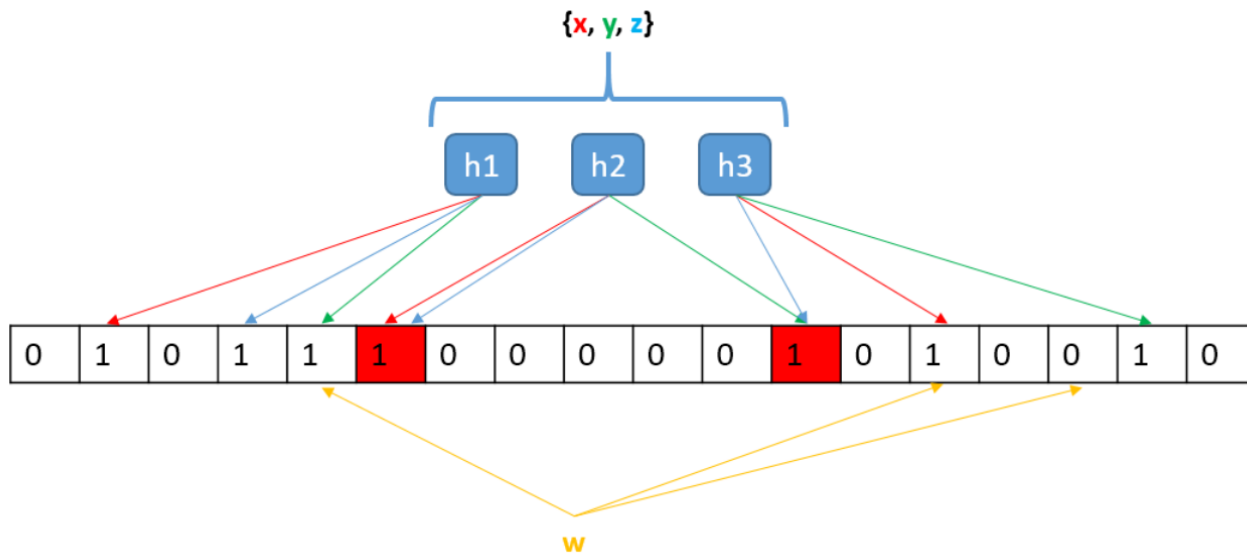
- 针对某个支付到自己比特币地址的交易建立布隆过滤器，限制只接收含有目标比特币地址的交易。
- 其他全节点探测到某个交易符合SPV节点设置的布隆过滤器条件时，以Merkleblock消息的形式发送该区块，Merkleblock消息包含区块头和一条连接目标交易与Merkle根的Merkle路径。
- 交易的存在性验证：SPV节点通过该Merkle路径找到跟该交易相关的区块，并验证对应区块中是否存在目标交易（**Merkle Path Proof**）。
- 交易是否双化验证：SPV节点检查这笔交易所在区块之后的区块个数，区块个数越多说明该区块被全网更多节点共识，一般来说，一笔交易所属区块之后的区块个数达到6个时，说明这笔交易是可信的。



布隆过滤器 (Bloom Filter)

布隆过滤器是一个基于 m 位的比特向量 (b_1, b_2, \dots, b_m)，这些比特向量的初始值为0。同时还有一系列的哈希函数 (h_1, h_2, \dots, h_k)，这些哈希函数运算后的哈希值范围在 $[1, m]$ 内。

