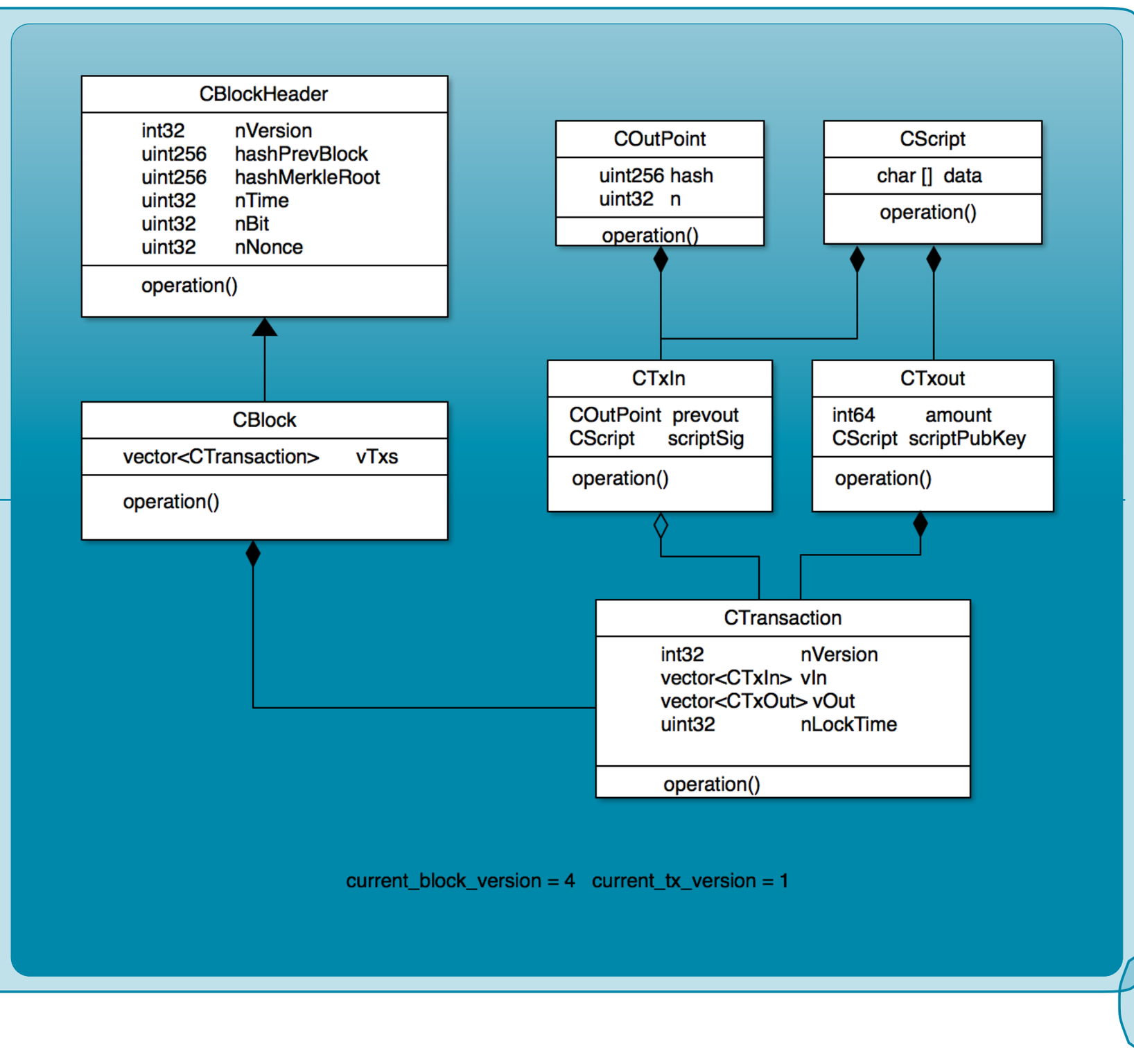
Bitcoin@Github (7K+)

- C/C++(200+)
  - Bitcoin-core
    - 要点
      - 协议、数据结构
        - Block
        - Transaction
        - MessageProtocol
        - Script
      - 基本算法
        - Hash256(SHA-256)/PIMEMD-160
        - ECDSA
        - Variable Integer
        - MerkelTree
        - BloomFilter
      - BIP
        - Bitcoin Improvement Proposal
        - Bip1/Bip32/Bip101
- Java (500+)
  - BitcoinJ
  - Android(90+)
    - Bitherj/Bitther-Android
- iOS(50~)
  - Bitheri/Bither-ios
- JS(1k+)
  - BitcoinJS
- Python(1k+)
- ....

