BlockChain内核源代码情景分析

v0.3 (2016/1)

by 南高所 杰伟成小组

All about Coding

区块链的业务形态

整体代码结构

代码解读

Q&A

1 区块链核心概念摘要

1个开放平台 + 4种区块链模式

1

开放平台

Ethereum

4

公共链

BitCoin

私有链

R3 CEV

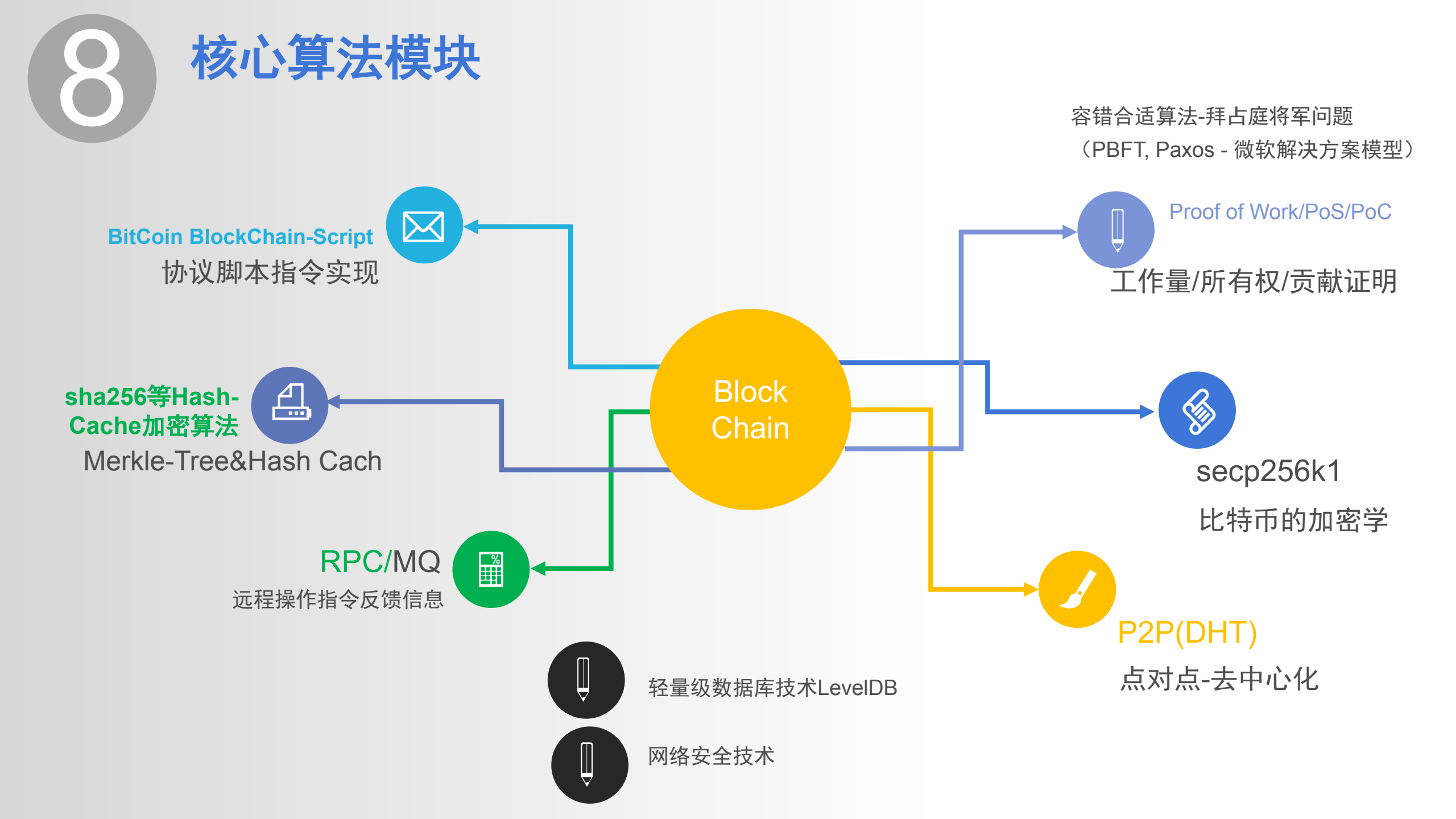
侧链

Blockstream

联盟链

Hydrachain

8 核心算法模块


容错合适算法-拜占庭将军问题
（PBFT, Paxos - 微软解决方案模型）
BitCoin BlockChain-Script
协议脚本指令实现
Proof of Work/PoS/PoC
工作量/所有权/贡献证明
sha256等Hash-Cache加密算法
Merkle-Tree&Hash Cach
Block Chain
secp256k1
比特币的加密学
RPC/MQ
远程操作指令反馈信息
P2P(DHT)
点对点-去中心化
轻量级数据库技术LevelDB
网络安全技术

2 区块链的4种形态

BlockChain的4种形态

- Public Chain(公用链)：

  最为人所知

  人人可参与，PoW(工作量证明)，PoS，全球性质，已有相对较多的应用...

  交易及运行效率问题（区块同步问题），安全性。

- Private Chain(私有链)：个人或公司内部控制权限，效率，安全等问题会较易解决。

  银行等机构大举进入研究，部署。R3等前景看好。

- Consortium Chain(联盟链)：由几个人，机构，政府控制。“多中心化”。PoS, 拜占庭容错共识机制保证效率。

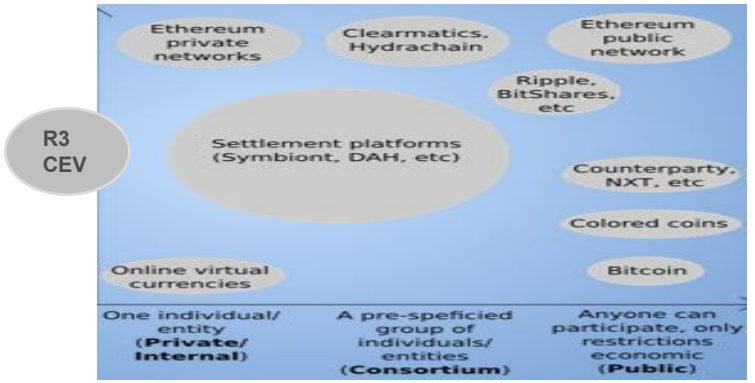
- Side Chain（侧链）:它通过一种双向铆钉的方式，允许资产从比特币主区块链移动到另一条区块链。这使用户能用他们已有的资产来使用新的和创新的加密货币系统。

  侧链可使得比特币成为其它竞争链的原生货币。

  侧链是一个具有高度灵活性的一般性扩展。

Vitalik Buterin（以太坊创始人）谈Private vs Public (1)

Vitalik Buterin（以太坊创始人）



1.公共区块链（Public blockchains）

公共区块链是指全世界任何人都可读取的、任何人都能发送交易且交易能获得有效确认的、任何人都能参与其中共识过程的区块链——共识过程决定哪个区块可被添加到区块链中和明确当前状态。作为中心化或者准中心化信任的替代物，公共区块链的安全由“加密数字经济”维护——“加密数字经济”采取工作量证明机制或权益证明机制等方式，将经济奖励和加密数字验证结合了起来，并遵循着一般原则：每个人从中可获得的经济奖励，与对共识过程作出的贡献成正比。这些区块链通常被认为是“完全去中心化”的。

2.联盟区块链：（Consortium blockchains）

联盟区块链是指其共识过程受到预选节点控制的区块链；例如，不妨想象一个有15个金融机构组成的共同体，每个机构都运行着一个节点，而且为了使每个区块生效需要获得其中10个机构的确认（2/3确认）。区块链或许允许每个人都可读取，或者只受限于参与者，或走混合型路线，例如区块的根哈希及其API（应用程序接口）对外公开，API可允许外界用来作有限次数的查询和获取区块链状态的信息。这些区块链可视为“部分去中心化”。

3.完全私有区块链（Fully private blockchains）

完全私有的区块链是指其写入权限仅在一个组织手里的区块链。读取权限或者对外开放，或者被任意程度地进行了限制。相关的应用囊括数据库管理、审计、甚至一个公司，尽管在有些情况下希望它能有公共的可审计性，但在很多的情形下，公共的可读性并非是必须的。

Vitalik Buterin（以太坊创始人）谈Private vs Public(2)

1. 规则的改变

如果需要的话，运行着私有区块链的共同体或公司可以很容易地修改该区块链的规则，还原交易，修改余额等。在一些情况下，例如全国土地登记，这个功能是必要的; 但绝对不会存在着这样的系统，可以让“恐怖海盗罗伯茨”在一块清晰可见的土地上拥有合法所有权，所以试图建立一个不受政府控制的土地登记机构，在实践中是不会被政府本身承认的。当然，有人会说争辩说，可以在公共区块链上给政府留一个后门钥匙；当然有人会反驳说这种做法是小题大做，私有区块链的效率更高。关于私有区块链，我在后面会作介绍。

2. 验证者是公开的

因为验证是公开的，所以并不存在，来自中国的一些矿工出于共谋原因而致的51%攻击风险。

3.交易成本更便宜

交易只需被几个受信的高算力节点验证就可以了，而不是需要数万台笔记本的确认，因此交易成本会便宜。当下公共区块链的每个交易的费用超过0.01美元，这是个非常值得重视的问题，但也要注意，长远来看，随着可扩展的比特币技术的进步，它会有所改变，该技术有望将公共区块链的费用降低一到两个数量级，大致与高效的私有区块链系统差不多。

4. 节点可以很好地连接

节点互相可以很好地连接，故障可以迅速通过人工干预来修复，并允许使用共识算法减少区块时间，从而更快完成交易。公共区块链技术的进步，例如以太坊1.0概念和后来的权益证明机制，可让公共区块链达到“即时交易”的目标。但私有区块链仍是会更快，所以造成的延迟误差永远不会消失，正如光速并不遵循摩尔定律那样每两年翻一番。

5.隐私

如果读取权限受到限制，这样私有区块链还可提供更好的隐私保护。

考虑到上述情况，私有区块链似乎看起来更适合为机构所用。然而，对于机构，公共区块链仍存在着很多价值，这就是公共区块链一直提倡的自由、中立和开放。

Vitalik Buterin（以太坊创始人）谈Private vs Public(3)

公共区块链的优点可以归结为两大类：

1. 保护用户，免受开发者的影响

在公共区块链中程序开发者无权干涉用户，所以区块链可以保护使用他们开发的程序的用户。从天真的角度来看，的确难以理解为何程序开发者会愿意放弃自己的权限。然而，较为超前的经济分析为此提供了两个理由：借用Thomas Schelling的话语，妥协是一种力量。第一，如果你明确地选择做一些很难或者不可能的事情，其他人会更容易信任你并与您产生互动，因为他们自信那些事情不大可能发生在他们身上。第二，如果你是受人或其他外界因素的强迫，无法去做自己想做的事，你大可说句“即使我想，但我也没有权力去做”的话语作为谈判筹码，这样可以劝阻对方不去强迫你去做不情愿的事。程序开发者们所面临的主要的压力或者说风险，主要是来自政府，所以说“审查阻力”，便是公共区块链最大的优势。

2.网络效应

公共区块链是开放的，因此有可能被许多外界用户使用和产生一定的网络效应。举一个特定的例子，就拿域名托管来说吧。现在，如果A想卖给B一个域名，就有个需要待解决的风险问题：如果A首先出售了域名，但B可能还没给钱；或者如果B给钱了，但A还没出售域名。为解决这个问题，我们要设立中心化的托管中介，但须支付三到六个百分点的手续费。然而，如果我们在区块链上拥有一个域名系统，并使用这个区块链的货币，那么我们可建立交易费低至0的智能合约：A向该系统出售域名，系统马上将域名出售给首先支付资金的人，而且因为这系统是建立在公共区块链上所以值得信任。但注意为了使交易过程更高效，要将来自完全不同行业的完全不同的资产寄放在同一公共数据库上——这在私有区块链上是不可能轻易做到的。同样的例子可以是土地登记和产权保险，但注意若想可交互操作，要使用能被公共区块链验证的私有区块链，这样可通过跨链完成交易。

