区块链技术与应用

计算机科学与技术学院　李京

02章 区块链数据层

目 录

2.1 区块结构

- 数据区块是区块链的基本元素，是一种记录交易的数据结构。每个区块由区块头和区块体组成，区块体只负责记录前一段时间内的所打包交易信息，区块头记录当前区块的元数据。
- 区块类似于账本中的账页，其物理存储形式可以是文件（如比特币），也可以是数据库（如以太坊）。
- 创始区块，第一个区块。
- 主流区块链平台的区块结构大同小异，后续以比特币系统为例。

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(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

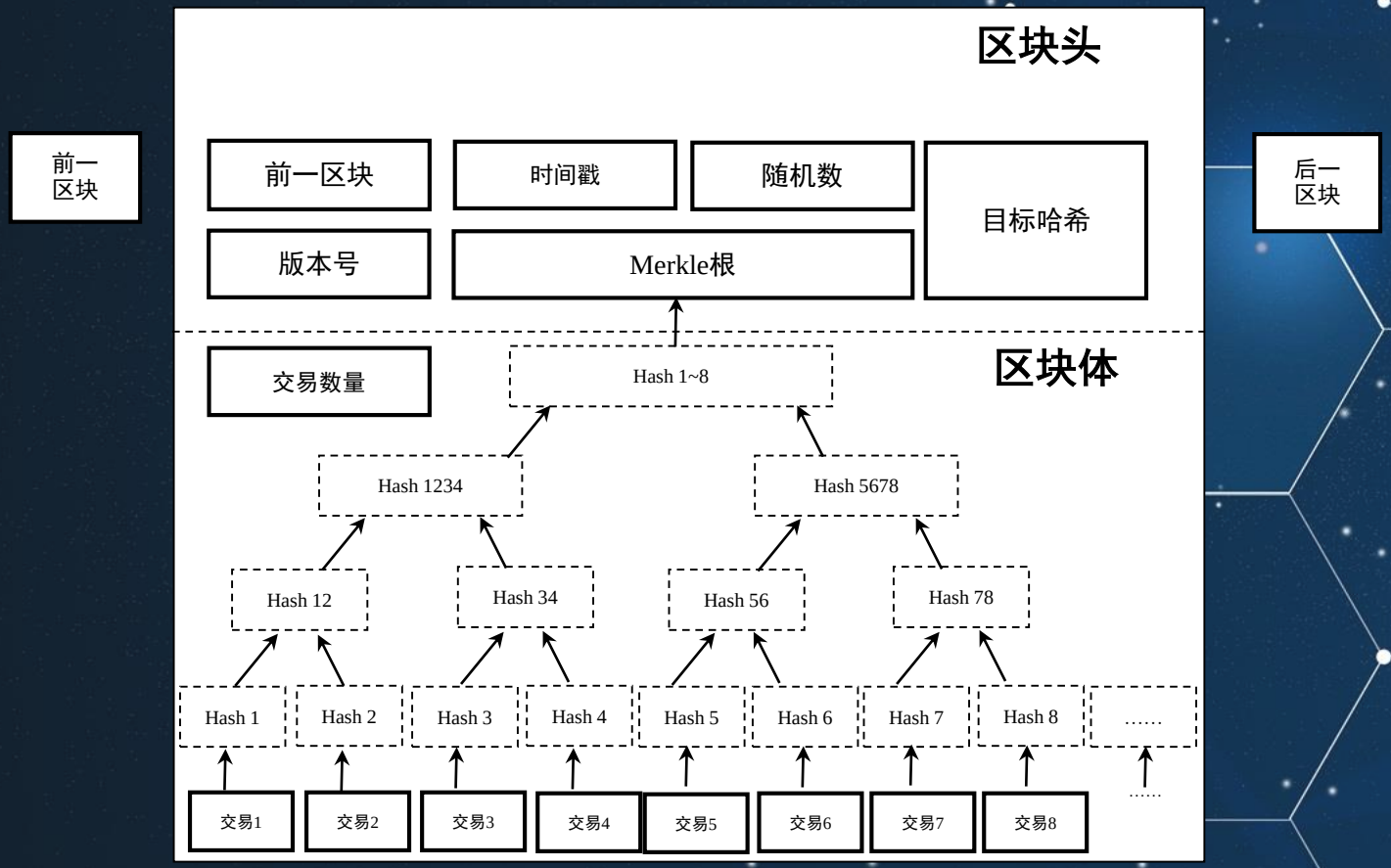
创世区块

比特币的创世区块创建于2009年1月3日

中本聪在位于芬兰赫尔辛基的一个小型服务器上挖出了比特币的第一个区块

CoinBase交易中写入了一个附加信息，即“The Times 03/Jan/2009 Chancellor on brink of second bailout for banks”

- 区块链就是将区块按某种方式链接在一起形成的账本。



比特币系统的区块结构

区块头

数据项	字节	字段	说明
Version	4	版本	区块版本号，表示本区块遵守的验证规则
Pre-block	32	父区块头哈希值	前一区块的哈希值，使用SHA256(SHA256(父区块头))计算
Merkle-root	32	Merkle根	该区块中交易的Merkle树根的哈希值，同样采用SHA256(SHA256())计算
Timestamp	4	时间戳	该区块产生的近似时间，精确到秒的UNIX时间戳，必须严格大于前11个区块时间的中值，同时全节点也会拒绝那些超出自己2个小时时间戳的区块
Bits	4	难度目标	该区块工作量证明算法的难度目标，压缩格式的当前目标哈希值
Nonce	4	Nonce	32位数字（从0开始），为了找到满足难度目标所设定的随机数

比特币系统的区块头主要封装了当前版本号(Version)、前一个区块的地址(Pre-block)、Merkle根(Merkle-root)以及时间戳(Timestamp)、当前区块的目标哈希值(Bits)、当前区块PoW共识过程的解随机数(Nonce)等信息。这些信息大体上可以分为三类:

- 引用父区块哈希值的数据Pre-block，将当前区块与前一区块相连，形成一条起始于创世区块且首尾相连的区块“链条”;
- 当前区块链所有交易经过哈希运算后得到的Merkle根，指向区块体所封装的交易;
- 由目标哈希值、时间戳与随机数组成，这些信息都与共识竞争相关，是决定共识难度或者达成共识之后写入区块的信息。

区块头

Block Height 558859 Blocks at depth 558859 in the bitcoin blockchain

Summary	
Height	558859 (Main chain)
Hash	00000000000000000002cb4fbb24ed9d2dd725f57a0f7d27cb795acc533fda762
Previous Block	00000000000000000000604670e69a340956bb16d967281921001ce9a4461fae7
Next Blocks	00000000000000000001700b1d88497bfa56ea3023aee59c6a6b2cd3ede731578
Time	2019-01-17 04:43:11
Difficulty	5,883,988,430,955.41
Bits	389010995
Number Of Transactions	2465
Output Total	4,326.55321847 BTC
Estimated Transaction Volume	622.8768482 BTC
Size	1033.844 KB
Version	0x20000000
Merkle Root	e82ed4d49fc2890b04405534997d22355c406d76a6d0db999c76db1878b4881a
Nonce	572760437
Block Reward	12.5 BTC
Transaction Fees	0.13823693 BTC

图 2-2　比特币区块＃558859 的基本信息

- 区块的主标识符是区块头的哈希值，是使用两次SHA256哈希算法之后的结果，可以唯一标识一个区块。
- 区块高度，将区块链看成一个垂直的栈。也常用来标识一个区块，但可能不唯一。

区块体

- 区块体包含了当前区块的交易数量和经过验证的、区块创建过程中生成的所有交易记录。
- 交易是区块链网络中传输的最基本的数据结构，所有有效的交易最终都会被封装到某个区块中，存于区块链上。下表是比特币交易的数据结构

数据项	数据描述	大小
Version No	版本号，目前为 1，表示这笔交易参照的规则	4字节
In-counter	输入数量，正整数 VI = VarInt	1-9字节
list of inputs	输入列表，每区块的第一个交易称为 “Coinbase” 交易	<in-counter>-许多输入
Out-counter	输出数量，正整数 VI = VarInt	1-9字节
list of outputs	输出列表，每区块第一个交易的输出是给矿工的奖励	<out-counter>-许多输出
lock_time	锁定时间，如果非0并且序列号小于0xFFFFFFFF ，则是指块序号；如果交易已经终结，则是指时间戳	4字节

比特币交易示例（比萨交易）

著名的比特币“披萨交易”的JSON可视化格式示例

元数据：主要存放一些内部处理的信息，包括版本号、交易大小、输入的数量、输出的数量、交易锁定时间，以及标识该交易的哈希值。可以使用该哈希指针指向这个交易。

交易的输入列表。每笔交易的所有输入排成一个序列，每个输入的格式相同，被序列化成字节流在网上传播。

交易的输出列表。每笔交易的所有输出也排成一个序列。每个输出的内容分成两部分，一部分是特定数量的比特币，以“聪”为单位(最小的比特币单位)；另一部分是锁定脚本，即提出支付输出所必须被满足的条件以“锁住”这笔总额。交易的所有输出金额之和必须小于或等于输入金额之和。当输出的总金额小于输入总金额时，二者的差额部分就作为交易费支付给为这笔交易记账的矿工。

所示的交易的输入是从交易a1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fb5d48d的索引为0号的输出中导入了10000个比特币。该交易中输入的10000个比特币，在两个输出中分别发送5777和4223个比特币到相应的比特币地址。

UTXO(Unspent Transaction Outputs)

- 借助前一笔交易的哈希指针，所有交易构成了多条以交易为结点的链表，每笔交易都可一直向前追溯至源头的Coinbase交易(即挖矿过程中新发行的比特币)，向后可延展至尚未花费的交易。（可溯源）
- 如果一笔交易的输出没有任何另一笔交易的输入与之对应，则说明该输出中的比特币尚未被花费,这种未花费的交易输出称为UTXO。
- 通过收集当前所有的UTXO，可以快速验证某交易中的比特币是否已被花费。
- 通过收集某人所有地址的UTXO，可以统计他所拥有的比特币数量。

例如使用100个比特币消费50个比特币，将会产生两个输出，一个50比特币发往接收方地址，另一个50比特币以“找零”方式发往发送方的某个地址。

UTXO：未使用的交易输出，比特币核心概念之一

- 比特币系统中其实并不存在“账户”，而只有“地址”（钱包）。可以在比特币区块链上开设无限多个钱包地址，某人所拥有的比特币数量是其所有钱包地址中比特币的总和。比特币系统并不会把某人的这些地址汇总起来形成其的账户。比特币甲到乙的转账，就是从甲的一个钱包地址转到乙的一个钱包地址上去。
- “其实没有什么比特币，只有UTXO”，对于计算机来说比特币是区块链账本上的交易输出。
- UTXO 的设计可以看成是借鉴了现金的思路。
- 一个转账交易过程是：用发起方私钥（从一个输出是发送方地址的交易中上一个UTXO）取出比特币，并用私钥对新交易进行签名。一旦交易完成，这些比特币就转到接收方的钱包地址中去。接收方钱包中新交易的未使用交易UTXO输出，只有接收方的私钥才可以打开。