Block数据结构



1 Coin
- POW(10, 8) Satoshis
- MaxMoney = 2100*POW(10,5)*Coin

数据传输协议

ver：70002

命令字符串	含义
addr	网络节点地址
alert	警告
block	区块
filteradd	添加过滤交易信息
filterclear	清理过滤器
filterload	加载过滤器
getaddr	获取地址
getblocks	获取区块
getdata	获取数据
getheaders	获取区块头
headers	区块头
inv	库存清单
mempool	内存池
merkleblock	merkle块
notfound	没有获取相匹配的数据
ping	判断网络是否连通
pong	ping回应
reject	拒绝
tx	交易信息
verack	版本信息回应
version	获取版本信息

```
/**
 * Compact Size
 * size <  253        -- 1 byte
 * size <= USHRT_MAX  -- 3 bytes  (253 + 2 bytes)
 * size <= UINT_MAX   -- 5 bytes  (254 + 4 bytes)
 * size >  UINT_MAX   -- 9 bytes  (255 + 8 bytes)
 */
```

CompactSize(VarInteger 变长整数)

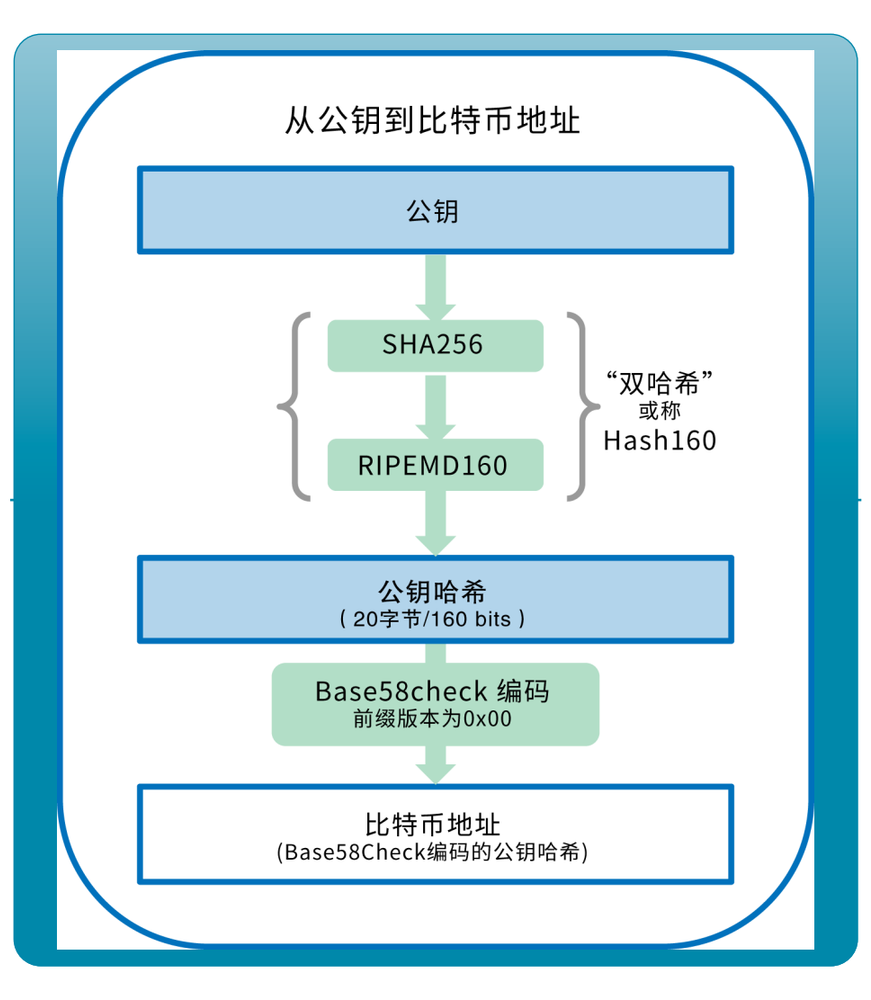
- <0xFD :uint8
- 0xFD:uint16
- 0XFE: uint32
- 0xFF: uint64