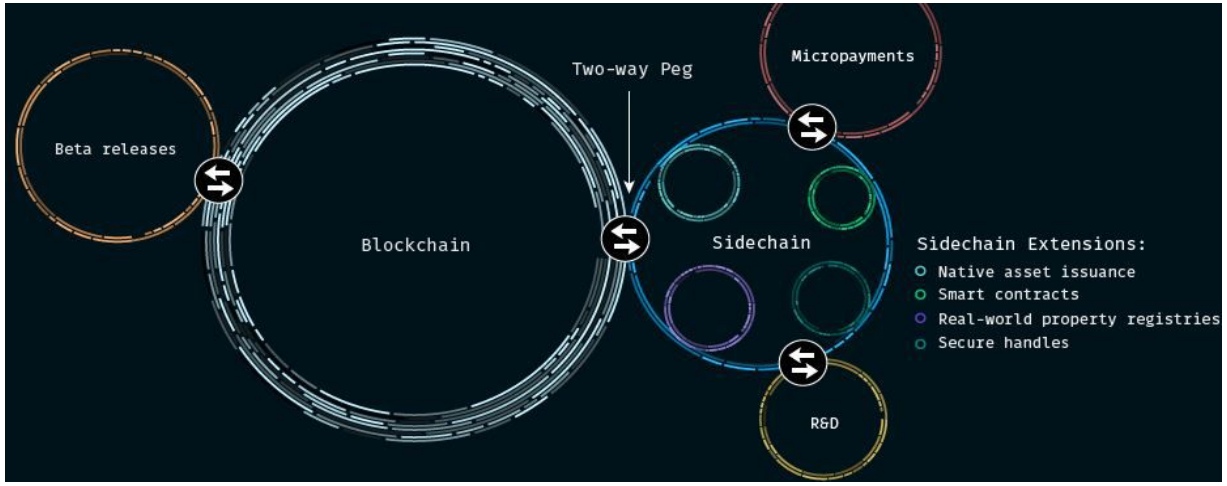
Side Chain(侧链介绍)- Blockstream谈侧链(1)

·我们有了比特币以及它的区块链，我们有了不少其它的区块链支撑着各种山寨币。而比特币的区块链本身，由于涉及了太多的能量，已变得低迷、笨拙，很难改变，阻碍了创新的脚步，成为了自身成功的牺牲品。

·直到遇到Austin Hill和Adam Back才发生改变。Hill曾是Zero-Knowledge Systems的创始人兼CEO，这家价值数百万美元公司曾领先时代15到20年；Back则是比特币背后的Hashcash算法的发明者。他们具有相当高的可信度，他们正在悄悄打造一家部分基于“侧链”（sidechains）概念的“区块链2.0”创业公司-blockstream

·侧链是由比特币通过比特币合约支持的新型区块链，就像由金条支持的美元英镑一样。原则上，你可以有成千上万个“挂”到比特币上的侧链，特性和目的各不相同……所有这些侧链依赖于比特币主区块链保障的弹性和稀缺性，一旦经过尝试和测试，反过来又能重复验证侧链的功能。

·如果侧链能够成功，尽管需要比特币核心协议做一些修改。可能会对现在的山寨币造成极大的打击。 因此这项提议早到广泛的质疑就不足为奇了，以太坊的首席科学家Vitalik Buterin称侧链不仅需要修改协议，还需要“所有比特币矿池运营商当中50%以上的同意以及积极配合。”



Blockstream

Side Chain(侧链介绍)- Blockstream谈侧链(2)

Blockstream

很明显每10分钟的1MB交易限制太小了，与我们期望的比特币交易处理能力相差太大。每秒1万次的交易处理将需要1GB大小的区块。Blockstream首席执行官奥斯汀.希尔(Austin Hill) 称，使用Liquid侧链的交易所，其资金移动所需的时间将从60分钟缩短至秒。

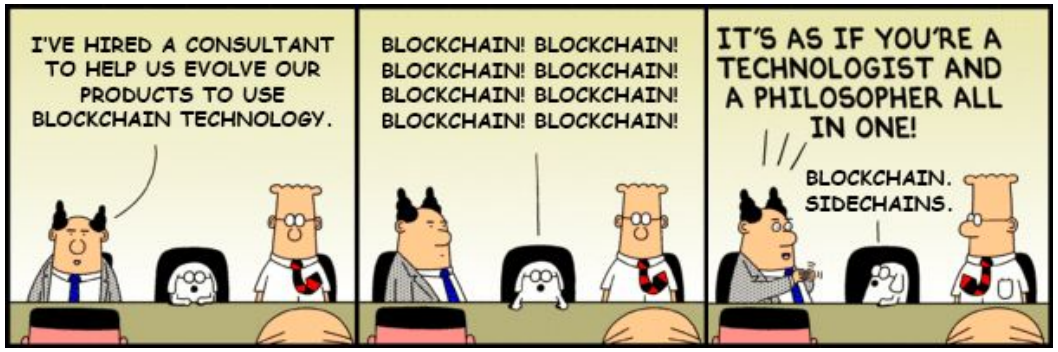
侧链可以起到缓解的作用。多个侧链共存是可能的，侧链可以具有更大的区块大小和更短的出块时间，比特币区块链只需要处理这些不同高速支付网络之间的比特币转移。这使得比特币可以按照人们的需求扩展，不需要剧烈地增加区块大小。

侧链是一个开放的理念，任何人都能够（和应该）使用，开发他们想要的任何侧链。侧链是一个仍在开发中的概念，也是引起比特币开发社区某些成员高度争议的一个概念，但如果它可以有效运行-这个方案可以有效地将比特币纳为互联网的储备货币。

侧链可以有许多不同种类的资产，只要侧链的创始人乐于这样做，这些资产可以发行给用户。侧链的限制是：外部资产，例如比特币，从侧链转回比特币区块链时，转回去的数量需等同于先前转进的数量–所以如果有5个比特币转到侧链中，只有5个比特币可以转回。

侧链不需要原始发行‘货币’。你可以从另一个链中转移资产到自己的链中。例如，持有比特币的人可以将它们转移到钉住比特币的侧链中。

Blockstream的一个侧链开源研究：https://github.com/ElementsProject/elements



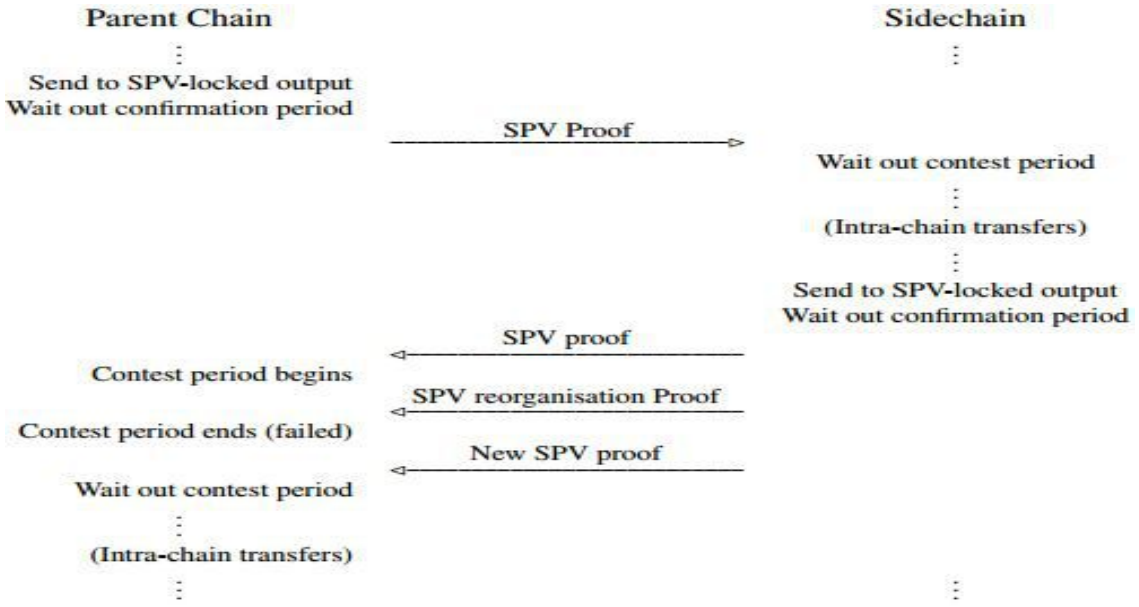


Figure 1: Example two-way peg protocol.

兼谈POW及POS

- POW是通过支付一定的工作量，来获得投票权力。POS是通过答应延缓消费你手中的财富来获得投票权力。
- 结算工作最核心的要素是什么？结算系统是处理债权和债务的规则，其核心要求就是规则的承担者有担保能力。
- POW工作量证明可以保证区块链上的信息真实性。但结算不是靠信息的真实性来做保的，而是通过抵押做保的。是要让债权方持有方即使是债务方破产倒闭都可以收回债务的信心来促成交易的。
- POS天然拥有担保能力，因为它是通过延迟消费手上的币龄来记账的。拥有记账的人一定要拥有币龄。而这币龄是无法伪造的。这就可以用来担保，这就可以实现结算的核心条件。权益证明要求证明人提供一定数量加密货币的所有权即可。
- POW（Proof Of Work-工作量证明）的代表：比特币
- POS（Proof Of Stake-权益证明) 的代表：Ethereum,Bitshares

分叉问题

·区块链 （block chain）本身也是一个线性的一串包含着交易信息的区块（blocks），一个接着一个，每一个block都包含着指向上一个block的指针。偶然情况下，一个blockchain上会产生分支。这种情况是因为，有时候两个挖矿者几乎同时验证出来一个区块的交易。他们同时公布到网络里，有些人用一个方法更新他们的blockchain，另一些人用另外一个方法更新他们的blockchain。这就造成了我们想要避免的情况 ——这种情况，交易的顺序就不清楚了，而且谁有哪个infocoin也就不清楚了。

·有一个简单的办法可以用来挪去分支。规则是这样的：如果分支情况出现，那么网络上的人们继续保持两个分支，任何情况下，挖矿者只在最长的那个blockchain上工作。这个过程保证了blockchain有统一的顺序。在比特币中，一个交易能不算作确认直到

·1）它存在于最长的分支中的block里，

·2）至少有5个验证过的block在其后面得到验证。这种情况，我们说这个交易有了“6个确认”。

·这给了整个网络时间去统一block的顺序。

·两个争议问题：硬分叉&软分叉

·硬分叉：

·是当比特币协议规则发生改变，旧节点拒绝接受由新节点创造的区块的情况。违反规则的区块将被忽视，矿工会按照他们的规则集，在他们最后见证的区块之后创建区块。“有争议”的硬分叉会导致比特币永久性地失去价值

·软分叉：

·是当比特币协议规则发生改变，旧的节点并不会意识到规则是不同的，它们将遵循改变后的规则集，继续接受由新节点创造的区块。矿工们可能会在他们完全没有理解，或者验证过的区块上进行工作。

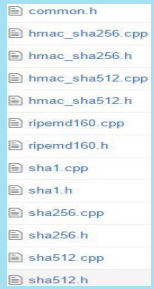
·软分叉带来的问题：出现旧软件继续处理数据，而无需进行升级的情况，这种”向前兼容“，会使运行一个节点的全部目标失效。在软分叉中，协议的改变是经过精心编制过的，本质上是在诱骗旧节点去相信一些东西是有效的。出现了一个称之为P2SH的比特币模式，P2SH是一个非常有用的模式，它可以让多重签名更易使用。问题就在于它的不安全：因为没有签名，所以当你广播了一笔，提供给任何人“密码”的交易，其他人也是可以进行一笔输出交易的，因为密码现在是公开的嘛，所以最后就存粹地变成了一场竞赛，看谁的交易第一个传递给矿工。P2SH之所以可行，是因为比特币协议进行了修改，包含了一条新规则：当你看到上述形式的输出脚本，实际上不要将它当作一个脚本看待，而是一直特殊的处理方式。软分叉，你将不会知道规则已经改变，这就好像是在盲目地飞行。

3 bitcoin 整体代码结构

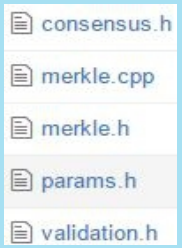
区块链基础架构举例-以Bitcoin为例(1)-其Github代码结构

1-工具类　　2-网络传输及协议类　　3-Blockchain类　　4-Bitcoin类

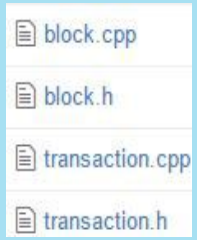
Crypto



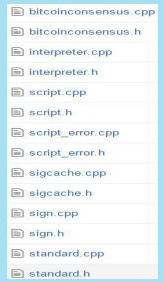
Consensus



Primitives

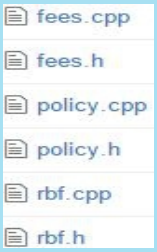


Script



Wallet

加密钱包实现



Bench

性能指标测试工具

Support

Test

单元测试

Compat

兼容性考虑：
Byteswap,Endian,glibc等

Policy

LevelDB

KV数据库

Qt

Qt GUI

secp256k1

椭圆形加密算法工具类

univalue

JSON格式的数据类型定义

ZMQ

ZeroMQ 消息队列

区块链基础架构举例-以Bitcoin代码为例(1)-1.工具

uint256 定义数据格式	**base58** 定义数据格式	**pubkey , key , keystore** 公钥，私钥	**hash** hash主要运算调用、/crypto/sha256.cpp里的CSHA256操作	**streams** 文件读写流
serialize 数据序列号辅助数据库使用	**compressor** 数据压缩	**prevector** boost集合Vector再封装	**random** 随机数运算	**timedata** 网络传输数据过滤调整，调整P2P同步时间
addrman 网络地址等操作封装	**sync** 系统操作同步预防死锁等	**bloom** Bloom is a probabilistic filter which SPV clients provide * so that we can filter the transactions we send them.	**utilstrencodings** 针对字符串的编码	**scheduler** 调用运算函数线程
dbwrapper 对leveldb的中间件层封装	**txdb** 比特币账户数据库封装	**txmempool** 数据交互操作的缓存池封装	**limitedmap** STL-like map container that only keeps the N elements with the highest value	**util** 底层操作的工具类实现
utiltime 时间操作	**reverselock** An RAII-style reverse lock. Unlocks on construction and locks on destruction			