UTXO的优点

- UTXO设计易于确认比特币的所有权，可以让双重花费更容易验证。

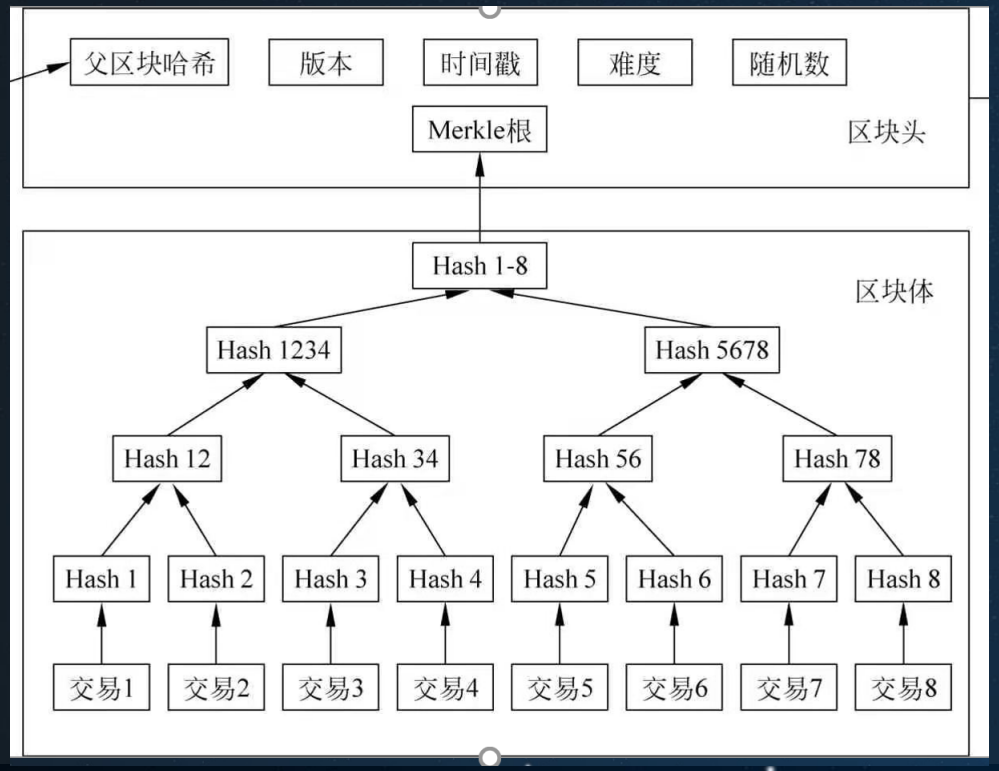
采用传统账户设计，当要转账8个比特币出去时，为了完全避免造假，需要逐一向上追溯，确认之前的每一笔交易，从而证明该账户的确拥有8个比特币。采用现在的UTXO设计，要确认确实拥有8个比特币，只要确认上一个交易的确获得了它们即可。通常只要上一个交易是真实的，就的确拥有这些比特币。而比特币系统中的交易可被认为是真实无误的。采用 UTXO 设计，验证双花只要沿着每个交易的输入逐级向上核查，直到查到这笔比特币的创币交易即可。

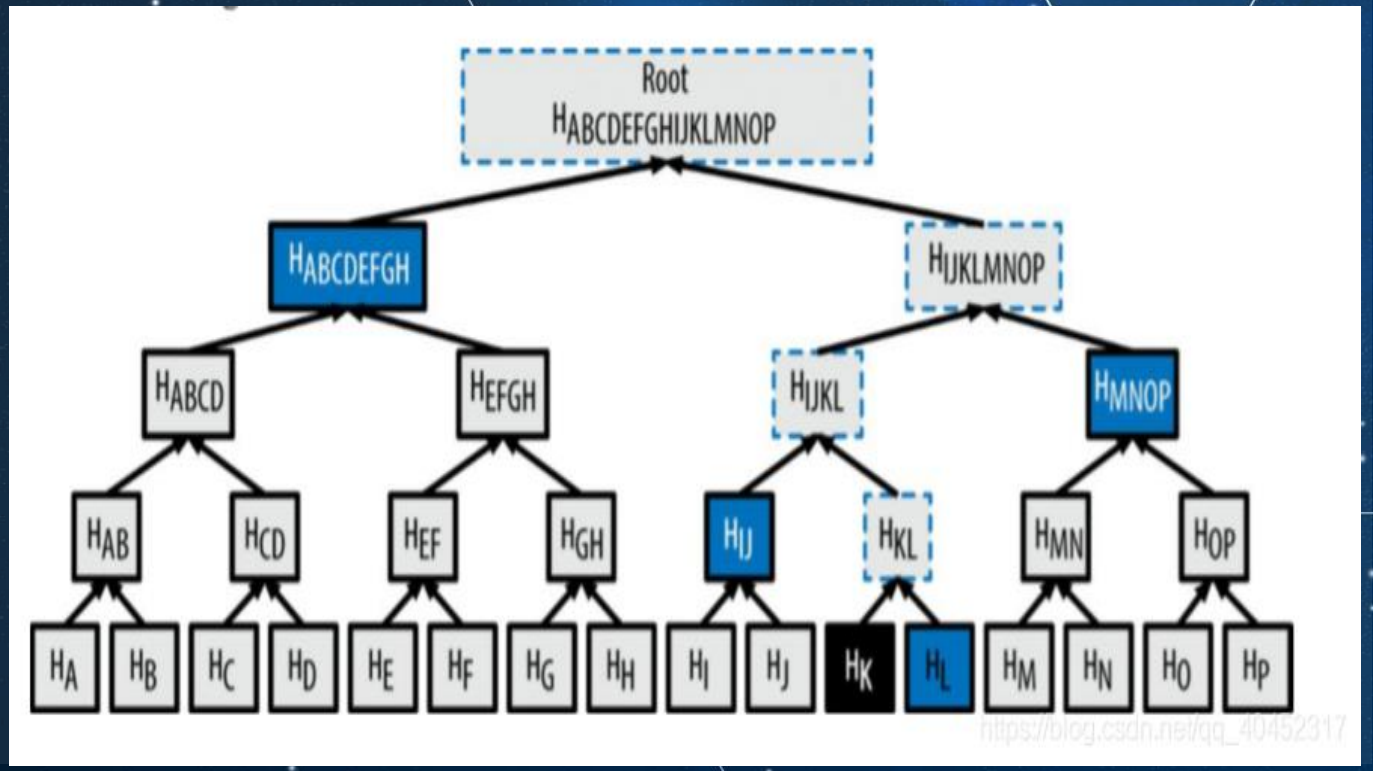
- UTXO设计与区块链账本是完全融为一体的

区块链账本存储的是状态。微观地看，每一个区块链中的交易都是一个状态转换函数，每一个新区块和它之前的所有区块一起形成了一个新的状态，如此重复、持续下去。在确认之后，之前的状态就不可篡改，即不可随意更改。UTXO（未使用的交易输出）是与这种状态的设计相对应的。

Merkle树：交易存储的组织

- 比特币系统采用二叉默克尔树来组织每个区快中的所有交易。
- 可以使用默克尔路径快速校验某个区块中是否有特定的交易。
  - 例如某轻客户端要验证Hk是否是某个区块中的交易，只需向其他全节点申请蓝色节点的值即可。

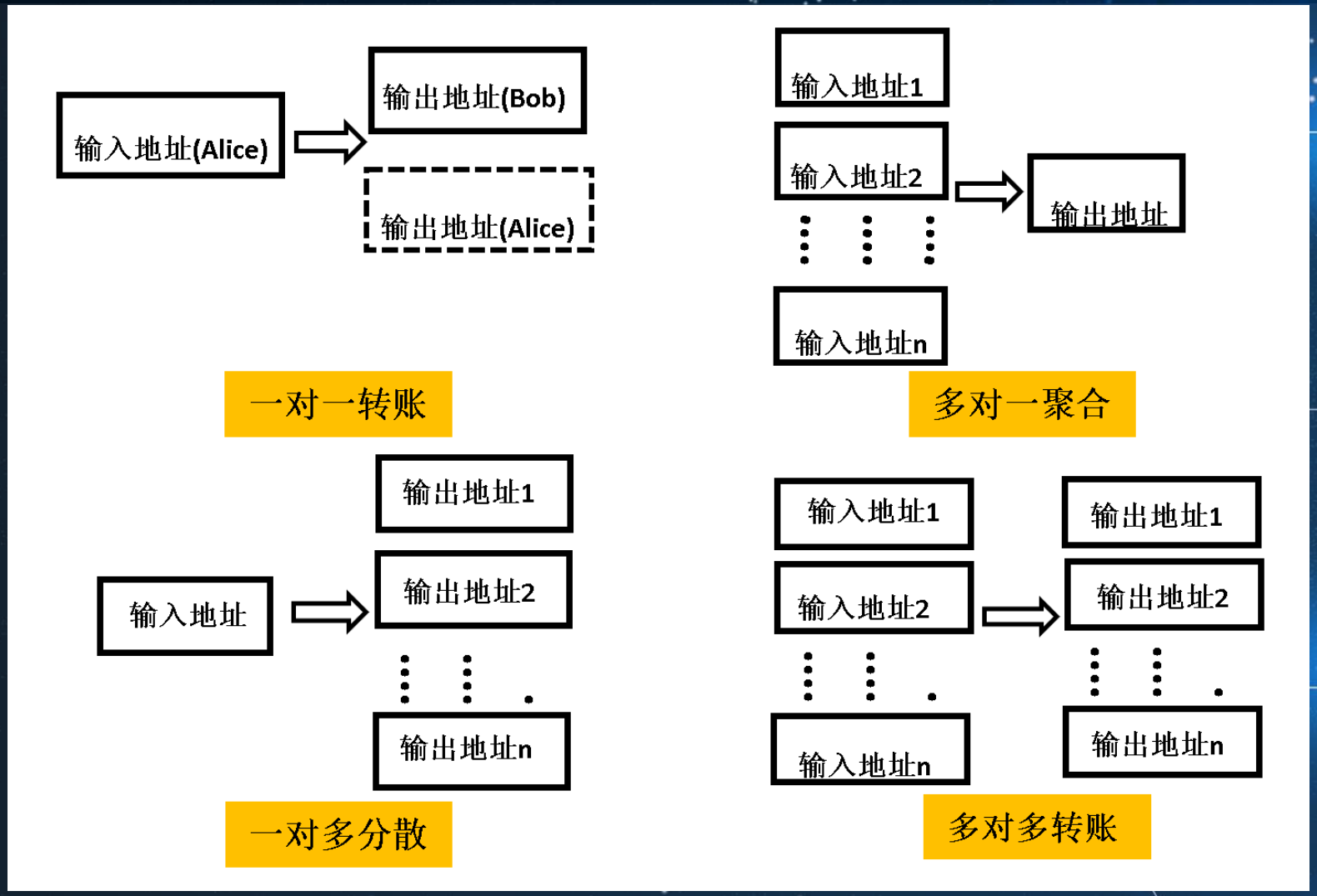




交易类型

- 生产交易：又叫coinbase交易，每个区块的第一笔交易都是生产新币的交易。该交易没有输入地址，仅有个输出地址，其作用是将系统新生成的加密货币奖励给创造当前区块的矿工。
- 通用地址交易：区块链系统中最常见的交易，由N个输入和M个输出构成，其中N,M>0。根据N和M的不同取值，可以进一步细分为一对一转账交易、一对多分散交易、多对一聚合交易和多对多转账交易。
- 合成地址交易：合成地址交易是一类特殊交易，其接收地址不是通常意义的地址，而是一个以3开头的合成地址。