7个有效bit位，第8个bit用于表示是否有后续

```
a base-128 encoding of the number
 * * Very small (0-127: 1 byte, 128-16511: 2 bytes, 16512-2113663: 3 bytes ...)
```

CVarInt（Google protobuf 变长编码）

- Protobuf vs Json
- ZigZag编码



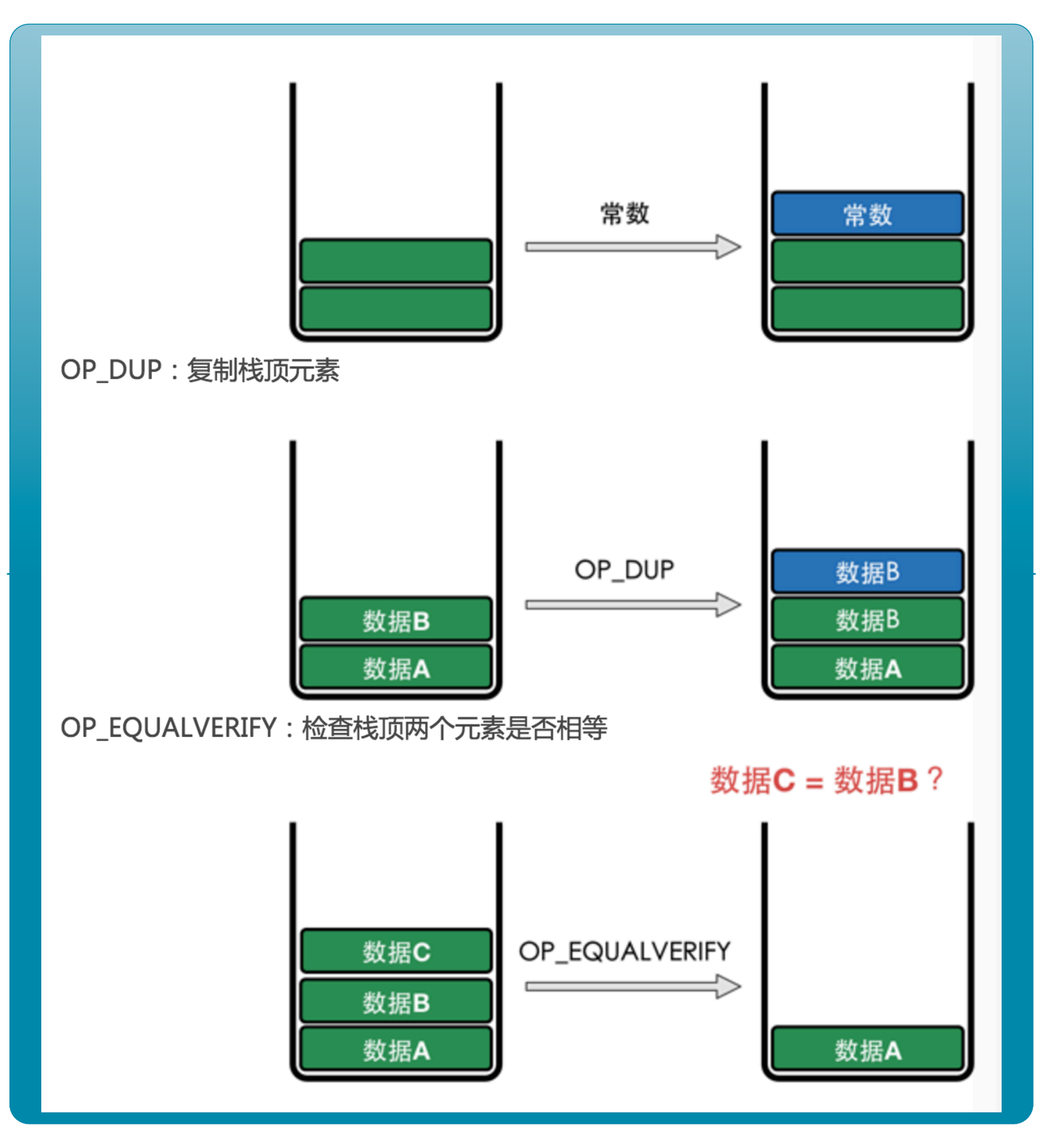
私钥、公钥、地址相关

- PrivateKey
  - ECDSA
    - Secp256k1
  - PublicKey
    - Address
  - HASH256-》32bytes
  - Ripemd160-》20Bytes

```
地址计算过程:
1) pubHash160 = (Ripemd160(Sha256(pubkey))