区块链基础架构举例-以Bitcoin代码为例(2)-2. 网络传输及协议

net 网络节点操作	netbase 网络底层操作	protocol 比特币协议实现	torcontrol communicating with Tor	rest RPC操作的Restful封装
httprpc 服务器访问的RPC服务实现	httpserver 服务端实现	rpcclient RPC 客户端实现	rpcmisc RPC输出信息反馈	rpcnet RPC网络信息服务实现
rpcprotocol JSON-RPC protocol实现	rpcserver RPC服务端实现			

区块链基础架构举例-以Bitcoin代码为例(3)-3. Blockchain

chain Blockchain实现	**chainparams** Blockchain共识及基础封装实现	**chainparamsbase** CBaseChainParams defines the base parameters (shared between bitcoin-cli and bitcoind)of a given instance of the Bitcoin system	**rpcblockchain** Blockchain交互RPC实现	**merkleblock** 默克尔树实现区块链	**pow (Proof of work)** 区块链工作量证明实现
core_io 区块链核心封装	**core_read** 区块链底层标准协议层读取操作	**core_write** 区块链底层标准协议层写操作	**checkpoints** Block-chain 验证	**checkqueue** 区块链Queue for verifications	**rpcmining** 挖矿操作RPC实现
init 系统初始化	**main** 区块链主要操作实现封装	**validationinterface** 协议层验证封装了ZMQ实现	**rpcrawtransaction** RPC一些原始信息	**undo** 协议层Undo information	

区块链基础架构举例-以Bitcoin代码为例(4)-4. Bitcoin

amount 比特币钱包账户	**coins** 比特币实现	**coincontrol** 比特币简单操作	**bitcoin-cli** 比特币客户端实现	**bitcoind** 比特币服务端实现
bitcoin-tx 比特币协议层实现	**core_memusage** 比特币界面消息	**memusage** 比特币消息	**noui** 线层安全性检测	**alert** 消息提示
miner 比特币挖矿	**utilmoneystr** 比特币具体数额格式化	**clientversion** 客户的版本号		

4 bitcoin 数据结构解读

BlockChain 数据结构体

- Block 等(数据结构体)：参照 https://en.bitcoin.it/wiki/Protocol_documentation#block

block

The **block** message is sent in response to a getdata message which requests transaction information from a block hash.

Field Size	Description	Data type	Comments
4	version	int32_t	Block version information (note, this is signed)
32	prev_block	char[32]	The hash value of the previous block this particular block references
32	merkle_root	char[32]	The reference to a Merkle tree collection which is a hash of all transactions related to this block
4	timestamp	uint32_t	A Unix timestamp recording when this block was created (Currently limited to dates before the year 2106!)
4	bits	uint32_t	The calculated difficulty target being used for this block
4	nonce	uint32_t	The nonce used to generate this block… to allow variations of the header and compute different hashes
?	txn_count	var_int	Number of transaction entries
?	txns	tx[]	Block transactions, in format of "tx" command

- Chain(链接起来，运行起来，不断增加形成一个完全平等的P2P匿名大数据表)
- DHT(Distributed Hash Tables)参照 https://github.com/blockstack/reading-list#dht
- 点对点（去中心化且匿名安全）的通讯、交易。
- RPC（Remote Procedure Call Protocol）

Blockchain数据结构（CBlockIndex角度）

```
//https://github.com/bitcoin/bitcoin/blob/master/src/primitives/block.h

class CBlockHeader
{
public:
    // header
    static const int32_t CURRENT_VERSION=4;
    int32_t nVersion;
    uint256 hashPrevBlock;
    uint256 hashMerkleRoot;
    uint32_t nTime;
    uint32_t nBits;
    uint32_t nNonce;

class CBlock : public CBlockHeader
{
public:
    // network and disk
    std::vector<CTransaction> vtx;
    CBlock(){ SetNull(); }
    CBlock(const CBlockHeader &header){
        SetNull();
        *((CBlockHeader*)this) = header;
    }
    template <typename Stream, typename Operation
    inline void SerializationOp(Stream& s, Operation
ser_action, int nType, int nVersion){
        READWRITE(*(CBlockHeader*)this);
        READWRITE(vtx);
    }
```

```
https://github.com/bitcoin/bitcoin/blob/master/src/chain.h
class CBlockIndex
{
public:
 //! pointer to the hash of the block, if any. Memory is owned by this CBlockIndex
    const uint256* phashBlock;
    //! pointer to the index of the predecessor of this block
    CBlockIndex* pprev;
    //! pointer to the index of some further predecessor of this block
    CBlockIndex* pskip;
    //! height of the entry in the chain. The genesis block has height 0
    int nHeight;
    //! Which # file this block is stored in (blk?????.dat)
    int nFile;
    //! Byte offset within blk?????.dat where this block's data is stored
    unsigned int nDataPos;
    //! Byte offset within rev?????.dat where this block's undo data is stored
    unsigned int nUndoPos;
    arith_uint256 nChainWork;
     unsigned int nTx;
      unsigned int nChainTx;
    //! Verification status of this block. See enum BlockStatus
    unsigned int nStatus;
    //! block header
    int nVersion;
    uint256 hashMerkleRoot;
    unsigned int nTime;
    unsigned int nBits;
    unsigned int nNonce;
    //Sequential id assigned to distinguish order in which blocks are received.
    uint32_t nSequenceId;
CBlockIndex(const CBlockHeader& block){
        SetNull();
        nVersion       = block.nVersion;
        hashMerkleRoot = block.hashMerkleRoot;
        nTime          = block.nTime;
        nBits          = block.nBits;
        nNonce         = block.nNonce;
    }
```

```
https://github.com/bitcoin/bitcoin/blob/master/src/script/script.h
typedef prevector<28, unsigned char> CScriptBase;
/** Serialized script, used inside transaction inputs and outputs */
class CScript : public CScriptBase{
public:
 CScript() { }
 CScript(const CScript& b) :  CScriptBase(b.begin(), b.end()) { }
CScript(const_iterator pbegin, const_iterator pend) : CScriptBase(pbegin, pend) { }
CScript(std::vector<unsigned char>::const_iterator pbegin, std::vector<unsigned char>::const_iterator pend) : CScriptBase(pbegin, pend) { }
CScript(const unsigned char* pbegin, const unsigned char* pend) : CScriptBase(pbegin, pend) { }

//https://github.com/bitcoin/bitcoin/blob/master/src/primitives/transaction.h
class CTxIn
{
public:
    COutPoint prevout;
    CScript scriptSig;
    uint32_t nSequence;
class CTxOut
{
public:
    CAmount nValue;
    CScript scriptPubKey;
class CTransaction{
private:
    /** Memory only. */
    const uint256 hash;
    void UpdateHash() const;
public:
    static const int32_t   CURRENT_VERSION=1;
    const std::vector<CTxIn> vin;
    const std::vector<CTxOut> vout;
```

Blockchain数据结构（MerkleBlock角度）

```
//https://github.com/bitcoin/bitcoin/blob/master/src/primitives/block.h

class CBlockHeader
{
public:
    // header
    static const int32_t CURRENT_VERSION=4;
    int32_t nVersion;
    uint256 hashPrevBlock;
    uint256 hashMerkleRoot;
    uint32_t nTime;
    uint32_t nBits;
    uint32_t nNonce;

class CBlock : public CBlockHeader
{
public:
    // network and disk
    std::vector<CTransaction> vtx;
    CBlock(){ SetNull(); }
    CBlock(const CBlockHeader &header){
        SetNull();
        *((CBlockHeader*)this) = header;
    }
    template <typename Stream, typename Operation>
    inline void SerializationOp(Stream& s, Operation
ser_action, int nType, int nVersion){
        READWRITE(*(CBlockHeader*)this);
        READWRITE(vtx);
    }
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/merkleblock.h
class CMerkleBlock
{
public:
    /** Public only for unit testing */
    CBlockHeader header;
    CPartialMerkleTree txn;
template <typename Stream, typename Operation>
inline void SerializationOp(Stream& s, Operation ser_action, int nType, int
nVersion) {
        READWRITE(header);
        READWRITE(txn);
    }
CMerkleBlock(const CBlock& block, CBloomFilter& filter);
 // Create from a CBlock, matching the txids in the set
 CMerkleBlock(const CBlock& block, const std::set<uint256>& txids);

class CPartialMerkleTree
{
protected:
    /** the total number of transactions in the block */
    unsigned int nTransactions;
    /** node-is-parent-of-matched-txid bits */
    std::vector<bool> vBits;
    /** txids and internal hashes */
    std::vector<uint256> vHash;
unsigned int CalcTreeWidth(int height) {
        return (nTransactions+(1 << height)-1) >> height;
    }

  /** calculate the hash of a node in the merkle tree (at leaf level: the txid's t
hemselves) */
    uint256 CalcHash(int height, unsigned int pos, const std::vector<uint256>
&vTxid);

    /** recursive function that traverses tree nodes, storing the data as bits and
hashes */
    void TraverseAndBuild(int height, unsigned int pos, const
std::vector<uint256> &vTxid, const std::vector<bool> &vMatch);
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/script/script.h
typedef prevector<28, unsigned char> CScriptBase;
/** Serialized script, used inside transaction inputs and
outputs */
class CScript : public CScriptBase{
public:
 CScript() { }
 CScript(const CScript& b) :  CScriptBase(b.begin(),
b.end()) { }
CScript(const_iterator pbegin, const_iterator pend) :
CScriptBase(pbegin, pend) { }
CScript(std::vector<unsigned char>::const_iterator
pbegin, std::vector<unsigned char>::const_iterator
pend) : CScriptBase(pbegin, pend) { }
CScript(const unsigned char* pbegin, const unsigned
char* pend) : CScriptBase(pbegin, pend) { }
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/primitives/transaction.h
class CTxIn
{
public:
    COutPoint prevout;
    CScript scriptSig;
    uint32_t nSequence;
class CTxOut
{
public:
    CAmount nValue;
    CScript scriptPubKey;
class CTransaction{
private:
    /** Memory only. */
    const uint256 hash;
    void UpdateHash() const;
public:
    static const int32_t   CURRENT_VERSION=1;
    const int32_t nVersion;
    const std::vector<CTxIn> vin;
    const std::vector<CTxOut> vout;
```

Blockchain数据结构（CChain- An in-memory indexed chain of blocks）(1)

```
class CChain {
private:
    std::vector<CBlockIndex*> vChain;

public:
 /** Returns the index entry for the genesis block of this chain, or NULL if none. */
    CBlockIndex *Genesis() const {
        return vChain.size() > 0 ? vChain[0] : NULL;
    }

    /** Returns the index entry for the tip of this chain, or NULL if none. */
    CBlockIndex *Tip() const {
        return vChain.size() > 0 ? vChain[vChain.size() - 1] : NULL;
    }

   /** Returns the index entry at a particular height in this chain,
                    or NULL if no such height exists. */
    CBlockIndex *operator[](int nHeight) const {
        if (nHeight < 0 || nHeight >= (int)vChain.size())
            return NULL;
        return vChain[nHeight];
    }

    /** Compare two chains efficiently. */
    friend bool operator==(const CChain &a, const CChain &b) {
        return a.vChain.size() == b.vChain.size() &&
               a.vChain[a.vChain.size() - 1] == b.vChain[b.vChain.size() - 1];
    }
```

```
    /** Efficiently check whether a block is present in this chain. */
    bool Contains(const CBlockIndex *pindex) const {
        return (*this)[pindex->nHeight] == pindex;
    }

    /** Find the successor of a block in this chain,
              or NULL if the given index is not found or is the tip. */
    CBlockIndex *Next(const CBlockIndex *pindex) const {
        if (Contains(pindex))
            return (*this)[pindex->nHeight + 1];
        else
            return NULL;
    }

    /** Return the maximal height in the chain.
        Is equal to chain.Tip() ? chain.Tip()->nHeight : -1. */
    int Height() const {
        return vChain.size() - 1;
    }

    /** Set/initialize a chain with a given tip. */
    void SetTip(CBlockIndex *pindex);

    /** Return a CBlockLocator that refers to a block in this chain
                                                  (by default the tip). */
    CBlockLocator GetLocator(const CBlockIndex *pindex = NULL) const;

    /** Find the last common block between this chain and a block index entry. */
    const CBlockIndex *FindFork(const CBlockIndex *pindex) const;
};
```