通用交易示例


输入地址(Alice)
输出地址(Bob)
输出地址(Alice)
一对一转账
输入地址1
输入地址2
输入地址n
输出地址
多对一聚合
输入地址
输出地址1
输出地址2
输出地址n
一对多分散
输入地址1
输入地址2
输入地址n
输出地址1
输出地址2
输出地址n
多对多转账

合成地址交易

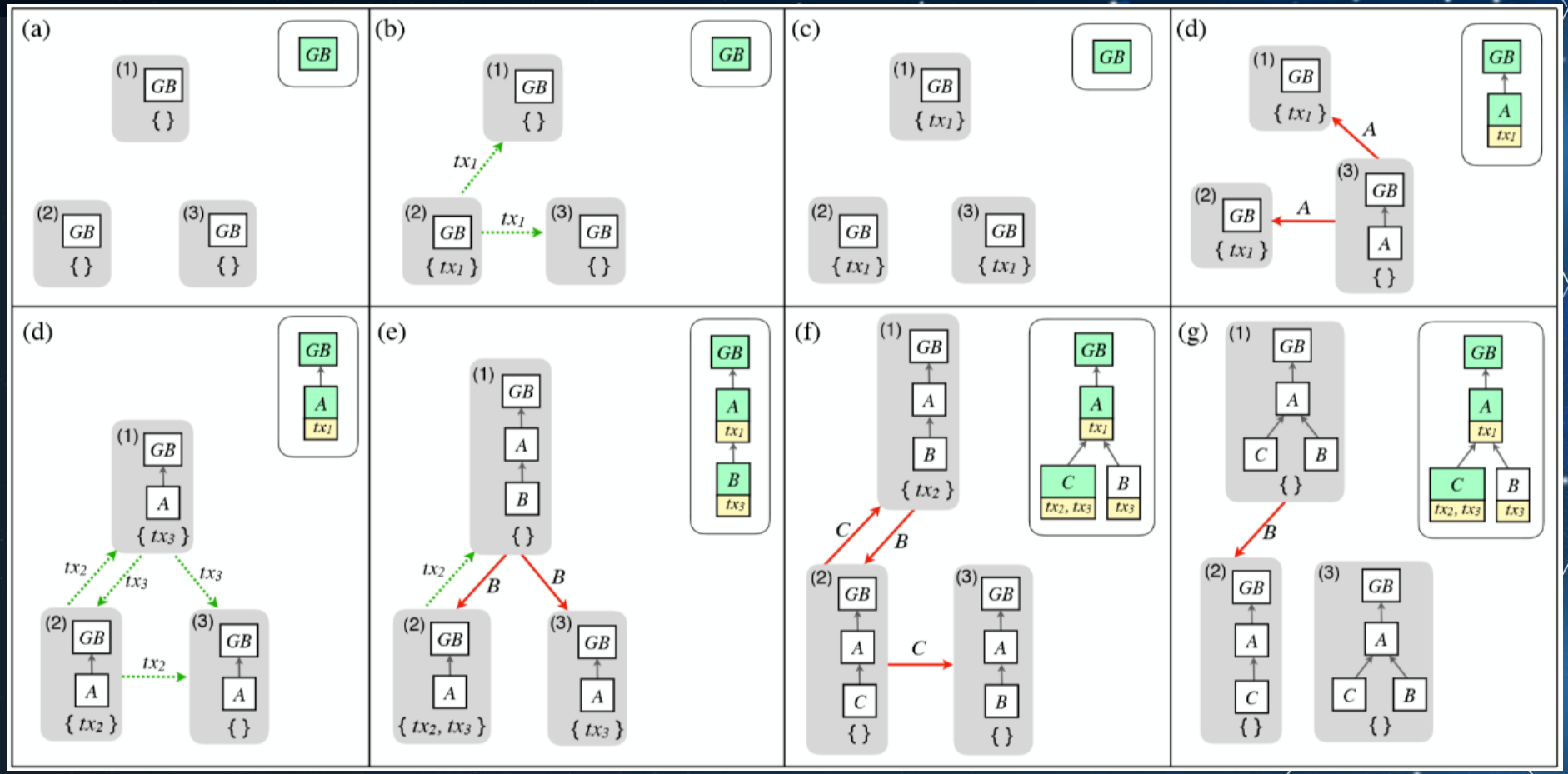
- 合成地址一般是M of N模式的多重签名地址，其中1≤N≤3、1M≤N，通常选择N=3。
- 合成地址的交易构造、签名和发送过程与普通交易类似，但其地址创建过程需要三对公钥和私钥，其中公钥用于创建地址、私钥用于签名。例如:
  - (1)如果M=1且N=3，则3个私钥中任意1个都可以签名使用该地址上的币，这种私钥冗余可防止私钥丢失，即使其他2个私钥丢失也不会造成损失。
  - (2)如果M=2且N=3，则3个私钥中必须有2个同时签名才可使用该地址的币，常见于三方中介交易场景。
  - (3)如果M=N=3，则必须3个私钥同时签名才可使用该地址的币，常见于多方资产管理场景。

2.2 区块链的运行流程

- 以比特币系统为例，讲解区块链运行过程中数据结构的处理要素。交易的步骤一般如下：

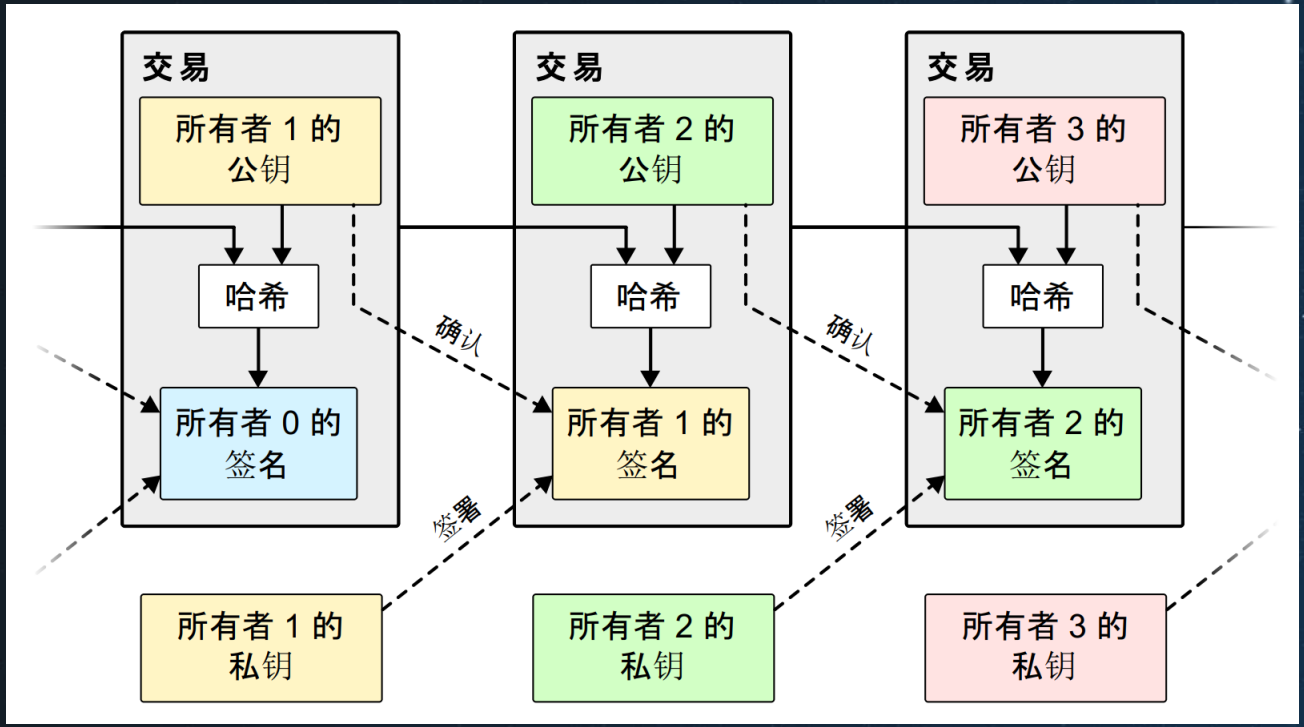
①源节点创建交易并验证目的节点的地址；
②源节点对交易进行签名加密；
③源节点将该交易广播至全网其他节点；
④全网节点接收交易并验证其有效性，直到该交易被全网大多数节点验证和接受；
⑤交易被暂存于节点内存池，并判断是否为孤立交易；
⑥交易被打包至节点本地区块中；
⑦全网共识结束后，获胜节点将其本地区块追加到主链；
⑧交易在主链上被越来越多的后续区块确认。