2) addr0 = versoin + pubHash160
2) verify = Sub4(Sha256(Sha256(addr0)))
3) addr = Base58(addr0 + verify)
```

Script（脚本）

- 脚本指令（0xFF）
- 堆栈式、从左像右
- 智能合约 ...



MerkleTree

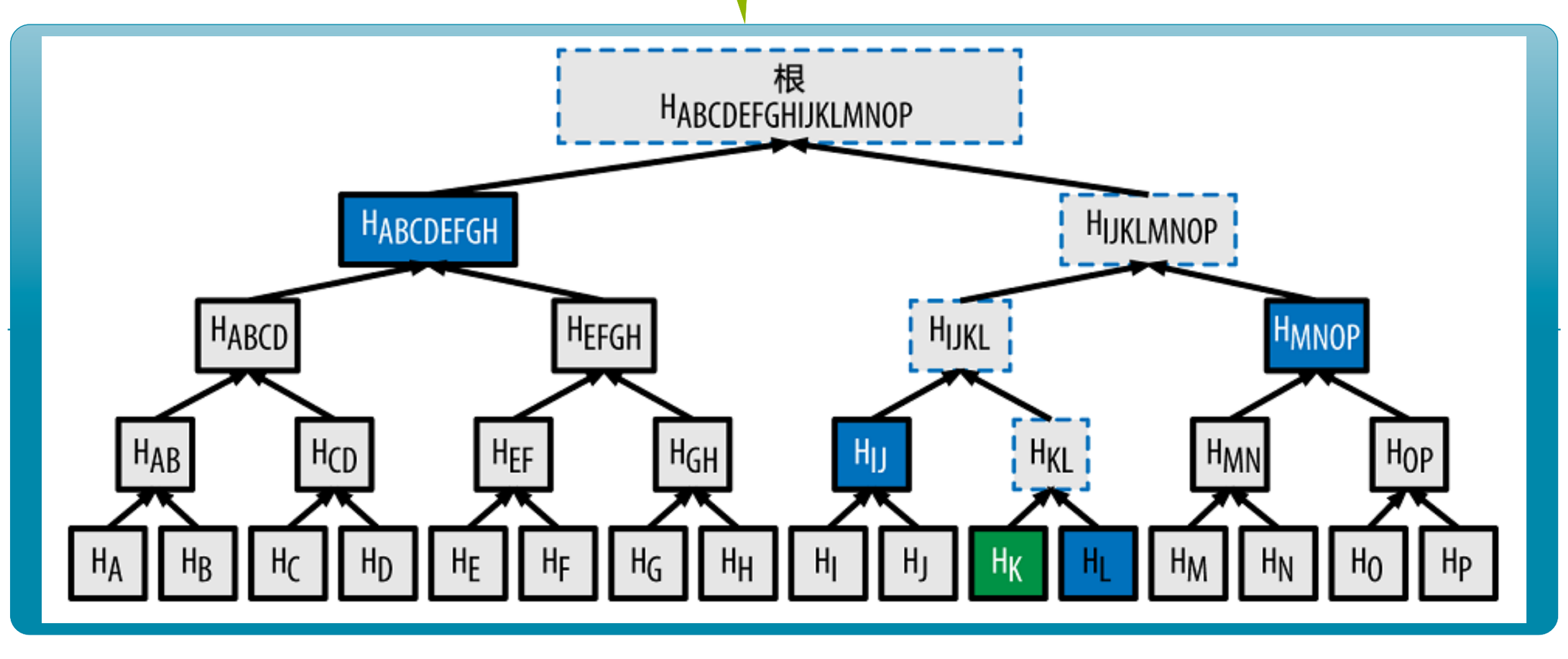
- 数据校验
- SPV
- 100%准备金

Merkle树的效率

Merkle区块消息包含区块头和一条连接目标交易与Merkle根的Merkle路径。SPV节点能够使用该路径找到与该交易相关的区块，进而验证对应区块中该交易的有无。SPV节点同时也使用区块头去关联区块和区块链中的区域区块。这两种关联，交易与区块、区块和区块链，证明交易存在于区块链。简而言之，SPV节点会收到少于1KB的有关区块头和Merkle路径的数据，其数据量比一个完整的区块（目前大约有1MB）少了一千倍有余。