Blockchain数据结构（CChain- An in-memory indexed chain of blocks）(2)

```
/** Set/initialize a chain with a given tip. */
void CChain::SetTip(CBlockIndex *pindex) {
    if (pindex == NULL) {
        vChain.clear();
        return;
    }
    vChain.resize(pindex->nHeight + 1);
    while (pindex && vChain[pindex->nHeight] != pindex) {
        vChain[pindex->nHeight] = pindex;
        pindex = pindex->pprev;
    }
}

 /** Find the last common block between this chain and a block index entry. */
const CBlockIndex *CChain::FindFork(const CBlockIndex *pindex) const {
    if (pindex == NULL) {
        return NULL;
    }
    if (pindex->nHeight > Height())
        pindex = pindex->GetAncestor(Height());
    while (pindex && !Contains(pindex))
        pindex = pindex->pprev;
    return pindex;
}
```

```
/** Return a CBlockLocator that refers to a block in this chain
                                        (by default the tip). */

CBlockLocator CChain::GetLocator(const CBlockIndex *pindex) const {
    int nStep = 1;
    std::vector<uint256> vHave;
    vHave.reserve(32);

    if (!pindex)
        pindex = Tip();
    while (pindex) {
        vHave.push_back(pindex->GetBlockHash());
        // Stop when we have added the genesis block.
        if (pindex->nHeight == 0)
            break;
        // Exponentially larger steps back, plus the genesis block.
        int nHeight = std::max(pindex->nHeight - nStep, 0);
        if (Contains(pindex)) {
            // Use O(1) CChain index if possible.
            pindex = (*this)[nHeight];
        } else {
            // Otherwise, use O(log n) skiplist.
            pindex = pindex->GetAncestor(nHeight);
        }
        if (vHave.size() > 10)
            nStep *= 2;
    }

    return CBlockLocator(vHave);
}
```

5 Hash算法解读

区块再说明及Hash算法(1)

区块序号：	#0
时间戳：	2009-1-3 18:15:05
上一区块HASH：	0000000000000000000000000000000000000000000000000000000000000000
本区块HASH：	000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
随机数：	2083236893
交易列表：	新生块奖励50BTC -> 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa

比特币第一个#0创世区块结构，创建出此区块需要计算来得到随机数。而作为找到随机数的奖励，将50BTC发给了第一个拥有比特币的地址1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa。这个创世地址较确定为中本聪拥有。

区块序号：	#1
时间戳：	2009-1-9 2:54:25
上一区块HASH：	000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
本区块HASH：	00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048
随机数：	2573394689
交易列表：	新生块奖励50BTC -> 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX

区块#1的“上一区块Hash值"与区块#0的"本区块Hash值"一致

区块序号：	#2
时间戳：	2009-1-9 2:55:44
上一区块HASH：	00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048
本区块HASH：	000000006a625f06636b8bb6ac7b960a8d03705d1ace08b1a19da3fdcc99ddbd
随机数：	1639830024
交易列表：	新生块奖励50BTC -> 1HLoD9E4SDFFPDiYfNYnkBLQ85Y51J3Zb1

区块#2的”上一区块Hash值“与区块#1的”本区块Hash值“一致。

2015年12月14日挖到的较新的区块#388388

区块序号：	#388388
时间戳：	2015-12-14 18:00:59
上一区块HASH：	00000000000000000cdcdc57154cf349fa8ad24831336abe583df544d59e196
本区块HASH：	0000000000000000012344120eba5bb76d4097087fe82c69117949c9d800ced4
随机数：	1796740019
交易列表：	新生块奖励25BTC+0.14002479 BTC手续费 -> 1PE7LXcntsLvavM2KXpJGNu51UbDhC3u63 1Gt9XeDYxLhwMJRBXjAcWCPwEvgoR2tbY -> 1NKQ7bP4UdPoUc8zk4Zduw9Jep1Biqm55P 100 BTC -> 13n7wTHweu6DcdMHs1tHrKM6Eb1X8phc9T 126.99997209 BTC

#388388区块，是近期2015年12月14日挖到的较新的区块。与早期的区块大体相同。不同点有：

1） HASH值结构，由开始的8个零，变成现在的17个零开头了。每增加1个零理论难度提高16倍可以想象现在打包的难度比诞生时提高了多少倍。（增加了144倍的难度！）

2） 新生区块奖励由50BTC降到了25BTC。这个是共识开源币协议决定的（防止货币通货膨胀!），每当挖完210000个区块大约四年时间后，就新生币量减半。以此来实现币量上限2100万BTC。

3） 除了新生奖励币外，还增加了0.14002479BTC手续费。这是由下方绿色背景的币交易的交易手续费提供的。这些币交易总输出币量不大于总输入币量，之间的差值就作为手续费，奖励给找到随机数打包此区块的人。虽然现在总手续费与新生币25BTC相比很小。但随着未来新生币的多次减半，手续费会渐渐成为打包新区块增加区块链长度的主要动力。

区块再说明及Hash算法(2)

区块链结构设计好后，怎么保证遵守有条不紊且避免攻击的增加新区块，以及由谁来增加新区块使区块链延续下去。这就要用到**POW算力竞争打包技术，即俗称的挖矿技术**。

HASH算法在将各个区块串来时起到了关键作用。

整个区块有id区块号，time时间戳，tx交易，random随机数等等，这些数据经过一个算法后压缩变成一个随机HASH字串，这个算法即是属于HASH算法。具体算法有很多比特币采用的是SHA256算法的HASH算法。每当任何输入信息任何微小的变化，都会引起HASH字串的完全变化。因此只要调整一个如random随机数，就可以每改变一次随机数就会有个新HASH字串。

所以**矿工去不断地自动更换尝试不同的随机数，换了随机数后计算整个区块的HASH。看看这个HASH前几位是不是零**。按概率每尝试计算16次预期便会有 16^{17} 次第一位是零。预期大约尝试计算16*16*16=4096次，会有一次出现前3位是零的情况。比特币目前难度要求的是前17位都是零。需要　　次运算尝试。因此需要全球各地的高算力矿机一起协作，以矿池为合作单位来合力尝试来做到，每10分钟左右全球找到一次。找到能满足当前难度条件的随机数，在全网公布出来便会得到25BTC新币加打包交易的手续费，作为给矿池打包交易增加更新维护区块新记录的奖励。**矿池再按算力贡献将奖励分给矿工（POW）**。

算力攻防-区块链安全的基石

51攻击就是拥有超半数全网算力（或虚拟算力，最好51%以上）的人或机构，主动发起的一种试图控制区块链，将交易记录按自己的意原来恶意打包成特定新区块的攻击行为。

而对51攻击预御方法，就是增加正常诚实算力，提高攻击者获得超一半全网算力的难度，另对现有的矿池算力，进行监控，当有矿池接近50%算力时，要警示矿工换其他矿池，预防其被控制由好矿池转变为攻击矿池。

明白以下几点，就会发现比特币的区块链，对于51攻击无需恐惧。

1） 51攻击不会修改比特币的历史区块链数据，只是可能对刚写入区块链没多久的新区块链数据，在攻击成功时，有一定的修改可能。因每个历史区块区块都需很高算力来寻找随机数打包，修改历史区块数据，需要从修改处重新打包，而这个算力需求是很高的。故**很重要的交易，一般等待6个确认的时间即可认为进入历史交易，而不再怕51攻击，确认越多越不怕**。

2） **比特币的算力矿池分布几乎都是实名公开的企业，不存在较大隐身算力**。即若真有51攻击，根据算力可以轻松地定位是哪个矿池在攻击，从而呼吁矿工离开此攻击矿池，即可中止51攻击。

3） 配置超过现有全网算力的矿机，将会是一笔很大的投入。且假设发现，那可以代码层面POW转POS模式，然后再转新算法POW。从而使整个矿业矿机归零。投入配置超现有全网算力的矿机因只能算SHA256的HASH算法而变为类似废品。**投入太高，且无攻击成功的预期**，从而不会攻击。这些算力直接加入正常挖矿反而获利更多。

区块再说明及Hash算法(3)

```
//https://github.com/bitcoin/bitcoin/blob/master/src/consensus/merkle.cpp

CHash256().Write(inner[level].begin(), 32).Write(h.begin(), 32).Finalize(h.begin());
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/hash.h
/** A hasher class for Bitcoin's 256-bit hash (double SHA-256). */
class CHash256 {
private:
    CSHA256 sha;
public:
    static const size_t OUTPUT_SIZE = CSHA256::OUTPUT_SIZE;
    void Finalize(unsigned char hash[OUTPUT_SIZE]) {
        unsigned char buf[sha.OUTPUT_SIZE];
        sha.Finalize(buf);
        sha.Reset().Write(buf, sha.OUTPUT_SIZE).Finalize(hash);
    }
    CHash256& Write(const unsigned char *data, size_t len) {
        sha.Write(data, len);
        return *this;
    }
    CHash256& Reset() {
        sha.Reset();
        return *this;
    }
};
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/crypto/sha256.cpp

CSHA256& CSHA256::Write(const unsigned char* data, size_t len){
    const unsigned char* end = data + len;
    size_t bufsize = bytes % 64;
    if (bufsize && bufsize + len >= 64) {
        // Fill the buffer, and process it.
        memcpy(buf + bufsize, data, 64 - bufsize);
        bytes += 64 - bufsize;
        data += 64 - bufsize;
        sha256::Transform(s, buf);
        bufsize = 0;
    }
    while (end >= data + 64) {
        // Process full chunks directly from the source.
        sha256::Transform(s, data);
        bytes += 64;
        data += 64;
    }
    if (end > data) {
        // Fill the buffer with what remains.
        memcpy(buf + bufsize, data, end - data);
        bytes += end - data;
    }
    return *this;
}
```

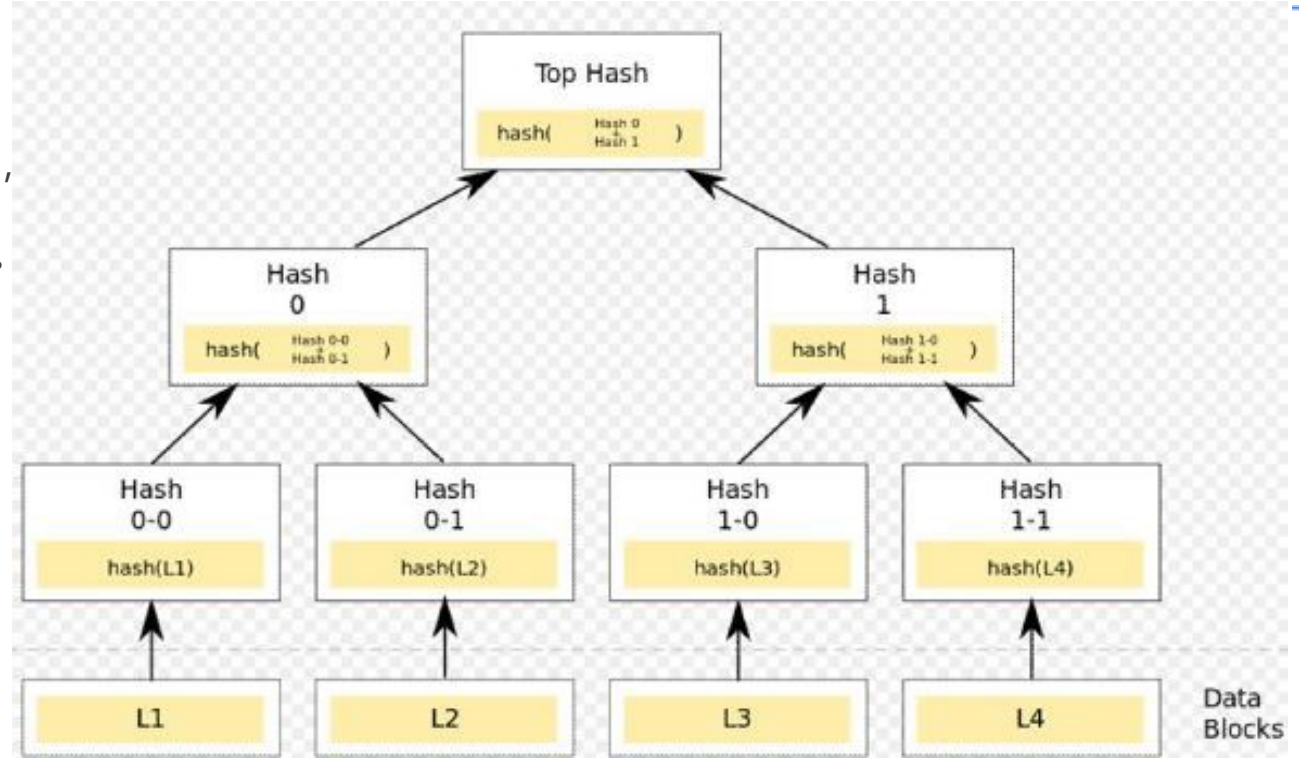
```
void CSHA256::Finalize(unsigned char hash[OUTPUT_SIZE]){
    static const unsigned char pad[64] = {0x80};  unsigned char sizedesc[8];
    WriteBE64(sizedesc, bytes << 3);
    Write(pad, 1 + ((119 - (bytes % 64)) % 64));
    Write(sizedesc, 8);
    WriteBE32(hash, s[0]);
    WriteBE32(hash + 4, s[1]);
    WriteBE32(hash + 8, s[2]);
    WriteBE32(hash + 12, s[3]);
    WriteBE32(hash + 16, s[4]);
    WriteBE32(hash + 20, s[5]);
    WriteBE32(hash + 24, s[6]);
    WriteBE32(hash + 28, s[7]);
}
```

```
//详细参见https://github.com/bitcoin/bitcoin/blob/master/src/crypto/sha256.cpp
/** Perform one SHA-256 transformation, processing a 64-byte chunk. */
void Transform(uint32_t* s, const unsigned char* chunk)
{
    uint32_t a = s[0], b = s[1], c = s[2], d = s[3], e = s[4], f = s[5], g = s[6], h = s[7];
    uint32_t w0, w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14, w15;
    Round(a, b, c, d, e, f, g, h, 0x428a2f98, w0 = ReadBE32(chunk + 0));
    Round(h, a, b, c, d, e, f, g, 0x71374491, w1 = ReadBE32(chunk + 4));
    Round(g, h, a, b, c, d, e, f, 0xb5c0fbcf, w2 = ReadBE32(chunk + 8));
    Round(f, g, h, a, b, c, d, e, 0xe9b5dba5, w3 = ReadBE32(chunk + 12));
    Round(e, f, g, h, a, b, c, d, 0x3956c25b, w4 = ReadBE32(chunk + 16));
    Round(d, e, f, g, h, a, b, c, 0x59f111f1, w5 = ReadBE32(chunk + 20));
    ......
    s[0] += a;   s[1] += b;  s[2] += c;  s[3] += d;   s[4] += e;    s[5] += f;    s[6] += g;  s[7] += h;
}
```