比特币系统交易的主要环节（3节点示例）



四个主要环节：交易生成、网络传播与验证、共识出块和激励分配

交易生成

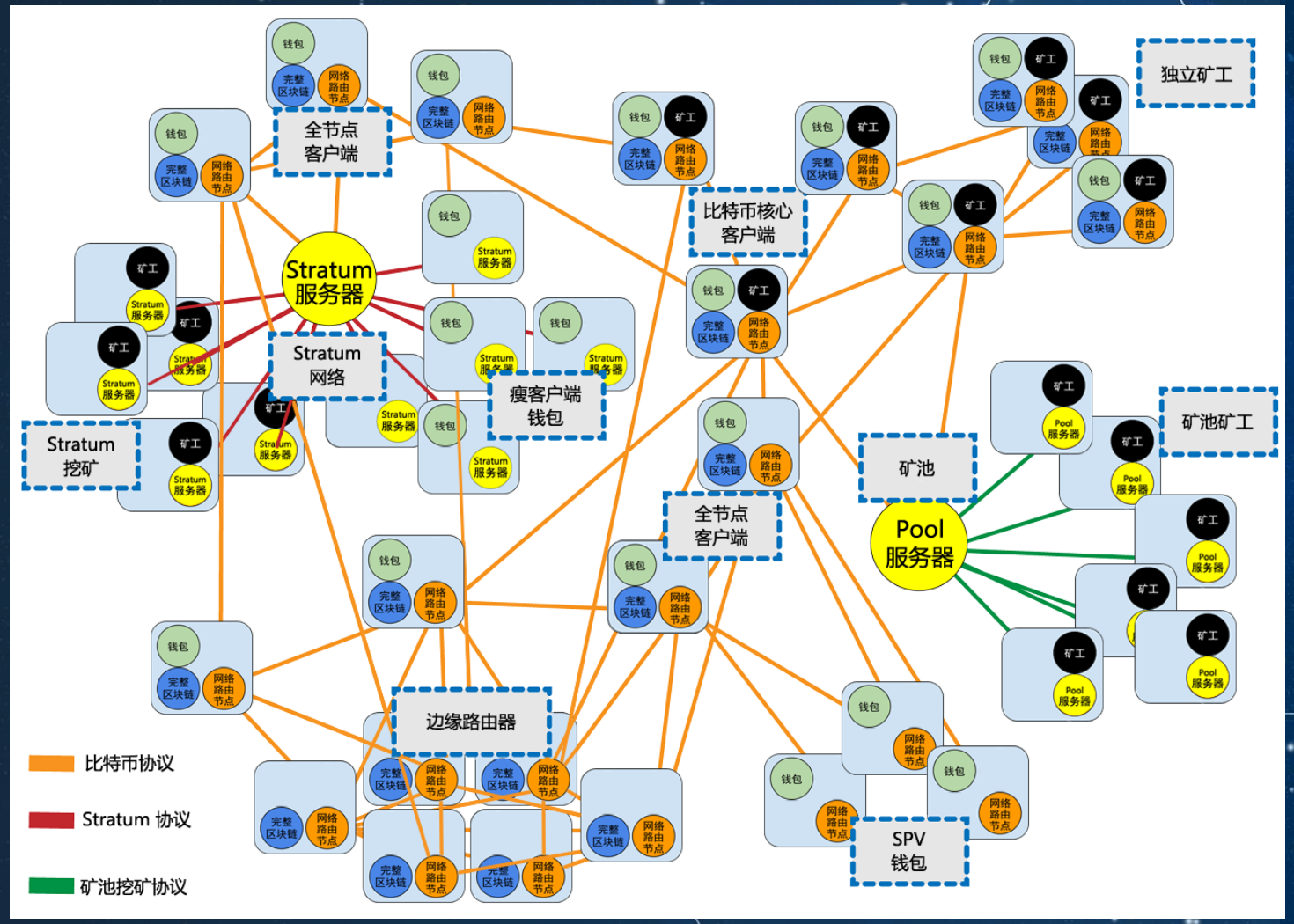


比特币系统的交易链结构

- 源节点创建交易，将目的节点的公钥作为交易的参数，使用自己的私钥对新交易签名。
- UTXO

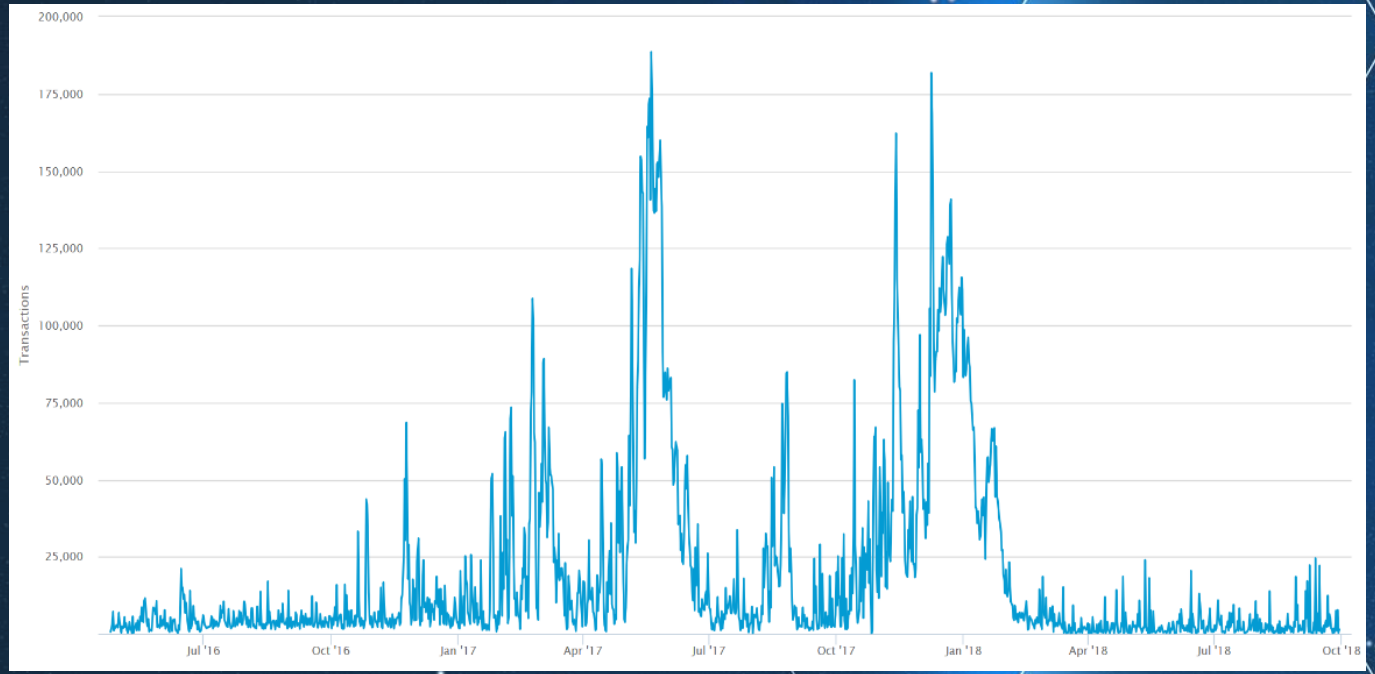
网络传播与验证

- 比特币网络是P2P网络，使用Gossip协议进行交易的传播。
- 每个节点收到交易后都会独立对其有效性进行验证，验证通过后才会中继转发到其他节点。
- 通过验证环节，有效抵御了恶意交易、垃圾信息的传播和拒绝服务攻击



区块链的运行流程-交易池管理

- 交易池：一个内存池用于存放待确认打包的有效交易。
- 孤立交易池：暂时存放缺失父交易的子交易。
- 交易池导致的交易拥堵和低手续费交易不能及时确认。



比特币系统的待确认交易数量（2016.04~2018.10）
2017年5月和12月待确认交易量峰值（>18万）

交易优先级排序和手续费定价

- 比特币系统的交易优先级公式：

$$\text{交易优先级} = \frac{\sum_{\text{每个输入对应的UTXO}} (\text{UTXO交易额} \times \text{UTXO存在时间})}{\text{交易的字节长度}}$$

$$\text{交易字节长度} = 148 \times \text{输入数量} + 34 \times \text{输出数量} + 10$$

- 比特币系统采用0.576作为交易的基准优先级，交易优先级低于该值则会被收费。（250字节长度、1比特币、存续时间1天）交易长度低于10000字节的交易会按每千字节收费，单价是0.0001个比特币。

共识竞争与构建区块

- 比特币采用工作量证明(Proof of Work,PoW)共识算法，其核心思想是通过引入分布式节点的算力竞争来保证数据一致性和共识的安全性。
- 各矿工节点基于各自的计算机算力相互竞争来共同解决一个求解复杂但验证容易的SHA256数学难题(即挖矿)，最快解决该难题的节点将获得区块记账权和系统自动生成的比特币奖励。
- 该数学难题可表述为：根据当前难度值，通过搜索求解一个合适的随机数(Nonce)使得区块头各元数据的双SHA256哈希值小于或等于目标哈希值。
- 比特币系统通过灵活调整随机数搜索的难度值来控制区块的平均生成时间为10分钟左右。

共识竞争与构建区块

- PoW共识中每个矿工重复执行以下步骤1至步骤4，最快搜索到符合要求的随机数Nonce的矿工获胜并取得记账权。

步骤1：搜集当前时间段的全网未确认交易，并增加一个用于发行新比特币奖励的Coinbase交易，形成当前区块体的交易集合；

步骤2：计算区块体交易集合的默克尔根记入区块头，并填写区块头的其他元数据，其中随机数Nonce置零；

步骤3：随机数Nonce加1；计算当前区块头的双SHA256哈希值，如果小于或等于目标哈希值，则成功搜索到合适的随机数并获得该区块的记账权；否则继续步骤3直到任一节点搜索到合适的随机数为止；

步骤4：如果一定时间内未成功，则更新时间戳和未确认交易集合、重新计算默克尔根后继续搜索。

难度和难度调整机制

- 难度是区块链系统(特别是PoW类型的公有链系统)的重要参数，用来度量矿工成功挖到下一个区块的难易程度。
- 符合要求的区块头哈希值通常由多个前导零构成，目标哈希值越小区块头哈希值的前导零越多，成功找到合适的随机数并“挖”出新区块的难度越大。
- 据区块链实时监测网站 Blockchainino显示，截止到2018年12月，符合要求的区块头哈希值一般有19个前导零(十六进制)。按照概率计算，每16次随机数搜索将会找到一个含有一个前导零的区块哈希值，因而比特币目前19位前导零的哈希值要求16的19次方次随机数搜索才能找到一个合适的随机数并生成一个新的区块。这使得比特币的共识过程必须付出巨大的算力投入，同时也使得恶意攻击者实施双花攻击时必须拥有极大的算力资源，从而保障了比特币系统的安全。

难度调整机制

Block Height 556459 Blocks at depth 556459 in the bitcoin blockchain

Summary	
Height	556459 (Main chain)
Hash	00000000000000000002479aed3082c1694f68173646a86a6e9b750009eb2ad32
Previous Block	00000000000000000000c351d88e917a179853750b57925cf6f181dc95421ec40
Next Blocks	00000000000000000000cb442b2944c5d176096c0e25c5fd8311e2dee166bff8e
Time	2019-01-01 00:03:10
Received Time	2019-01-01 00:03:10
Relayed By	BitFury
Difficulty	5,618,595,848,853.28
Bits	389159077
Number Of Transactions	946
Output Total	3,263.17177298 BTC
Estimated Transaction Volume	2,038.46010943 BTC
Size	728.024 KB
Version	0x20000000
Merkle Root	3351928351b35b828c2c998481cf953715c325899543b91edfa21a9bdb13e488
Nonce	1584471910
Block Reward	12.5 BTC
Transaction Fees	0.0524382 BTC

比特币区块难度

目标值的计算公式为：

$$Target = Coefficient * 256^{Exponent-3}$$

区块#556459的目标值为：0x173218a5

$$Target = 0x3218a5 * 256^{0x17-3} = 3283109 * 256^{20}$$

难度调整机制

Block Height 556459 Blocks at depth 556459 in the bitcoin blockchain

Summary	
Height	556459 (Main chain)
Hash	00000000000000000002479aed3082c1694f68173646a86a6e9b750009eb2ad32
Previous Block	00000000000000000000c351d88e917a179853750b57925cf6f181dc95421ec40
Next Blocks	00000000000000000000cb442b2944c5d176096c0e25c5fd8311e2dee166bff8e
Time	2019-01-01 00:03:10
Received Time	2019-01-01 00:03:10
Relayed By	BitFury
Difficulty	5,618,595,848,853.28
Bits	389159077
Number Of Transactions	946
Output Total	3,263.17177298 BTC
Estimated Transaction Volume	2,038.46010943 BTC
Size	728.024 KB
Version	0x20000000
Merkle Root	3351928351b35b828c2c998481cf953715c325899543b91edfa21a9bdb13e488
Nonce	1584471910
Block Reward	12.5 BTC
Transaction Fees	0.0524382 BTC