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

依表可得，当区块大小由16笔交易（4KB）急剧增加至65,535笔交易（16MB）时，为证明交易存在的Merkle路径长度增长极其缓慢，仅仅从128字节到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小的Merkle路径就能认证一笔交易的存在，而不需要存储或者传输大量区块链中大多数内容，这些内容可能有几个G的大小。这种不需要维护一条完整的区块链的节点，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。

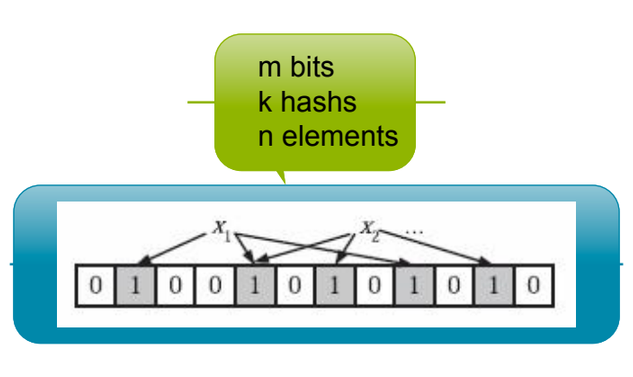


BloomFilter

- Bitcoin-core SPV
- 地址追踪

布隆过滤器 (Bloom Filter)是由Burton Howard Bloom于1970年提出，它是一种space efficient的概率型数据结构，用于判断一个元素是否在集合中。在垃圾邮件过滤的黑白名单方法、爬虫(Crawler)的网址判重模块中等等经常被用到。哈希表也能用于判断元素是否在集合中，但是布隆过滤器只需要哈希表的1/8或1/4的空间复杂度就能完成同样的问题。布隆过滤器可以插入元素，但不可以删除已有元素。其中的元素越多，false positive rate(误报率)越大，但是false negative (漏报)是不可能的。

m bits
k hashs
n elements




```
//! 20,000 items with fp rate < 0.1%
or 10,000 items and <0.0001%

static const unsigned int
MAX_BLOOM_FILTER_SIZE = 36000; //
bytes
static const unsigned int
MAX_HASH_FUNCS = 50;

36000/1024 = 3.5k
```

三、举例说明

以垃圾邮件过滤中黑白名单为例：现有1亿个email的黑名单，每个都拥有8 bytes的指纹信息，则可能的
$2^{64} = 8 * 10^{18}\ bits = 10^{9}\ GB$ ，对于bit array来说是根本不可能的范围，而且元素的数量（即emai
，相比于元素范围过于稀疏，而且还没有考虑到哈希表中的collision问题。

若采用哈希表，由于大多数采用open addressing来解决collision，而此时的search时间复杂度为：

$$\frac{1}{1-\frac{n}{m}}$$

即若哈希表半满(n/m = 1/2)，则每次search需要probe 2次，因此在保证效率的情况下哈希表的存储
过50%。此时每个元素占8 bytes，总空间为：

$$\frac{10^{8} * 8\ bytes}{50\%} = 1.6\ GB$$

若采用Perfect hashing（这里可以采用Perfect hashing是因为主要操作是search/query，而并不是
remove），虽然保证worst-case也只有一次probe，但是空间利用率更低，一般情况下为50%，wor
到一半的概率为25%。

若采用布隆过滤器，取k=8。因为n为1亿，所以总共需要 $8 * 10^{8}\ bits$ 被置位为1，又因为在保证误判
取合适时，空间利用率为50%（后面会解释），所以总空间为：

$$m = \frac{10^{8} * 8\ bits}{50\%} = 1.6 * 10^{9} bits = 200MB$$

所需空间比上述哈希结构小得多，并且误判率在万分之一以下。