判断结果：某个元素不在集合中 | 可能在集合中。

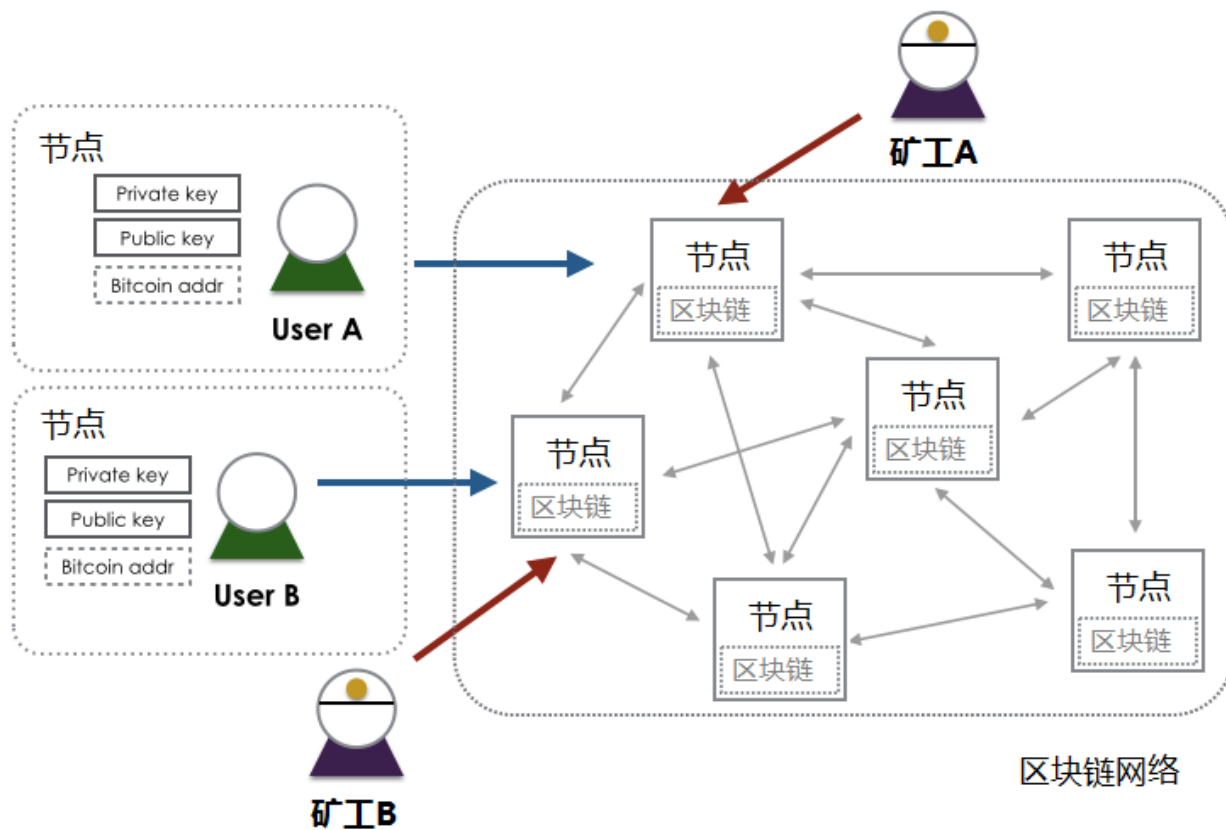


x 、 y 、 z 三个元素插入到布隆过滤器中，并判断 w 值是否在集合中的示意图。



区块链技术的定义

- 区块链技术是一种以非对称加密技术对交易进行数字签名，通过工作量证明等共识机制进行记账节点协调，数据以链式区块形式组织存储的分布式账本技术。



区块链支撑技术

- 非对称加密与数字签名
- 哈希计算：SHA256算法
- 链式区块结构
- Merkle树
- 共识机制
 - 拜占庭将军问题 (Byzantine Generals Problem)
 - PoW, PoS, DPoS
 - PBFT、Raft



区块链技术的优点

- 去中心化：避免垄断，点对点交易，去代理
- 数据公开：无暗箱操作，平等性，开放生态体系
- 可信：数据永久可靠，记录可信
 - 信任体系、价值传递体系



比特币区块链的问题

➤ 隐私问题

- 匿名地址如何监管

➤ 性能问题

- 交易确认时间长
- 区块容量有限

➤ 系统性风险

- 区块链分叉和51%攻击



支付通道

- 用于链外支付或其他交易
- 面向性能要求高、交易数量大等场景
 - 第一个支付通道应用场景：阿根廷有线电视比特币付费，按秒计费，如果每秒计费交易上链，比特币区块链无法满足性能要求，也会面对海量交易压力。
- 涉及的链上交易：
 - 注资交易 (Funding Transaction)
 - 应承交易 (Commitment Transaction)
 - 决算交易 (Settlement Transaction)
 - 退款交易 (Refund Transaction)



支付通道

- Funding transaction, 建立支付通道

- A、B双方建立一个2-of-2多签名的联合地址，由拟议中的付方发布一个交易将一笔钱打到这个地址中；但这是P2SH支付，即支付给能够正确提供清算脚本Hash值的收款方，因为是2-of-2就必须有双方的签名才能花，。这是后面链外支付的资金来源。
- 1个付款方：单向通道，2个付款方：双向通道

- Commitment transaction

- A、B双方的链下交易，可以有很多个，不上链。每次支付的资金都来自同一个UTX0，每次交易的输出分成两部分，一是给收方的UTX0，其数值是付方至此为止承诺支付的总和，二是给付方自己的找零。

- Settlement transaction

- 收款方把手中由付款方开具并签名的最后那个应承交易上签上自己的名并把它发送到比特币网上，从当初注资阶段生成的那个UTX0中把钱划给自己，同时把剩余的钱（如果还有的话）找还给付款方。这个操作也可以由付款方发起，因为收款方每次收到应承交易都会签上自己的名并发还给付款方。



闪电网络（ Lightning Network ）

- 是一种支付通道，除了支付通道上述交易外，还有：
 - 转存交易（Delivery Transaction），任何一方都可以从联合地址认领属于自己的资金，把它转到一个自己更方便花费的地址中。
 - 可撤转存交易（Revocable Delivery Transaction），可以撤销。
 - 补救交易（Breach Remedy），这是在对方违约情况下加以补救并使对方受惩罚的交易。



作业1

- 提交一个SHA256的代码实现，并用实验数据说明SHA256在比特币区块链中发挥的作用。
- 提交格式：word文件
- 文件名：姓名+学号+作业1.docx
- 提交邮箱：sherlin@zju.edu.cn
- 提交截止日期：2020年12月7日9点



深入阅读

- 比特币白皮书
- 比特币源码
- 《精通比特币》，Andreas M. Antonopoulos
- *Mastering Bitcoin*, Andreas M. Antonopoulos



谢谢！