6 Merkel树实现解读

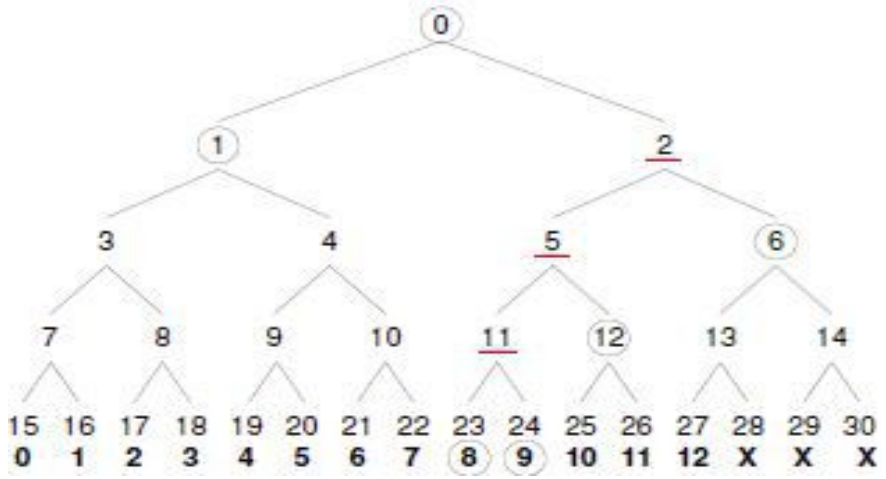
Blockchain核心算法介绍(1)-默克尔树

Merkle Tree：
在最底层，和哈希列表一样，我们把数据分成小的数据块，有相应地哈希和它对应。但是往上走，并不是直接去运算根哈希，而是把相邻的两个哈希合并成一个字符串，然后运算这个字符串的哈希，这样每两个哈希就结婚生子，得到了一个"子哈希"。如果最底层的哈希总数是单数，那到最后必然出现一个单身哈希，这种情况就直接对它进行哈希运算，所以也能得到它的子哈希。于是往上推，依然是一样的方式，可以得到数目更少的新一级哈希，最终必然形成一棵倒挂的树，到了树根的这个位置。这一代就剩下一个根哈希了。我们把它叫做 Merkle root.



Merkle Tree是基于数据HASH构建的一个树。它具有以下几个特点：

1、数据结构是一个树，可以是二叉树，也可以是多叉树。
2、Merkle Tree的叶子节点的value是数据集合的单元数据或者单元数据HASH。叶子节点的Hash值是真实数据的Hash值.
3、Merke Tree非叶子节点value是其所有子节点value的HASH值。这些非叶子节点的Hash被称作路径哈希值。



H11 = SHA256(H23 + H24)
H5 = SHA256(H11 + H12)
H2 = SHA256(H5 + H6)
H0 = SHA256(H1 + H2)

注：其中的 '+'表示字符连接运算

Merkle Tree在BT中应用

在BT中, 通常种子文件中包含的信息是Root值, 此外还有文件长度、数据块长等重要信息.
当客户端下载数据块8时，在下载前，它将要求peer提供校验块8所需的全部路径哈希值：H24、H12、H6和H1. 下载完成后, 客户端就会开始校验, 它先计算它已经下载的数据块8的Hash值23, 记做H23′, 表示尚未验证. 随后会按照左边的几个公式, 来依次求解 直到得到H0′并与H0做比较, 校验通过则下载无误. 校验通过的这些路径哈希值会被缓存下来, 当一定数量的路径哈希值被缓存之后，后继数据块的校验过程将被极大简化。此时我们可以直接利用校验路径上层次最低的已知路径哈希值来对数据块进行部分校验，而无需每次都校验至根哈希值H0.

Blockchain核心算法介绍(1)-1.1创世区块链默克尔树的创建

```
/** Build the genesis block. Note that the output of its generation  transaction cannot be spent since it did not originally exist in the database.
 * CBlock(hash=000000000019d6, ver=1, hashPrevBlock=00000000000000, hashMerkleRoot=4a5e1e, nTime=1231006505, nBits=1d00ffff, nNonce=2083236893, vtx=1)
 * CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
 * CTxIn(COutPoint(000000, -1), coinbase 04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73)
 * CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
 * vMerkleTree: 4a5e1e
*/
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t nVersion, const CAmount& genesisReward)
{
    const char* pszTimestamp = "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks";
    const CScript genesisOutputScript = CScript() << ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f") << OP_CHECKSIG; //CScript()<< 代码见1.1
  return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime, nNonce, nBits, nVersion, genesisReward); //代码见下一页-1.2
}
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/script/script.h - 1.1
CScript& operator<<(const std::vector<unsigned char>& b)
{
    if (b.size() < OP_PUSHDATA1){
        insert(end(), (unsigned char)b.size());
    }
    else if (b.size() <= 0xff){
        insert(end(), OP_PUSHDATA1);
        insert(end(), (unsigned char)b.size());
    }
    else if (b.size() <= 0xffff){
        insert(end(), OP_PUSHDATA2);
        uint8_t data[2];
        WriteLE16(data, b.size());
        insert(end(), data, data + sizeof(data));
    }
    else{
        insert(end(), OP_PUSHDATA4);  //OP_PUSHDATA4 = 0x4e
        uint8_t data[4];
        WriteLE32(data, b.size());
        insert(end(), data, data + sizeof(data));
    }
    insert(end(), b.begin(), b.end());
    return *this;
}
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/crypto/common.h
void static inline WriteLE32(unsigned char* ptr, uint32_t x){
    *((uint32_t*)ptr) = htole32(x);
}
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/compat/endian.h
inline uint32_t htole32(uint32_t host_32bits){
    return bswap_32(host_32bits);
}
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/compat/byteswap.h
inline uint32_t bswap_32(uint32_t x){
    return (((x & 0xff000000U) >> 24) | ((x & 0x00ff0000U) >>  8) |
            ((x & 0x0000ff00U) <<  8) | ((x & 0x000000ffU) << 24));
}
```

Blockchain核心算法介绍(1)-1.2 创世区块链默克尔树的创建

```
//https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp  1.2
static CBlock CreateGenesisBlock(const char* pszTimestamp, const CScript& genesisOutputScript, uint32_t nTime,
uint32_t nNonce, uint32_t nBits, int32_t nVersion, const CAmount& genesisReward){
    CMutableTransaction txNew;
    txNew.nVersion = 1;
    txNew.vin.resize(1);
    txNew.vout.resize(1);
    txNew.vin[0].scriptSig = CScript() << 486604799 << CScriptNum(4) << std::vector<unsigned char>((const unsigned
char*)pszTimestamp, (const unsigned char*)pszTimestamp + strlen(pszTimestamp));
    txNew.vout[0].nValue = genesisReward;
    txNew.vout[0].scriptPubKey = genesisOutputScript;

    CBlock genesis;
    genesis.nTime    = nTime;
    genesis.nBits    = nBits;
    genesis.nNonce   = nNonce;
    genesis.nVersion = nVersion;
    genesis.vtx.push_back(txNew);
    genesis.hashPrevBlock.SetNull();
    genesis.hashMerkleRoot = BlockMerkleRoot(genesis);
    return genesis;
}
//https://github.com/bitcoin/bitcoin/blob/master/src/consensus/merkle.cpp
uint256 BlockMerkleRoot(const CBlock& block, bool* mutated)
{
    std::vector<uint256> leaves;
    leaves.resize(block.vtx.size());
    for (size_t s = 0; s < block.vtx.size(); s++) {
        leaves[s] = block.vtx[s].GetHash();
    }
    return ComputeMerkleRoot(leaves, mutated);
}
uint256 ComputeMerkleRoot(const std::vector<uint256>& leaves, bool* mutated) {
    uint256 hash;
    MerkleComputation(leaves, &hash, mutated, -1, NULL);
    return hash;
}
/* This implements a constant-space merkle root/path calculator, limited to 2^32 leaves. */
static void MerkleComputation(const std::vector<uint256>& leaves, uint256* proot, bool*  pmutated, uint32_t branchpos,
std::vector<uint256>* pbranch) {
    if (pbranch)   pbranch->clear();
    if (leaves.size() == 0) {
        if (pmutated) *pmutated = false;
        if (proot) *proot = uint256();
        return;
    }
    bool mutated = false;
    uint32_t count = 0;  // count is the number of leaves processed so far.
    uint256 inner[32];
    int matchlevel = -1;// Which position in inner is a hash that depends on the matching leaf.
    while (count < leaves.size()) {
        uint256 h = leaves[count];
        bool matchh = count == branchpos;
        count++;
        int level;
        for (level = 0; !(count & (((uint32_t)1) << level)); level++) {
            if (pbranch) {
                if (matchh) {
                    pbranch->push_back(inner[level]);
                } else if (matchlevel == level) {
                    pbranch->push_back(h);
                    matchh = true;
                }
            }
            mutated |= (inner[level] == h);
            CHash256().Write(inner[level].begin(), 32).Write(h.begin(), 32).Finalize(h.begin());
        }
        inner[level] = h;// Store the resulting hash at inner position level.
        if (matchh) {
            matchlevel = level;
        }
    }
    int   level = 0;
    while (!(count & (((uint32_t)1) << level))) {
            level++;
    }
    uint256 h = inner[level];
    bool matchh = matchlevel == level;
    while (count != (((uint32_t)1) << level)) {
        if (pbranch && matchh) {
            pbranch->push_back(h);
        }
        CHash256().Write(h.begin(), 32).Write(h.begin(), 32).Finalize(h.begin());
        count += (((uint32_t)1) << level);
        level++;
        while (!(count & (((uint32_t)1) << level))) {
            if (pbranch) {
                if (matchh) {
                    pbranch->push_back(inner[level]);
                } else if (matchlevel == level) {
                    pbranch->push_back(h);
                    matchh = true;
                }
            }
            CHash256().Write(inner[level].begin(), 32).Write(h.begin(), 32).Finalize(h.begin());
            level++;
        }
    }
    if (pmutated) *pmutated = mutated;
    if (proot) *proot = h;
}
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/primitives/transaction.h
/** A mutable version of CTransaction. */
struct CMutableTransaction{
    int32_t nVersion;
    std::vector<CTxIn> vin;
    std::vector<CTxOut> vout;
    uint32_t nLockTime;
     /** Compute the hash of this CMutableTransaction. This is computed on the
     * fly, as opposed to GetHash() in CTransaction, which uses a cached result.
     */
    uint256 GetHash() const;
};
```

```
CScript& operator<<(const std::vector<unsigned char>& b){
if (b.size() < OP_PUSHDATA1)
        {
            insert(end(), (unsigned char)b.size());
        }
        else if (b.size() <= 0xff)
        {
            insert(end(), OP_PUSHDATA1);
            insert(end(), (unsigned char)b.size());
        }
        else if (b.size() <= 0xffff)
        {
            insert(end(), OP_PUSHDATA2);
            uint8_t data[2];
        WriteLE16(data, b.size());
            insert(end(), data, data + sizeof(data));
        }
        else
        {
            insert(end(), OP_PUSHDATA4);
            uint8_t data[4];
        WriteLE32(data, b.size());
            insert(end(), data, data + sizeof(data));
        }
insert(end(), b.begin(), b.end());
        return *this;
    }
```