比特币区块难度

计算当前区块的难度值：

$$Difficulty_{当前区块} = \frac{Target_{创世区块}}{Target_{当前区块}}$$

$$Target_{创世区块} = 0x00ffff * 256^{0x1d-3} = 65535 * 256^{26}$$

区块#556459的难度计算公式为：

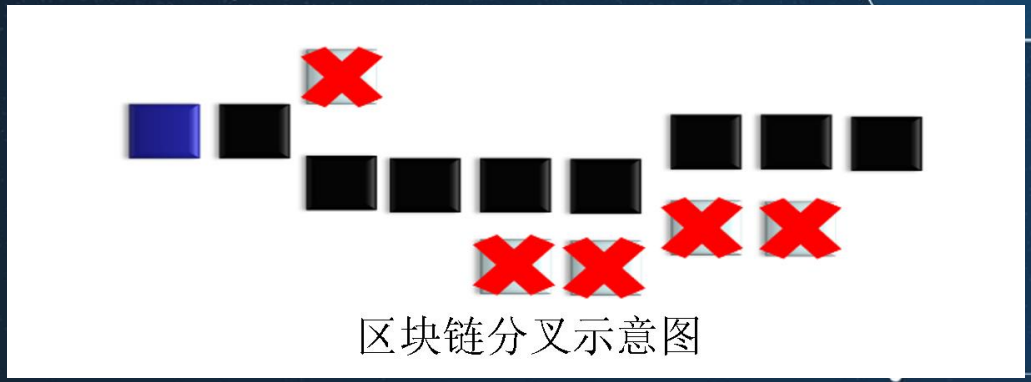
$$Difficulty_{556459} = \frac{0x00ffff*256^{26}}{0x3218a5*256^{20}} = \frac{65535}{3283109} * 256^{6} \approx 5618595848853.28$$

难度调整机制

- 新难度=旧难度×(过去2016个区块的实际时间/20160分钟)

分叉处理与主链判定

- 如果多个矿工节点在同一时间段内成功搜索到符合哈希结果要求的随机数，则这些矿工都将认为自己在共识竞争中获胜并向比特币网络中广播其构造的区块，从而产生在同一区块高度出现多个不同的有效区块的情况，即产生分叉现象。
- 为保证区块链系统中仅有唯一的主链，必须定义合适的主链判定准则来从多个分叉链中选择符合条件的唯一主链。此时不在主链上的区块将成为“孤块”，发现孤块的矿工节点也不会得到相应的比特币奖励。
- 由于孤块的存在,区块链的形状并非单一的“链条”，而是如下图所示的树状结构，其中每个共识轮次对应的时间点上仅有唯一区块是有效的，因而树状结构中也仅有唯一的主链条。



区块链分叉示意图

分叉处理与主链判定

- 比特币系统采用“最大工作量原则”作为消除区块链分叉时的主链判定规则，该原则可体现为多种形式，即当主链出现多个分支区块子链时的主链判定规则如下:

步骤1：如果不同分支的区块高度不同，则选择最长区块高度的分支为主链;

步骤2：如果高度一致，则选择难度系数最大的分支作为主链;

步骤3：如果高度和难度系数均相同，则选择接受时间最早的分支为主链;

步骤4：如果上述所有评判系数均相同，则等待新区块产生并连接到某个或者多个分支、区块高度增加后，重复步骤1-3直至选出主链。此时，生成新区块的节点即可对当前多个分支子链进行“投票”，并链接至最有可能成为主链的分支子链上。

2.3 数据层的关键技术

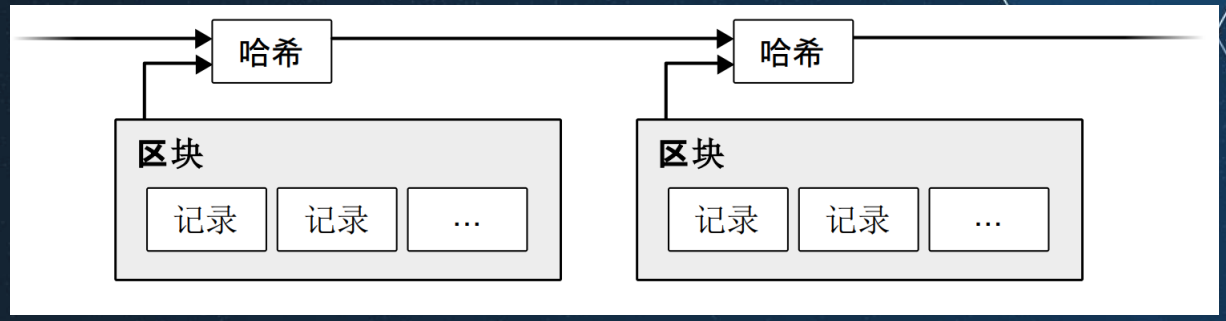
- 时间戳
- 哈希函数
- 默克尔树
- 非对称加密
- 数字签名

时间戳（Timestamp）

- 维基百科定义时间戳是指格林威治时间1970年01月01日00时00分00秒(北京时间1970年01月01日08时00分00秒)起至现在的总秒数。通俗的讲， 时间戳是一份能够表示一份数据在一个特定时间点已经存在的完整的可验证的数据。
- 时间戳的功能通常是为了记录某件事情的发生日期与时间，以及证明事实存在并保证先后关系。
- 物理时间戳、计算机上的时间轴、逻辑时间戳
- 基于文档时间戳的数字公证服务以证明各类电子文档的创建时间，由此保证数据的可追溯与不可篡改。时间戳服务器对新建文档、当前时间及指向之前文档签名的哈希指针进行签名，后续文档又对当前文档的签名再进行签名，如此形成了一个基于时间戳的证书链，该链反映了文件创建的先后顺序，且链中的时间戳极难改。

时间戳——比特币系统的时间戳设计

- 时间戳服务器：通过把以数据区块形式存在的一组比特币交易实施哈希运算并加盖时间戳，并在比特币网络中广播该哈希值。这个时间戳证明在该时间这个数据一定是存在的，因为只有数据只有在该时间才能得到相应的哈希值。
- 每个时间戳的哈希值包含了前一个时间戳，后续的时间戳都是对之前时间戳的增强，形成了一个环环相扣的时间戳链条。
- 对其中一个时间戳的篡改的代价极大，需要同时篡改其后生成的所有时间戳。



时间戳——比特币系统的时间戳设计

- 比特币系统设计了两个防止节点恶意修改本地时间的规则：

① 比特币节点会与其连接上的所有其他节点进行时间校正，且要求连接的节点数量至少为5个，然后选择这群节点的时间中位数作为时间戳，该中位数时间(称为网络调整时间)与本地系统时间的差别不超过70分钟，否则不会更改并会提醒节点更新本机的时间；

② 合法的时间戳必须大于前11个区块的中位数并且小于比特币节点的网络调整时间+2小时。换言之特币节点会拒绝接收时间戳不在此时间范围内的区块。

哈希函数

- 哈希函数可以在有限且合理的时间内，将任意长度的二进制字符串映射为固定长度的二进制字符串，其输出值称为哈希值(Hash Value)或者数字摘要(Digital Digest)。
- 哈希函数的数学表达形式如下:

$$h=H(m)$$

其中m表示任意长度的输入消息，H表示哈希函数的具体实现，h则表示固定长度的输出哈希值。

- 哈希碰撞：如果出现两个不同的输入m1和m2，使得H(m1)=H(m2)时称为发生一次哈希碰撞。理论上，哈希碰撞不可避免，实际应用中，加长输出字符串长度可以使得发生哈希碰撞的概率极低。

哈希函数的技术特征

- 抗原像性(Pre-Image Resistance)：也称单向性，即对任意给定的y来说，找到任意原像x使得H(x)=y在计算上是不可行的。即对任意预定义的输出数据，无法反推其输入数据。因此，哈希函数可以看作是一类只有加密过程而没有解密过程的“单向”加密函数。
- 抗第二原像性(Second Pre-Image Resistance)或称弱抗碰撞性：即给定输入数据x1时，寻找其他不等于x1的数据x2，使得H(x1)=H(x2)在计算上是不可行的。
- 强抗碰撞性(Collision Resistance)：寻找任意两个不同的输入x1和x2，使得H(x1)=H(x2)在计算上是不可行的。
- 迷题友好性(Puzzle Friendly)：对于任意n位输出y来说，假设k是从具有较高不可预测性的高阶最小熵分布中选取的，则无法找到有效方法可在比2的n次方小很多的时间内找到x，使得H(k|x)=y成立。
- 雪崩效应：输入数据发生任何细微变化，哪怕仅有一个二进制位不同，也会导致输出结果发生明显改变。
- 定长/定时性：不同长度输入数据的哈希过程消耗大约相同的时间且产生固定长度的输出。

哈希函数在区块链和数字加密货币体系中的应用模式

- 完整性校验：哈希函数的单向性和抗碰撞性通常可以用于校验消息的完整性，以防止消息在传输和存储的过程中出现未经授权的篡改。
- 数据要素管理：哈希函数的抗碰撞性使其可以作为任意数据的“数字指纹”，从而可以利用数据的哈希值来对其进行高效管理。例如，区块链系统的公钥、私钥、地址交易ID、区块ID等要素均是通过哈希算法生成并加以标识；区块链交易数据的重要组织方式——默克尔树、区块链系统的数字签名等主要操作也均是利用哈希函数来完成。
- 共识竞争：大多数区块链系统，特别是基于PoW共识的公有链系统，都是利用大量的哈希函数运算来确定共识过程中获胜的矿工。这主要是利用哈希函数的迷题友好性，使得矿工除了付出大量算力资源执行哈希运算之外，没有其他捷径可以对PoW共识过程进行求解。

哈希函数-MD系列算法

- MD是消息摘要(Message Digest)的缩写，MD系列算法是一类较为成熟的哈希算法，包括MD2、MD4和MD5算法，由MIT的罗纳尔多·李维斯特教授(Ronald L. Rivest，美国密码学家、图灵奖获得者、RSA算法的第一设计者“R”)分别于1989年、1990年和1992年设计提出，可将输入数据映射为128位的哈希值。
- 虽然这些算法的安全性逐渐提高，但均被证明是不够安全的。2004年，中国密码学家王小云教授在美国加州圣巴巴拉召开的国际密码学会议Crypto2004上宣布找出了MD5算法的碰撞实例，从而证明了MD5不具备“强抗碰撞性”。MD5算法被破解后，Ronald L. Rivest教授在国际会议Crypto2008上提出了更为完善的MD6算法，但并未得到广泛使用。

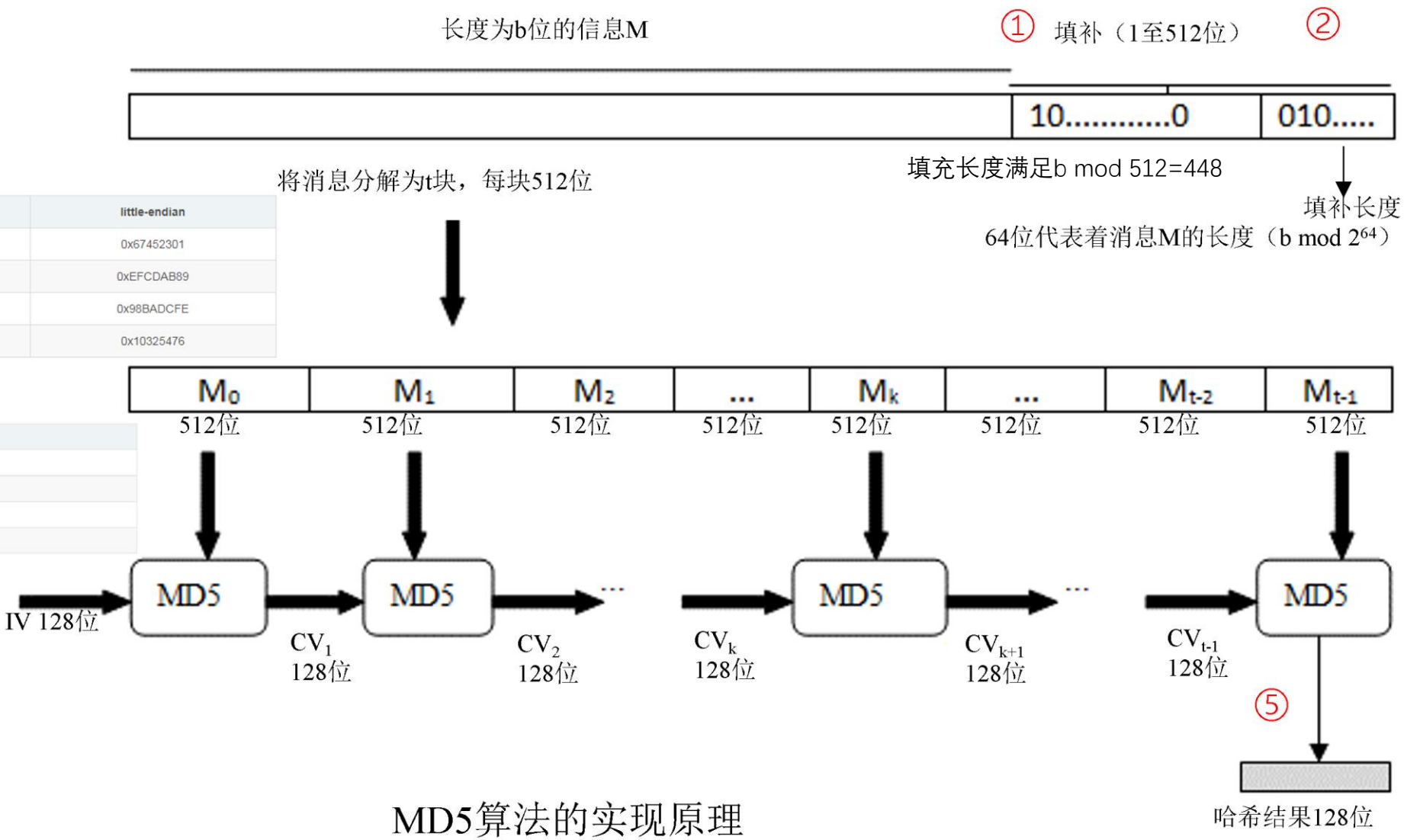
哈希函数-MD5算法

③ 初始化128位缓冲区（4个32位）

word					little-endian
A	01	23	45	67	0x67452301
B	89	AB	CD	EF	0xEFCDAB89
C	FE	DC	BA	98	0x98BADCFE
D	76	54	32	10	0x10325476

④ 迭代消息处理，4轮64个子步骤

Function	return
F(X,Y,Z)	$(X \wedge Y) \vee (\neg X \wedge Z)$
G(X,Y,Z)	$(X \wedge Z) \vee (Y \wedge \neg Z)$
H(X,Y,Z)	$X \oplus Y \oplus Z$
I(X,Y,Z)	$Y \oplus (X \vee \neg Z)$



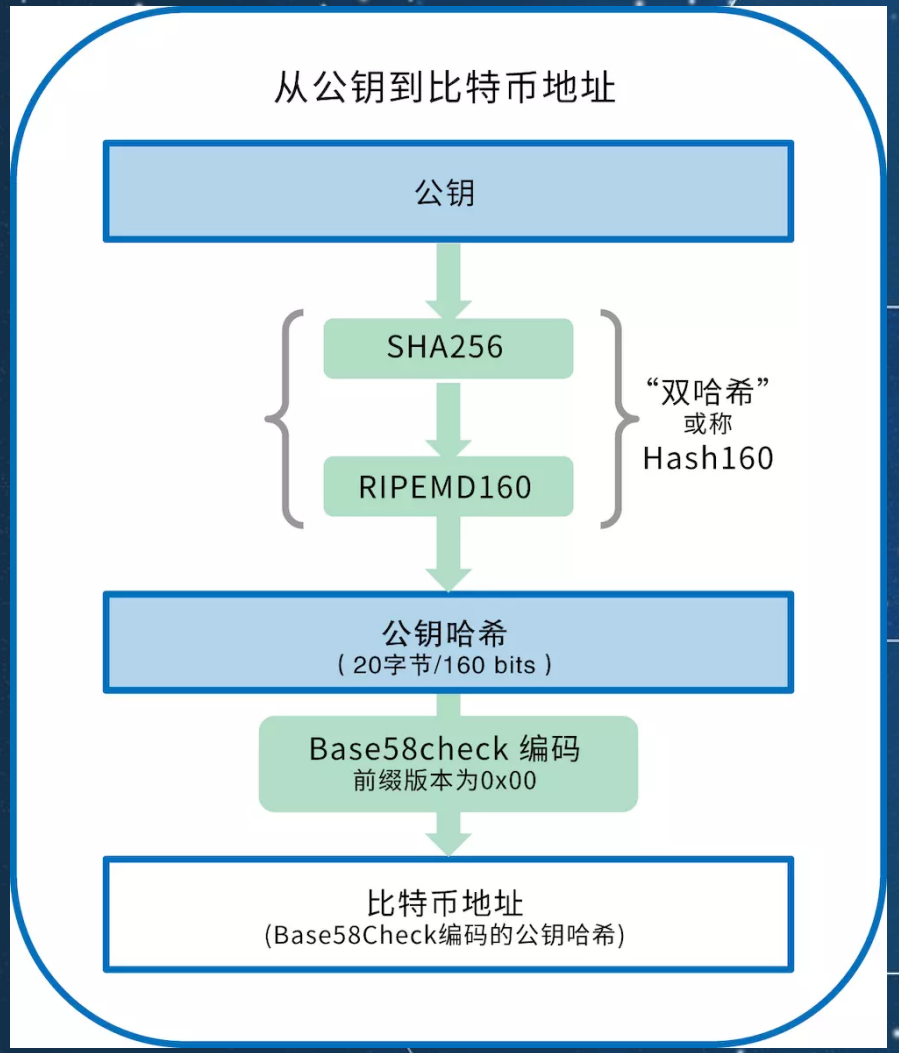
MD5算法的实现原理

哈希函数-SHA算法

- SHA是安全哈希算法(Secure Hash Algorithm)的缩写，是由美国国家安全局(National Security Agency,NSA)设计、美国国家标准与技术研究院(National Institute of Standard and Technology,NIST)发布的密码学哈希算法族，其家族成员包括SHA-1、SHA224、SHA-256、SHA384和SHA-512等。
- 1993年，NIST发布了SHA系列算法的首个实现SHA-0，但很快被撤回。
- 1995年提出SHA-1，设计原理与MD5算法相似，输出长度为160位哈希值。SHA-1算法已被证明不具备“强抗碰撞性”。2005年有中国密码学家王小云破解。
- 随后，为了提高安全性，NIST陆续发布了SHA-256、SHA-384、SHA-512以及SHA-224算法(统称为SHA-2)，按照输出哈希值的长度命名，至今尚未出现针对SHA-2的有效攻击。因而，比特币在设计之初即选择采用了当时公认最安全和最先进的SHA-256算法。

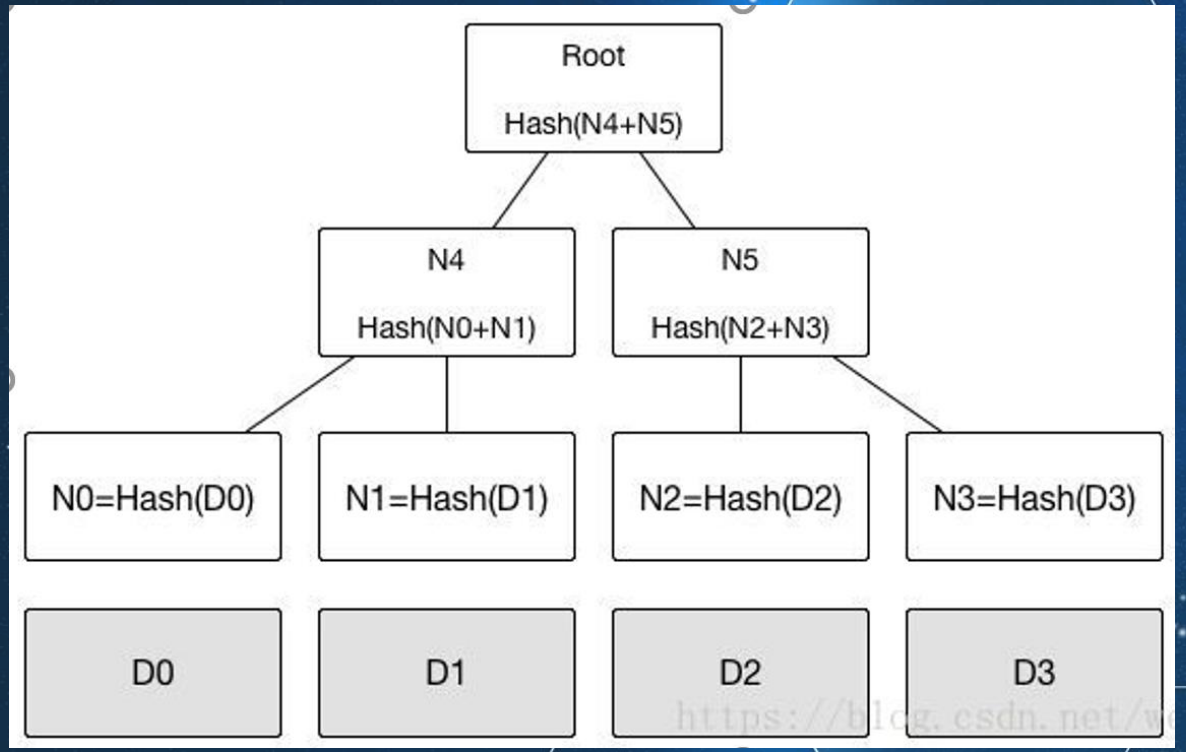
哈希函数-RIPEMD算法

- RIPEMD（RACE Integrity Primitives Evaluation Message Digest，RACE原始完整性校验消息摘要）是1996年由鲁汶大学的汉斯·多贝尔廷（Hans Dobbertin）、安东·波塞拉尔斯（Antoon Bosselaers）和巴特·普拉尼（Bart Prenee）三人组成的COSIC研究小组基于MD4和MD5算法提出来的。
- RIPEMD家族成员有RIPEMD-128、RIPEMD-160、RIPEMD-256和RIPEMD-320，对应输出长度分别为128位、160位、256位和320位，其中RIPEMD-160是最常见的版本。
- 比特币系统的公钥—地址转换过程就是利用SHA-256 + RIPEMD-160双哈希算法，将65字节的公钥转换为20字节的摘要结果，再经过SHA-256哈希算法和Base58转换过程，形成长度为33字节的比特币地址。