7 bitcoin 系统启动过程

Blockchain核心算法介绍(2) 系统初始化运行过程

1-setup
网络开启安装
信号准备，中断

2-parameter interactions
初始化参数

3-parameter-to-internal-flags
初始化系统硬件参数，验证存储空间

4-application initialization: dir lock, daemonize，pidfile, debug log
打开钱包账户文件，开启RPC server

5-verify wallet database integrity
验证钱包数据库完整有效性

6-network initialization
网络初始化，查询网路端节点

7-load block chain
加载区块链，

8-load wallet
加载钱包账户

9-data directory maintenance
钱包数据有效性维护

10-import blocks
导入区块数据文件

11-start node
开启P2P服务

12-finished
完成扫尾工作，RPC预热

Blockchain核心算法介绍(2) 系统初始化运行过程 - 1.

```
// ********************** Step 1: setup
bool AppInit2(boost::thread_group& threadGroup, CScheduler& scheduler){
if (!SetupNetworking())
        return InitError("Initializing networking failed");
// Clean shutdown on SIGTERM
    struct sigaction sa;
    sa.sa_handler = HandleSIGTERM;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGTERM, &sa, NULL);
    sigaction(SIGINT, &sa, NULL);
```

```
// ***************************** Step 2: parameter interactions
// also see: InitParameterInteraction()
    const CChainParams& chainparams = Params();
nMaxConnections = std::max(std::min(nMaxConnections,
(int)(FD_SETSIZE - nBind - MIN_CORE_FILEDESCRIPTORS)), 0);
```

```
// ***************************** Step 3: parameter-to-internal-flags
fCheckBlockIndex = GetBoolArg("-checkblockindex", chainparams.DefaultConsistencyChecks());
// mempool limits
 int64_t nMempoolSizeMax = GetArg("-maxmempool", DEFAULT_MAX_MEMPOOL_SIZE) * 1000000;
 int64_t nMempoolSizeMin = GetArg("-limitdescendantsize", DEFAULT_DESCENDANT_SIZE_LIMIT) * 1000 * 40;
 if (nMempoolSizeMax < 0 || nMempoolSizeMax < nMempoolSizeMin)
     return InitError(strprintf(_("-maxmempool must be at least %d MB"), std::ceil(nMempoolSizeMin / 1000000.0)));
// block pruning; get the amount of disk space (in MiB) to allot for block & undo files
    int64_t nSignedPruneTarget = GetArg("-prune", 0) * 1024 * 1024;
std::string strWalletFile = GetArg("-wallet", "wallet.dat");
```

```
// ***** Step 4: application initialization: dir lock, daemonize,  pidfile, debug log
// Wallet file must be a plain filename without a directory
 if (strWalletFile != boost::filesystem::basename(strWalletFile) + boost::filesystem::extension(strWalletFile))
       return InitError(strprintf(_("Wallet %s resides outside data directory %s"), strWalletFile, strDataDir));
// Make sure only a single Bitcoin process is using the data directory.
    boost::filesystem::path pathLockFile = GetDataDir() / ".lock";
 // empty lock file; created if it doesn't exist.
    FILE* file = fopen(pathLockFile.string().c_str(), "a");
if (!lock.try_lock())
    return InitError(strprintf(_("Cannot obtain a lock on data directory %s. Bitcoin Core is probably already running."),
strDataDir));
// Start the lightweight task scheduler thread
    CScheduler::Function serviceLoop = boost::bind(&CScheduler::serviceQueue, &scheduler);
    threadGroup.create_thread(boost::bind(&TraceThread<CScheduler::Function>, "scheduler", serviceLoop));
/* Start the RPC server already.*/
if (fServer)
    {
        uiInterface.InitMessage.connect(SetRPCWarmupStatus);
        if (!AppInitServers(threadGroup))
            return InitError(_("Unable to start HTTP server. See debug log for details."));
    }
```

```
// ****************************** Step 5: verify wallet database integrity
if (!CWallet::Verify(strWalletFile, warningString, errorString))
            return false;
```

Blockchain核心算法介绍(2) 系统初始化运行过程 - 2.

```
// ******************** Step 6: network initialization
RegisterNodeSignals(GetNodeSignals());
std::set<enum Network> nets;
BOOST_FOREACH(const std::string& snet, mapMultiArgs["-onlynet"]) {
      enum Network net = ParseNetwork(snet);
      if (net == NET_UNROUTABLE)
         return InitError(strprintf(_("Unknown network specified in -onlynet: '%s'"), snet));
         nets.insert(net);
}
if (mapArgs.count("-whitelist")) {
       BOOST_FOREACH(const std::string& net, mapMultiArgs["-whitelist"]) {
          CSubNet subnet(net);
          if (!subnet.IsValid())
              return InitError(strprintf(_("Invalid netmask specified in -whitelist: '%s'"), net));
          CNode::AddWhitelistedRange(subnet);
       }
 }
struct in_addr inaddr_any;
inaddr_any.s_addr = INADDR_ANY;
fBound |= Bind(CService(in6addr_any, GetListenPort()), BF_NONE);
fBound |= Bind(CService(inaddr_any, GetListenPort()), !fBound ? BF_REPORT_ERROR : BF_NONE);
BOOST_FOREACH(const std::string& strDest, mapMultiArgs["-seednode"])
AddOneShot(strDest);
```

```
// Initialize the block index (no-op if non-empty database was already loaded)
     if (!InitBlockIndex(chainparams)) {//具体参见后面
                  strLoadError = _("Error initializing block database");   break;
     }
 LOCK(cs_main);
 CBlockIndex* tip = chainActive.Tip();
 if (tip && tip->nTime > GetAdjustedTime() + 2 * 60 * 60) {
  strLoadError = _("The block database contains a block which appears to be from the future. ")
break;
}
if (!CVerifyDB().VerifyDB(chainparams, pcoinsdbview, GetArg("-checklevel",
DEFAULT_CHECKLEVEL), GetArg("-checkblocks", DEFAULT_CHECKBLOCKS))) {
                  strLoadError = _("Corrupted block database detected");
                  break;
}
```

```
// ********************** Step 7: load block chain
// Upgrading to 0.8; hard-link the old blknnnn.dat files into /blocks/
    boost::filesystem::path blocksDir = GetDataDir() / "blocks";
if (!boost::filesystem::exists(blocksDir))
{
        boost::filesystem::create_directories(blocksDir);
        bool linked = false;
        for (unsigned int i = 1; i < 10000; i++) {
            boost::filesystem::path source = GetDataDir() / strprintf("blk%04u.dat", i);
            if (!boost::filesystem::exists(source)) break;
            boost::filesystem::path dest = blocksDir / strprintf("blk%05u.dat", i-1);
            try {
                boost::filesystem::create_hard_link(source, dest);
// cache size calculations
  nt64_t nTotalCache = (GetArg("-dbcache", nDefaultDbCache) << 20);
// total cache cannot be less than nMinDbCache
 nTotalCache = std::max(nTotalCache, nMinDbCache << 20);
 // total cache cannot be greated than nMaxDbcache
 nTotalCache = std::min(nTotalCache, nMaxDbCache << 20);
int64_t nBlockTreeDBCache = nTotalCache / 8;
do {
    try {
        UnloadBlockIndex();
        delete pblocktree;
        pblocktree = new CBlockTreeDB(nBlockTreeDBCache, false, fReindex);
if (fReindex) {
                    pblocktree->WriteReindexing(true);
//If  reindexing in prune mode, wipe away unusable block files and all undo data files
                    if (fPruneMode)
                        CleanupBlockRevFiles();
                    }
if (!LoadBlockIndex()) {
                    strLoadError = _("Error loading block database");  break;
                }
// If the loaded chain has a wrong genesis, bail out immediately
             if (!mapBlockIndex.empty() &&
mapBlockIndex.count(chainparams.GetConsensus().hashGenesisBlock) == 0)
                    return InitError(_("Incorrect or no genesis block found. Wrong datadir for
network?"));
```

Blockchain核心算法介绍(2) 系统初始化运行过程 - 3.

```
// ********************************* Step 8: load wallet
// needed to restore wallet transaction meta data after -zapwallettxes
std::vector<CWalletTx> vWtx;
pwalletMain = new CWallet(strWalletFile); bool fFirstRun = true;
DBErrors nLoadWalletRet = pwalletMain->LoadWallet(fFirstRun);
if (fFirstRun){
        // Create new keyUser and set as default key
        RandAddSeedPerfmon();
        CPubKey newDefaultKey;
        if (pwalletMain->GetKeyFromPool(newDefaultKey)) {
            pwalletMain->SetDefaultKey(newDefaultKey);
            if (!pwalletMain->SetAddressBook(pwalletMain->vchDefaultKey.GetID(), "", "receive"))
                strErrors << _("Cannot write default address") << "\n";
        }
    pwalletMain->SetBestChain(chainActive.GetLocator());
}
RegisterValidationInterface(pwalletMain);
CBlockIndex *pindexRescan = chainActive.Tip();
CWalletDB walletdb(strWalletFile);
CBlockLocator locator;
if (walletdb.ReadBestBlock(locator))
      pindexRescan = FindForkInGlobalIndex(chainActive, locator);
else
      pindexRescan = chainActive.Genesis();
BOOST_FOREACH(const CWalletTx& wtxOld, vWtx){
         uint256 hash = wtxOld.GetHash();
         std::map<uint256, CWalletTx>::iterator mi = pwalletMain->mapWallet.find(hash);
         if (mi != pwalletMain->mapWallet.end()){
                    const CWalletTx* copyFrom = &wtxOld;
                    CWalletTx* copyTo = &mi->second;
                    copyTo->mapValue = copyFrom->mapValue;
                    copyTo->vOrderForm = copyFrom->vOrderForm;
                    copyTo->nTimeReceived = copyFrom->nTimeReceived;
                    copyTo->nTimeSmart = copyFrom->nTimeSmart;
                    copyTo->fFromMe = copyFrom->fFromMe;
                    copyTo->strFromAccount = copyFrom->strFromAccount;
                    copyTo->nOrderPos = copyFrom->nOrderPos;
                    copyTo->WriteToDisk(&walletdb);
          }
}
```

```
//****** Step 9: data directory maintenance
// if pruning, unset the service bit and perform the initial blockstore prune
// after any wallet rescanning has taken place.
if (!fReindex) {
          uiInterface.InitMessage(_("Pruning blockstore..."));
          PruneAndFlush();
}
```

```
   // ************************* Step 10: import blocks
CValidationState state;
 if (!ActivateBestChain(state, chainparams))
       strErrors << "Failed to connect best block";
 std::vector<boost::filesystem::path> vImportFiles;
 if (mapArgs.count("-loadblock")){
       BOOST_FOREACH(const std::string& strFile, mapMultiArgs["-loadblock"])
       vImportFiles.push_back(strFile);
 }
 threadGroup.create_thread(boost::bind(&ThreadImport, vImportFiles));
```

```
// ************************ Step 11: start node
RandAddSeedPerfmon();
iStartNode(threadGroup, scheduler);
// Monitor the chain, and alert if we get blocks much quicker or slower than expected
nt64_t nPowTargetSpacing = Params().GetConsensus().nPowTargetSpacing;
CScheduler::Function f = boost::bind(&PartitionCheck, &IsInitialBlockDownload,
                  boost::ref(cs_main), boost::cref(pindexBestHeader), nPowTargetSpacing);
 scheduler.scheduleEvery(f, nPowTargetSpacing);
// Generate coins in the background
GenerateBitcoins(GetBoolArg("-gen", DEFAULT_GENERATE), GetArg("-genproclimit",
DEFAULT_GENERATE_THREADS), chainparams);
```

```
// ************************************ Step 12: finished
SetRPCWarmupFinished();
if (pwalletMain) {
       // Add wallet transactions that aren't already in a block to mapTransactions
       pwalletMain->ReacceptWalletTransactions();
 threadGroup.create_thread(boost::bind(&ThreadFlushWalletDB, boost::ref(pwalletMain-
>strWalletFile))); // Run a thread to flush wallet periodically
   }
```

Blockchain核心算法介绍(2) 系统初始化运行过程 - 4.1-Blockchain补充

```
bool InitBlockIndex(const CChainParams& chainparams)
{
    LOCK(cs_main);
    // Initialize global variables that cannot be constructed at startup.
    recentRejects.reset(new CRollingBloomFilter(120000, 0.000001));
    // Check whether we're already initialized
    if (chainActive.Genesis() != NULL)
        return true;
    // Use the provided setting for -txindex in the new database
    fTxIndex = GetBoolArg("-txindex", DEFAULT_TXINDEX);
    pblocktree->WriteFlag("txindex", fTxIndex);
    LogPrintf("Initializing databases...\n");
    // Only add the genesis block if not reindexing (in which case we reuse the one already on disk)
    if (!fReindex) {
        try {
            CBlock &block = const_cast<CBlock&>(chainparams.GenesisBlock());//4.1
            // Start new block file
            unsigned int nBlockSize = ::GetSerializeSize(block, SER_DISK, CLIENT_VERSION);
            CDiskBlockPos blockPos;
            CValidationState state;
            if (!FindBlockPos(state, blockPos, nBlockSize+8, 0, block.GetBlockTime()))//4.3
                return error("LoadBlockIndex(): FindBlockPos failed");
            if (!WriteBlockToDisk(block, blockPos, chainparams.MessageStart()))//4.4
                return error("LoadBlockIndex(): writing genesis block to disk failed");
            CBlockIndex *pindex = AddToBlockIndex(block); //4.5
            if (!ReceivedBlockTransactions(block, state, pindex, blockPos)) //4.6
                return error("LoadBlockIndex(): genesis block not accepted");
            if (!ActivateBestChain(state, chainparams, &block)) //4.7见后面proof of work部分
                return error("LoadBlockIndex(): genesis block cannot be activated");
            // Force a chainstate write so that when we VerifyDB in a moment, it doesn't check stale data
            return FlushStateToDisk(state, FLUSH_STATE_ALWAYS); //4.8
        } catch (const std::runtime_error& e) {
            return error("LoadBlockIndex(): failed to initialize block database: %s", e.what());
        }
    }
    return true;
}
```

```
//代码4.3
bool FindBlockPos(CValidationState &state,
                  CDiskBlockPos &pos, unsigned int nAddSize, unsigned int nHeight,
                  uint64_t nTime, bool fKnown = false)
{
    LOCK(cs_LastBlockFile);
    unsigned int nFile = fKnown ? pos.nFile : nLastBlockFile;
    if (vinfoBlockFile.size() <= nFile)    vinfoBlockFile.resize(nFile + 1);
    if (!fKnown) {
        while (vinfoBlockFile[nFile].nSize + nAddSize >= MAX_BLOCKFILE_SIZE) {
            nFile++;
            if (vinfoBlockFile.size() <= nFile)  vinfoBlockFile.resize(nFile + 1);
        }
        pos.nFile = nFile;
        pos.nPos = vinfoBlockFile[nFile].nSize;
    }
    if ((int)nFile != nLastBlockFile) {
        if (!fKnown)   LogPrintf("Leaving block file %i: %s\n", nLastBlockFile, vinfoBlockFile[nLastBlockFile].ToString());
        FlushBlockFile(!fKnown);
        nLastBlockFile = nFile;
    }
    vinfoBlockFile[nFile].AddBlock(nHeight, nTime);
    if (fKnown)
        vinfoBlockFile[nFile].nSize = std::max(pos.nPos + nAddSize, vinfoBlockFile[nFile].nSize);
    else
        vinfoBlockFile[nFile].nSize += nAddSize;
    if (!fKnown) {
        unsigned int nOldChunks = (pos.nPos + BLOCKFILE_CHUNK_SIZE - 1) / BLOCKFILE_CHUNK_SIZE;
        unsigned int nNewChunks = (vinfoBlockFile[nFile].nSize + BLOCKFILE_CHUNK_SIZE - 1) /
BLOCKFILE_CHUNK_SIZE;
        if (nNewChunks > nOldChunks) {
            if (fPruneMode)   fCheckForPruning = true;
            if (CheckDiskSpace(nNewChunks * BLOCKFILE_CHUNK_SIZE - pos.nPos)) {
                FILE *file = OpenBlockFile(pos);
                if (file) {
                    LogPrintf("Pre-allocating up to position 0x%x in blk%05u.dat\n", nNewChunks *
BLOCKFILE_CHUNK_SIZE, pos.nFile);
                    AllocateFileRange(file, pos.nPos, nNewChunks * BLOCKFILE_CHUNK_SIZE - pos.nPos);
                    fclose(file);
                }
            }
            else
                return state.Error("out of disk space");
        }
    }
    setDirtyFileInfo.insert(nFile);
    return true;
}
```

Blockchain核心算法介绍(2) 系统初始化运行过程 - 4.2-Blockchain补充

```
//4.4代码
bool WriteBlockToDisk(const CBlock& block, CDiskBlockPos& pos,
                      const CMessageHeader::MessageStartChars& messageStart)
{
    // Open history file to append
    CAutoFile fileout(OpenBlockFile(pos), SER_DISK, CLIENT_VERSION);
    if (fileout.IsNull())
        return error("WriteBlockToDisk: OpenBlockFile failed");
    // Write index header
    unsigned int nSize = fileout.GetSerializeSize(block);
    fileout << FLATDATA(messageStart) << nSize;
    // Write block
    long fileOutPos = ftell(fileout.Get());
    if (fileOutPos < 0)
        return error("WriteBlockToDisk: ftell failed");
    pos.nPos = (unsigned int)fileOutPos;
    fileout << block;
    return true;
}
```

```
class CChainParams
{
    const CBlock& GenesisBlock() const { return genesis; }

    protected:
                CBlock genesis;
}
```

```
//4.5代码
CBlockIndex* AddToBlockIndex(const CBlockHeader& block)
{
    // Check for duplicate
    uint256 hash = block.GetHash();
    BlockMap::iterator it = mapBlockIndex.find(hash);
    if (it != mapBlockIndex.end())
        return it->second;
    // Construct new block index object
    CBlockIndex* pindexNew = new CBlockIndex(block);
    assert(pindexNew);
    // We assign the sequence id to blocks only when the full data is available,
    // to avoid miners withholding blocks but broadcasting headers, to get a
    // competitive advantage.
    pindexNew->nSequenceId = 0;
    BlockMap::iterator mi = mapBlockIndex.insert(make_pair(hash, pindexNew)).first;
    pindexNew->phashBlock = &((*mi).first);
    BlockMap::iterator miPrev = mapBlockIndex.find(block.hashPrevBlock);
    if (miPrev != mapBlockIndex.end()){
        pindexNew->pprev = (*miPrev).second;
        pindexNew->nHeight = pindexNew->pprev->nHeight + 1;
        pindexNew->BuildSkip();
    }
    pindexNew->nChainWork = (pindexNew->pprev ? pindexNew->pprev->nChainWork : 0) +
GetBlockProof(*pindexNew);
    pindexNew->RaiseValidity(BLOCK_VALID_TREE);
    if (pindexBestHeader == NULL || pindexBestHeader->nChainWork < pindexNew->nChainWork)
        pindexBestHeader = pindexNew;
    setDirtyBlockIndex.insert(pindexNew);
    return pindexNew;
}
```

Blockchain核心算法介绍(2) 系统初始化运行过程 - 4.3-Blockchain补充

```
//4.6代码
/** Mark a block as having its data received and checked (up to BLOCK_VALID_TRANSACTIONS). */
bool ReceivedBlockTransactions(const CBlock &block, CValidationState& state,
                               CBlockIndex *pindexNew, const CDiskBlockPos& pos){
    pindexNew->nTx = block.vtx.size();
    pindexNew->nChainTx = 0;
    pindexNew->nFile = pos.nFile;
    pindexNew->nDataPos = pos.nPos;
    pindexNew->nUndoPos = 0;
    pindexNew->nStatus |= BLOCK_HAVE_DATA;
    pindexNew->RaiseValidity(BLOCK_VALID_TRANSACTIONS);
    setDirtyBlockIndex.insert(pindexNew);
    if (pindexNew->pprev == NULL || pindexNew->pprev->nChainTx) {
        // If pindexNew is the genesis block or all parents are BLOCK_VALID_TRANSACTIONS.
        deque<CBlockIndex*> queue;
        queue.push_back(pindexNew);
        // Recursively process any descendant blocks that now may be eligible to be connected.
        while (!queue.empty()) {
            CBlockIndex *pindex = queue.front();
            queue.pop_front();
            pindex->nChainTx = (pindex->pprev ? pindex->pprev->nChainTx : 0) + pindex->nTx;
            pindex->nSequenceId = nBlockSequenceId++;
          if (chainActive.Tip() == NULL || !setBlockIndexCandidates.value_comp()(pindex,
chainActive.Tip())) {     setBlockIndexCandidates.insert(pindex);  }
    std::pair<std::multimap<CBlockIndex*, CBlockIndex*>::iterator,
    std::multimap<CBlockIndex*, CBlockIndex*>::iterator> range =
mapBlocksUnlinked.equal_range(pindex);
            while (range.first != range.second) {
                std::multimap<CBlockIndex*, CBlockIndex*>::iterator it = range.first;
                queue.push_back(it->second);
                range.first++;
                mapBlocksUnlinked.erase(it);
            }
        }//end of while
    } else {
        if (pindexNew->pprev && pindexNew->pprev->IsValid(BLOCK_VALID_TREE)) {
            mapBlocksUnlinked.insert(std::make_pair(pindexNew->pprev, pindexNew));
        }
    } return true;
}
```

```
/**The set of all CBlockIndex entries with BLOCK_VALID_TRANSACTIONS (for itself and all
ancestors) and as good as our current tip or better. Entries may be failed, though, and pruning
nodes may be missing the data for the block.
 */
set<CBlockIndex*, CBlockIndexWorkComparator> setBlockIndexCandidates;
```

```
/** Update the on-disk chain state.
* The caches and indexes are flushed depending on the mode we're called with
 * if they're too large, if it's been a while since the last write,
 * or always and in all cases if we're in prune mode and are deleting files.
 */
bool static FlushStateToDisk(CValidationState &state, FlushStateMode mode) {
    const CChainParams& chainparams = Params();
    LOCK2(cs_main, cs_LastBlockFile);
    static int64_t nLastWrite = 0;
    static int64_t nLastFlush = 0;
    static int64_t nLastSetChain = 0;
    std::set<int> setFilesToPrune;
    bool fFlushForPrune = false;
    try {
    if (fPruneMode && fCheckForPruning && !fReindex) {
        FindFilesToPrune(setFilesToPrune, chainparams.PruneAfterHeight());
        fCheckForPruning = false;
        if (!setFilesToPrune.empty()) {
            fFlushForPrune = true;
            if (!fHavePruned) {
                pblocktree->WriteFlag("prunedblockfiles", true);
                fHavePruned = true;
            }
        }
    }
    int64_t nNow = GetTimeMicros();
```

Blockchain核心算法介绍(2) 系统初始化运行过程 - 4.4-Blockchain补充

```
// Avoid writing/flushing immediately after startup.
    if (nLastWrite == 0) {   nLastWrite = nNow;  }
    if (nLastFlush == 0) {  nLastFlush = nNow;  }
    if (nLastSetChain == 0) {  nLastSetChain = nNow;  }
    size_t cacheSize = pcoinsTip->DynamicMemoryUsage();
    // The cache is large and close to the limit, but we have time now (not in the middle of a block processing).
    bool fCacheLarge = mode == FLUSH_STATE_PERIODIC && cacheSize * (10.0/9) > nCoinCacheUsage;
    // The cache is over the limit, we have to write now.
    bool fCacheCritical = mode == FLUSH_STATE_IF_NEEDED && cacheSize > nCoinCacheUsage;
    // It's been a while since we wrote the block index to disk. Do this frequently, so we don't need to redownload after a crash.
    bool fPeriodicWrite = mode == FLUSH_STATE_PERIODIC && nNow > nLastWrite + (int64_t)DATABASE_WRITE_INTERVAL * 1000000;
    // It's been very long since we flushed the cache. Do this infrequently, to optimize cache usage.
    bool fPeriodicFlush = mode == FLUSH_STATE_PERIODIC && nNow > nLastFlush + (int64_t)DATABASE_FLUSH_INTERVAL * 1000000;
    // Combine all conditions that result in a full cache flush.
    bool fDoFullFlush = (mode == FLUSH_STATE_ALWAYS) || fCacheLarge || fCacheCritical || fPeriodicFlush || fFlushForPrune;
    // Write blocks and block index to disk.
    if (fDoFullFlush || fPeriodicWrite)
    {  // Depend on nMinDiskSpace to ensure we can write block index
        if (!CheckDiskSpace(0))    return state.Error("out of disk space");
        // First make sure all block and undo data is flushed to disk.
        FlushBlockFile();
        // Then update all block file information (which may refer to block and undo files).
            std::vector<std::pair<int, const CBlockFileInfo*> > vFiles;
            vFiles.reserve(setDirtyFileInfo.size());
            for (set<int>::iterator it = setDirtyFileInfo.begin(); it != setDirtyFileInfo.end(); ) {
                vFiles.push_back(make_pair(*it, &vinfoBlockFile[*it]));
                setDirtyFileInfo.erase(it++);
            }
            std::vector<const CBlockIndex*> vBlocks;
            vBlocks.reserve(setDirtyBlockIndex.size());
            for (set<CBlockIndex*>::iterator it = setDirtyBlockIndex.begin(); it != setDirtyBlockIndex.end(); ) {
                vBlocks.push_back(*it);
                setDirtyBlockIndex.erase(it++);
            }
            if (!pblocktree->WriteBatchSync(vFiles, nLastBlockFile, vBlocks)) { return AbortNode(state, "Files to write to block index database");  }
       // Finally remove any pruned files
        if (fFlushForPrune)    UnlinkPrunedFiles(setFilesToPrune);
        nLastWrite = nNow;
    }
```

```
// Flush best chain related state. This can only be done if
// the blocks / block index write was also done.
if (fDoFullFlush) {
 // Typical CCoins structures on disk are around 128
//bytes in size.Pushing a new one to the database can
//cause it to be written twice (once in the log, and once in
//the tables).
//This is already an overestimation, as most will delete an
//existing entry orn overwrite one. Still, use a
//conservative safety factor of 2.
    if (!CheckDiskSpace(128 * 2 * 2 *
                                pcoinsTip->GetCacheSize()))
            return state.Error("out of disk space");
// Flush the chainstate (which may refer to
// block index entries).
    if (!pcoinsTip->Flush())
            return AbortNode(state, "Failed to write to coin
database");
        nLastFlush = nNow;
    }
    if ((mode == FLUSH_STATE_ALWAYS ||
         mode == FLUSH_STATE_PERIODIC) &&
         nNow > nLastSetChain +
(int64_t)DATABASE_WRITE_INTERVAL * 1000000) {
 // Update best block in wallet
 // (so we can detect restored wallets).

GetMainSignals().SetBestChain(chainActive.GetLocator
());
        nLastSetChain = nNow;
    }
    } catch (const std::runtime_error& e) {
        return AbortNode(state, std::string("System error
while flushing: ") + e.what());
    }
    return true;
}
```