默克尔树MerKle Tree

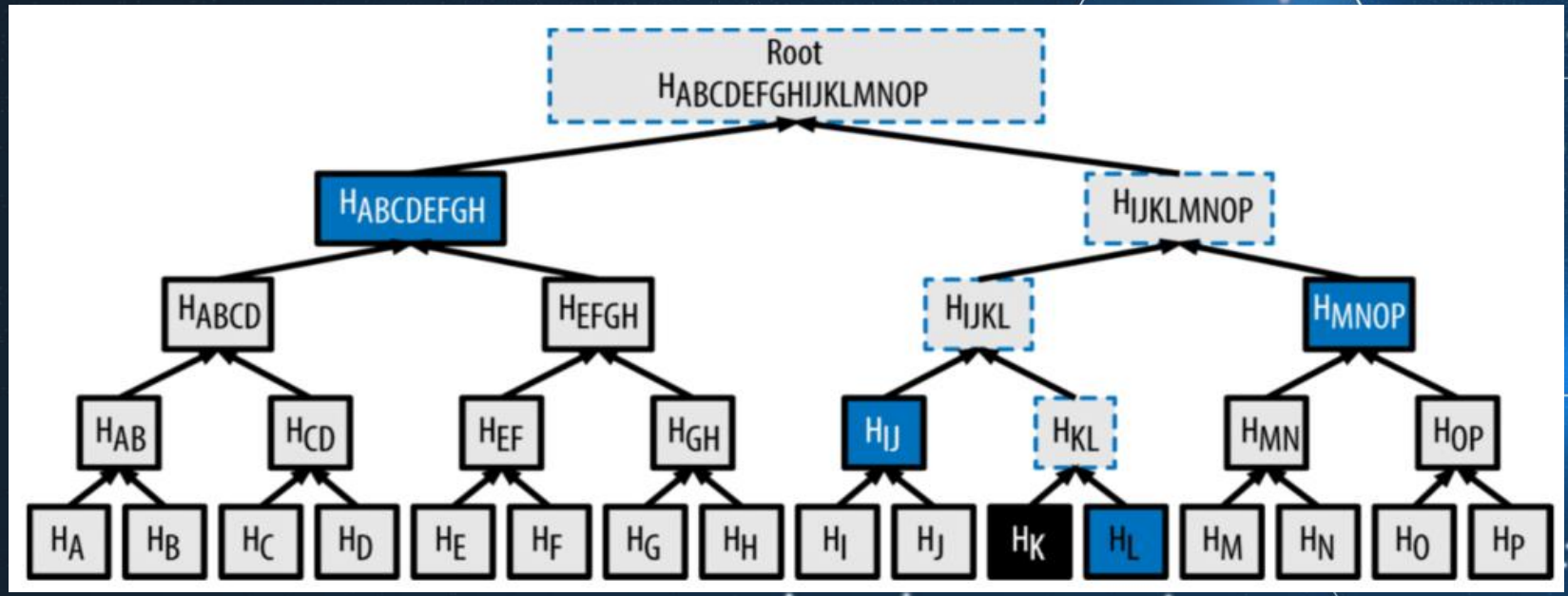
- 默克尔树指1979年由拉尔夫·默克尔发明的树结构哈希链，也被称为哈希树。
- 区块链系统最重要的底层数据结构之一，其作用是快速归纳和校验区块数据的存在性和完整性。
- 优点一，区块头仅需包含根哈希值。
- 优点二，支持“简化支付验证”，即在不运行完整区块链网络节点的情况下也能对交易数据进行校验。（轻节点）



数据块

默克尔树在比特币系统的应用

- Hash函数采用SHA256算法
- 二叉hash树（交易数量为奇数时，最后一个交易重复）
- 默克尔路径，可快速验证某个区块是否存在指定交易



验证交易H_k

默克尔树-简化支付验证

简化支付验证（Simplified Payment Verification，SPV）是基于区块链和默克尔树结构的特点而设计的一种即使没有完整交易记录，也能够方便、安全和快速的验证支付的方法。

SPV技术可以使得区块链节点在不需下载和存储完整区块链数据的情况下方便地验证支付。

SPV节点验证支付的基本思路是首先根据待验证交易信息向区块链网络发起查询请求（称为默克尔区块消息，Merkle Block Message）；其他有完整区块链数据的节点收到该请求后，利用待验证交易信息在其本地区块链数据库中查询，并将获得的验证路径返回给SPV节点；SPV节点利用该验证路径再做一次校验，如果确认无误，即可认为该交易是可信的。

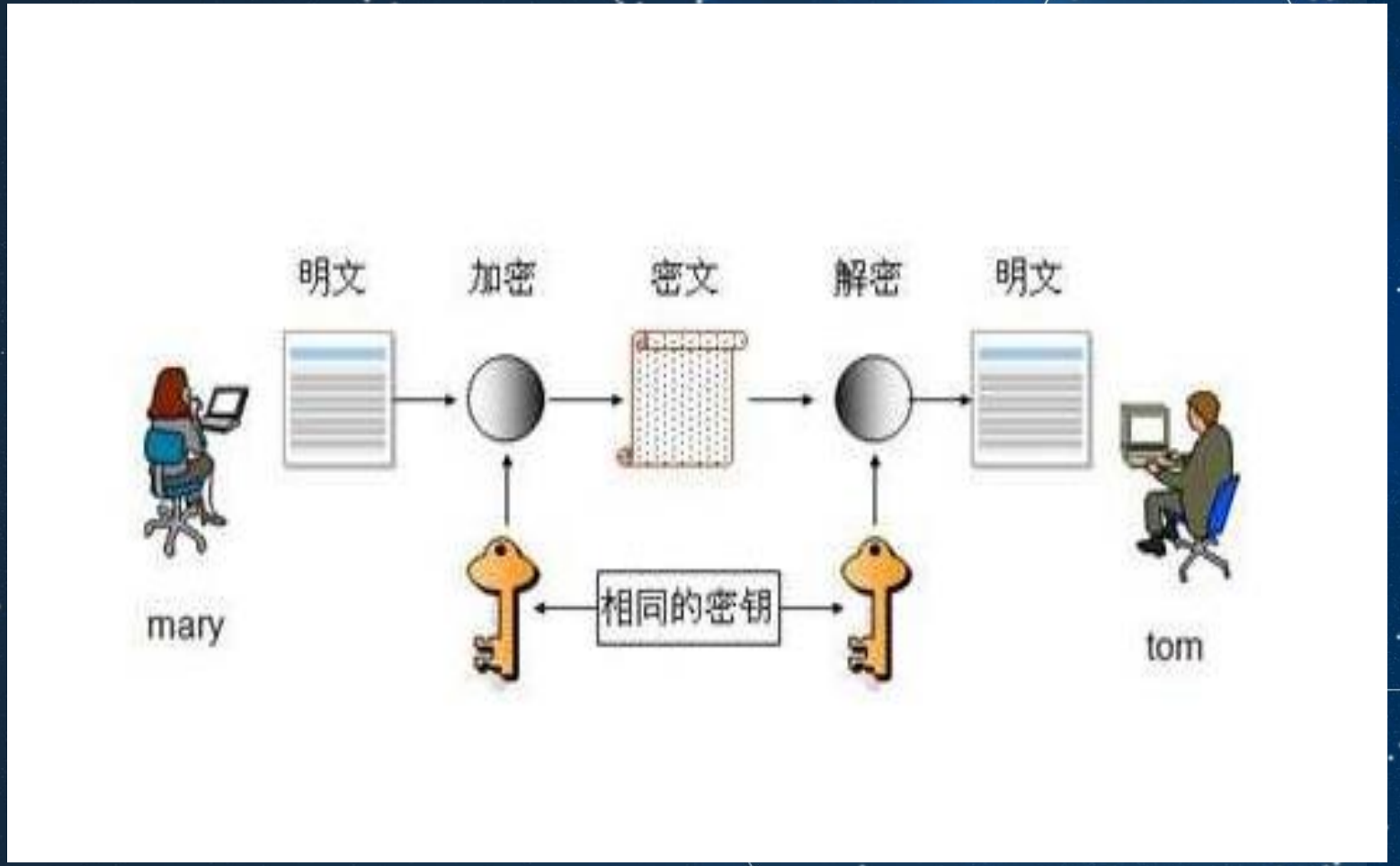
默克尔树-简化支付验证

简易支付验证SPV节点验证支付的具体步骤如下：

1) 步骤1：SPV节点获得待验证交易信息，向区块链网络发起Merkle Block Message查询请求；
2) 步骤2：其他有完整区块链数据的节点收到请求之后，执行如下步骤：
   - 定位包含该交易的区块
   - 检查该区块是否属于整个网络中的最长链
   - 取出所有交易生成默克尔树，利用getProof方法获得待验证交易的验证路径
   - 将验证路径发送回请求源SPV节点
3) 步骤3：SPV节点获得验证路径后，执行如下操作：
   - 同步区块链，确保是整个网络中最长的一条
   - 先拿默克尔根去区块链中查找，确保该默克尔根哈希是在链条中
   - 利用获得的验证路径，再进行一次默克尔哈希校验，确保验证路径全部合法，则交易真实存在
   - 根据该交易所在区块头的位置，确定该交易已经得到多少个确认

非对称加密

加密算法一般分为**对称加密**和**非对称加密**。对称加密算法也称为**单密钥加密**，同一个密钥同时作为信息的加密和解密，如果在分布式网络中，使用对称加密，如何将对应的密钥发送给需要解密信息的人是一个很难解决的问题。



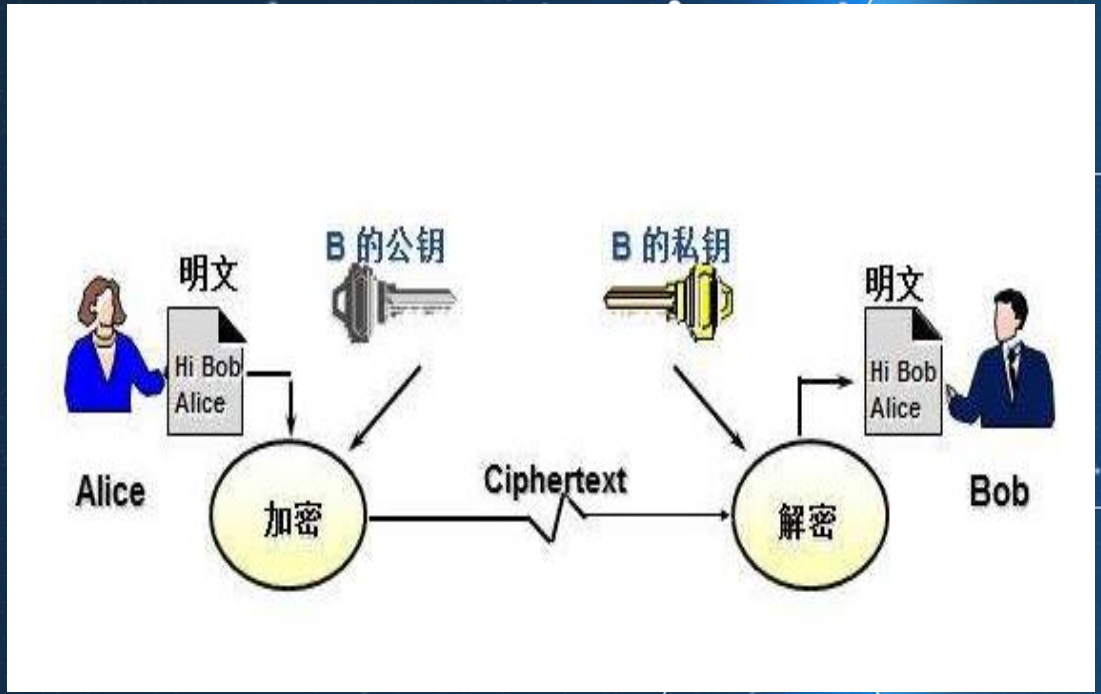
非对称加密

非对称加密中含有一个**密钥对**，**公钥和私钥**。私钥由一方安全保管，不轻易外泄，而公钥可以发送给任何人。非对称加密算法中使用密钥对中的一个进行加密，只能使用另一个进行解密。

目前最常用的非对称加密算法是**RSA算法**。

它的原理是：两个超大素数相乘得到的结果几乎无法因式分解逆运算得到原本的素数，从而实现可加密而不易破解。

比特币区块链网络则采用了**椭圆加密算法**。



非对称加密在比特币系统中的应用

- 私钥证明了用户对于账户的所有权，如果用户想要使用某个账户中的比特币，只有拥有该账户对应的私钥，才能如愿使用。在登录认证的场景中，用户输入私钥信息，客户端使用私钥加密登录信息后发送给服务器，服务器接收后采用对应的公钥解密认证登录信息。
- 在比特币交易中用户使用私钥对交易进行签名，交易信息广播后，验证节点通过公钥对信息进行解密，从而确保信息是由A发送的
- 区块链网络充分利用了非对称加密的特性，一是使用其中一个密钥加密信息后，只有对应另外一个密钥才能解开；二是公钥可以向其他人公开，公钥不能逆推出私钥，保证了账户安全性。

数字签名

- 数字签名(Digital Signature)是一种证明数字消息、文档或者资产的真实性的数学方案，其作用：
  - 身份认证，使接收者有理由相信其接收到的内容是由已知的发送者发出的
  - 不可抵赖，发送者无法否认其曾经发送过
  - 完整性，该内容在传输过程中未被算改(完整性)
- 数字签名定义（ISO7498-2标准）：是附加在数据单元上的一些数据，或是对数据单元所做的密码变换，这种数据和变换允许数据单元的接收者用以确认数据单元来源和数据单元的完整性，并保护数据，防止被人进行伪造。
- 1976年, Whitfield Diffie和Martin Hellman首次描述了数字签名方案的概念。（基于RSA算法等）

数字签名的模型

数字签名方案的模型通常可以表示为七元组(M,S,SK,PK,Gen,Sign,Verify)：

① M：某字母表中串的集合组成的明文消息空间，即待发送消息内容集合；

② S：可能的签名空间；

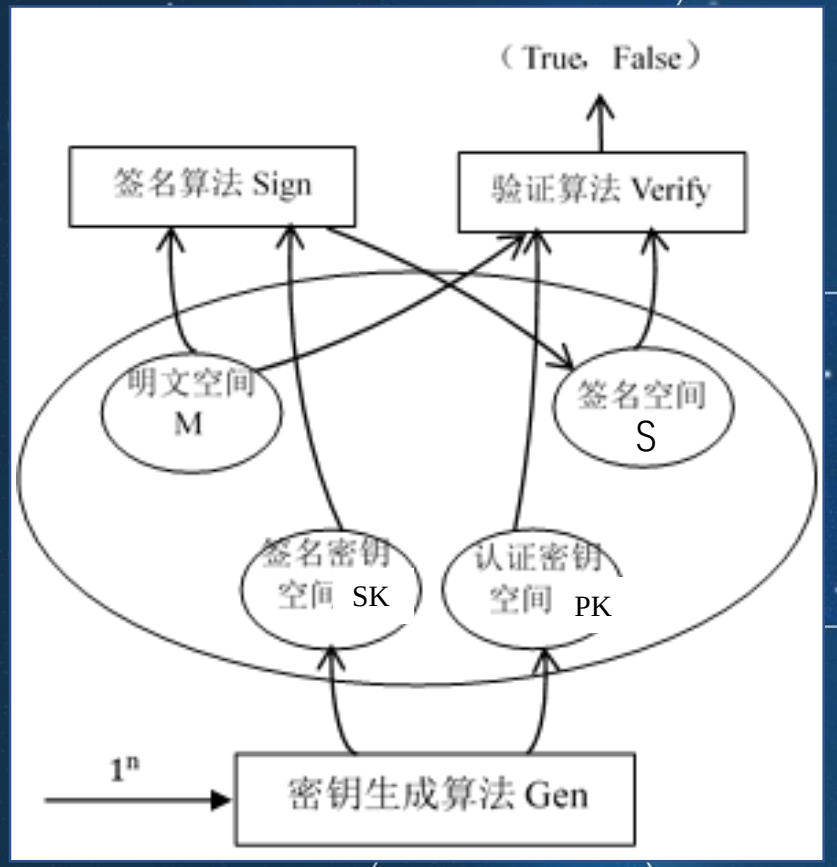
③ SK：签名密钥空间，即用于生成签名的私钥集合；

④ PK：验证密钥空间，即用于验证签名的公钥集合；

⑤ Gen:N→SK×PK：密钥生成算法，可生成一对匹配的公钥pk和私钥sk；

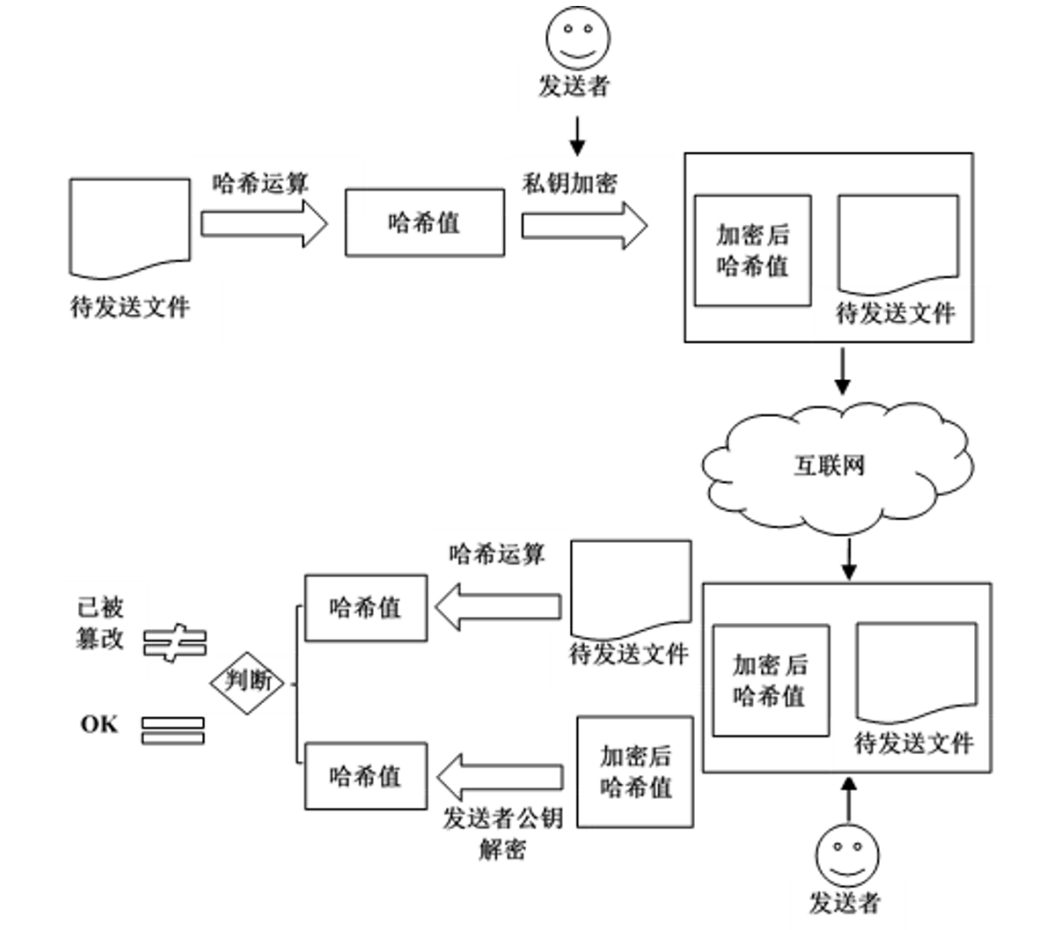
⑥ Sign:M×SK→S：签名算法，利用私钥sk生成消息m的签名s；

⑦ Verify:M×S×PK→{True,False}：验证算法，利用公钥pk验证消息的签名s是否正确。



数字签名的模型要素

数字签名的流程



签名阶段Sign：

① 发送者运用哈希函数等数字摘要技术，获得待发送消息m的哈希值H(m)；

② 发送者使用私钥sk对H(m)加密，生成数字签名Sign[H(m),sk]→s;

③ 发送者将数字签名s和待发送消息m一起发送给接收者。

验证阶段Verify:

① 接收者使用发送者的公钥pk对签名s解密；还原出哈希值H1(m)；

② 同时采用相同的哈希算法重新生成接收消息m的哈希值H2(m)；

③ 判断H1(m)和H2(m)是否相等；如果相等则证明信息未被篡改，否则证明信息在传输过程中已被篡改。

多重签名

- 多重签名(Multi Signature)是数字签名技术的重要应用模式，常用于多个参与者对某个消息、文件和资产同时拥有签名权或者支付权的场景。如银行的联名账户。
- 多重签名场景通常需要N个参与者之中至少有M个参与者联合签名，其中N≥M≥1。当N=M=1时,多重签名退化为传统的单人签名。
- 根据签名过程的不同,多重签名方案可以分为两类：有序多重签名和广播多重签名。前者，签名者之间的签名次序是一种串行的顺序，而后者，签名者之间的签名次序是一种并行的顺序。
- 多重签名被认为是区块链和比特币发展历史上的重要里程碑之一,不仅可以极大地提升比特币的安全性,同时也衍生出许多新型的商业模式。比特币系统一般采用“N选M”的形式，即该多重签名地址共有N个私钥，至少需要其中M个私钥共同签名才能从这个地址中转账。

谢谢!