8 PoW代码实现解读

Blockchain核心算法介绍(3)-Proof of Work-1.

```
void RegisterNodeSignals(CNodeSignals& nodeSignals){
    nodeSignals.GetHeight.connect(&GetHeight);
    nodeSignals.ProcessMessages.connect(&ProcessMessages);
    nodeSignals.SendMessages.connect(&SendMessages);
    nodeSignals.InitializeNode.connect(&InitializeNode);
    nodeSignals.FinalizeNode.connect(&FinalizeNode);
}
```

```
bool ProcessMessages(CNode* pfrom)
{
    const CChainParams& chainparams = Params();
    if (!pfrom->vRecvGetData.empty())
        ProcessGetData(pfrom, chainparams.GetConsensus());
    if (!pfrom->vRecvGetData.empty()) return fOk;
    std::deque<CNetMessage>::iterator it = pfrom->vRecvMsg.begin();
    while (!pfrom->fDisconnect && it != pfrom->vRecvMsg.end()) {
        // get next message
        CNetMessage& msg = *it;
        // end, if an incomplete message is found
        if (!msg.complete())
            break;
        it++;
memcmp(msg.hdr.pchMessageStart, chainparams.MessageStart(),
MESSAGE_START_SIZE)
// Read header
CMessageHeader& hdr = msg.hdr;
hdr.IsValid(chainparams.MessageStart())
string strCommand = hdr.GetCommand();
CDataStream& vRecv = msg.vRecv;
// Process message
bool fRet = false;
try{
  fRet = ProcessMessage(pfrom, strCommand, vRecv, msg.nTime);
```

```
else if (strCommand == NetMsgType::BLOCK && !fImporting && !fReindex) // Ignore blocks received while importing
{
        CBlock block;
        vRecv >> block;
        CInv inv(MSG_BLOCK, block.GetHash());
        LogPrint("net", "received block %s peer=%d\n", inv.hash.ToString(), pfrom->id);
        pfrom->AddInventoryKnown(inv);
        CValidationState state;
        // Process all blocks from whitelisted peers, even if not requested,
        // unless we're still syncing with the network.
        // Such an unrequested block may still be processed, subject to the conditions in AcceptBlock().
        bool forceProcessing = pfrom->fWhitelisted && !IsInitialBlockDownload();
        ProcessNewBlock(state, chainparams, pfrom, &block, forceProcessing, NULL);
        int nDoS;
        if (state.IsInvalid(nDoS)) {
            assert (state.GetRejectCode() < REJECT_INTERNAL); // Blocks are never rejected with internal reject codes
            pfrom->PushMessage(NetMsgType::REJECT, strCommand, (unsigned char)state.GetRejectCode(),
                               state.GetRejectReason().substr(0, MAX_REJECT_MESSAGE_LENGTH), inv.hash);
            if (nDoS > 0) {
                LOCK(cs_main);
                Misbehaving(pfrom->GetId(), nDoS);
            }
        }
    }
```

Blockchain核心算法介绍(3)-Proof of Work-2.

```
bool ProcessNewBlock(CValidationState& state,
                     const CChainParams& chainparams,
                     const CNode* pfrom,
                     const CBlock* pblock,
                     bool fForceProcessing,
                     CDiskBlockPos* dbp)
{
    // Preliminary checks
    bool checked = CheckBlock(*pblock, state);  // 1.
    {
        LOCK(cs_main);
        bool fRequested = MarkBlockAsReceived(pblock->GetHash());
        fRequested |= fForceProcessing;
        if (!checked) {
            return error("%s: CheckBlock FAILED", __func__);
        }
        // Store to disk
        CBlockIndex *pindex = NULL;
        bool ret = AcceptBlock(*pblock, state, chainparams, &pindex, fRequested, dbp); // 2.
        if (pindex && pfrom) {
            mapBlockSource[pindex->GetBlockHash()] = pfrom->GetId();
        }
        CheckBlockIndex(chainparams.GetConsensus());  // 3.
        if (!ret)
            return error("%s: AcceptBlock FAILED", __func__);
    }
    if (!ActivateBestChain(state, chainparams, pblock))   // 4.
        return error("%s: ActivateBestChain failed", __func__);
    return true;
}
```

```
bool CheckBlock(const CBlock& block, CValidationState& state, bool fCheckPOW, bool
fCheckMerkleRoot){
    // These are checks that are independent of context.
    if (block.fChecked)    return true;
    // Check that the header is valid (particularly PoW).  This is mostly
    // redundant with the call in AcceptBlockHeader.
    if (!CheckBlockHeader(block, state, fCheckPOW))
        return false;
    // Check the merkle root.
    if (fCheckMerkleRoot) {
        bool mutated;
        uint256 hashMerkleRoot2 = BlockMerkleRoot(block, &mutated); //详见核心算法介绍（1）
        if (block.hashMerkleRoot != hashMerkleRoot2)
            return state.DoS(100, error("CheckBlock(): hashMerkleRoot mismatch"),
                             REJECT_INVALID, "bad-txnmrklroot", true);
        // Check for merkle tree malleability (CVE-2012-2459): repeating sequences
        // of transactions in a block without affecting the merkle root of a block,
        if (mutated)
            return state.DoS(100, error("CheckBlock(): duplicate transaction"),
                             REJECT_INVALID, "bad-txns-duplicate", true);
    ....
    if (fCheckPOW && fCheckMerkleRoot)
        block.fChecked = true;
    }
```

```
bool CheckBlockHeader(const CBlockHeader& block, CValidationState& state, bool fCheckPOW)
{

    // Check proof of work matches claimed amount
    if (fCheckPOW && !CheckProofOfWork(block.GetHash(), block.nBits, Params().GetConsensus())) // 2.1.1
        return state.DoS(50, error("CheckBlockHeader(): proof of work failed"),
                         REJECT_INVALID, "high-hash");
    // Check timestamp
    if (block.GetBlockTime() > GetAdjustedTime() + 2 * 60 * 60)
        return state.Invalid(error("CheckBlockHeader(): block timestamp too far in the future"),
                             REJECT_INVALID, "time-too-new");
    return true;
}
```

Blockchain核心算法介绍(3)-Proof of Work-3.

```
//https://github.com/bitcoin/bitcoin/blob/master/src/pow.cpp   2.1.1
bool CheckProofOfWork(uint256 hash,
                      unsigned int nBits,
                      const   Consensus::Params& params)
{
    bool fNegative;
    bool fOverflow;
    arith_uint256 bnTarget;
    bnTarget.SetCompact(nBits, &fNegative, &fOverflow);
    // Check range
    if (fNegative || bnTarget == 0 || fOverflow || bnTarget > UintToArith256(params.powLimit))
        return error("CheckProofOfWork(): nBits below minimum work");
    // Check proof of work matches claimed amount
    if (UintToArith256(hash) > bnTarget)
        return error("CheckProofOfWork(): hash doesn't match nBits");
    return true;
}
```

```
/** Store block on disk. If dbp is non-NULL, the file is known to already reside on disk */
static bool AcceptBlock(const CBlock& block, CValidationState& state,
                        const CChainParams& chainparams, CBlockIndex** ppindex,
                        bool fRequested, CDiskBlockPos* dbp){
    AssertLockHeld(cs_main);
    CBlockIndex *&pindex = *ppindex;
    if (!AcceptBlockHeader(block, state, chainparams, &pindex))
        return false;
bool fAlreadyHave = pindex->nStatus & BLOCK_HAVE_DATA;
bool fHasMoreWork = (chainActive.Tip() ? pindex->nChainWork > chainActive.Tip()->nChainWork : true);
bool fTooFarAhead = (pindex->nHeight > int(chainActive.Height() + MIN_BLOCKS_TO_KEEP));
int nHeight = pindex->nHeight;
 // Write block to history file
    try {
        unsigned int nBlockSize = ::GetSerializeSize(block, SER_DISK, CLIENT_VERSION);
        CDiskBlockPos blockPos;
        if (dbp != NULL)
            blockPos = *dbp;
        if (!FindBlockPos(state, blockPos, nBlockSize+8, nHeight, block.GetBlockTime(), dbp != NULL))
            return error("AcceptBlock(): FindBlockPos failed");
        if (dbp == NULL)
            if (!WriteBlockToDisk(block, blockPos, chainparams.MessageStart()))
                AbortNode(state, "Failed to write block");
        if (!ReceivedBlockTransactions(block, state, pindex, blockPos))
            return error("AcceptBlock(): ReceivedBlockTransactions failed");
    } catch (const std::runtime_error& e) {
        return AbortNode(state, std::string("System error: ") + e.what());
    }
    if (fCheckForPruning)
        FlushStateToDisk(state, FLUSH_STATE_NONE); // we just allocated more disk space for block files
return true;
}
```

```
static bool AcceptBlockHeader(const CBlockHeader& block, CValidationState& state, const
CChainParams& chainparams, CBlockIndex** ppindex=NULL)
{
// Check for duplicate
uint256 hash = block.GetHash();
//mapBlockIndex为main中定义的全局变量
BlockMap::iterator miSelf = mapBlockIndex.find(hash);    CBlockIndex *pindex = NULL;
    if (hash != chainparams.GetConsensus().hashGenesisBlock) {
        if (miSelf != mapBlockIndex.end()) {
            pindex = miSelf->second; // Block header is already known.
            if (ppindex)    *ppindex = pindex;
            if (pindex->nStatus & BLOCK_FAILED_MASK)
                return state.Invalid(error("%s: block is marked invalid", __func__), 0, "duplicate");
            return true;
        }
// Get prev block index
 CBlockIndex* pindexPrev = NULL;
 BlockMap::iterator mi = mapBlockIndex.find(block.hashPrevBlock);
 if (mi == mapBlockIndex.end())
            return state.DoS(10, error("%s: prev block not found", __func__), 0, "bad-prevblk");
pindexPrev = (*mi).second;
```

```
    if (pindexPrev->nStatus & BLOCK_FAILED_MASK)
        return state.DoS(100, error("%s: prev block invalid", __func__), REJECT_INVALID, "bad-prevblk");
    assert(pindexPrev);
    if (fCheckpointsEnabled && !CheckIndexAgainstCheckpoint(pindexPrev, state, chainparams, hash))//2.2.1
        return error("%s: CheckIndexAgainstCheckpoint(): %s", __func__, state.GetRejectReason().c_str());
    if (!ContextualCheckBlockHeader(block, state, pindexPrev)) //2.2.2
        return false;
}
```

Blockchain核心算法介绍(3)-Proof of Work-4.

```
static bool CheckIndexAgainstCheckpoint(const CBlockIndex* pindexPrev,   CValidationState& state,
                                        const CChainParams& chainparams, const uint256& hash) //2.2.1
{
    if (*pindexPrev->phashBlock == chainparams.GetConsensus().hashGenesisBlock)
        return true;
    int nHeight = pindexPrev->nHeight+1;
    // Don't accept any forks from the main chain prior to last checkpoint
    CBlockIndex* pcheckpoint = Checkpoints::GetLastCheckpoint(chainparams.Checkpoints());
    if (pcheckpoint && nHeight < pcheckpoint->nHeight)
      return state.DoS(100, error("%s: forked chain older than last checkpoint (height %d)", __func__, nHeight));
    return true;
}
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/pow.cpp 2.2.2.1
unsigned int GetNextWorkRequired(const CBlockIndex* pindexLast, const CBlockHeader *pblock, const
Consensus::Params& params){
    unsigned int nProofOfWorkLimit = UintToArith256(params.powLimit).GetCompact();
    if (pindexLast == NULL)  // Genesis block
        return nProofOfWorkLimit;
    // Only change once per difficulty adjustment interval
    if ((pindexLast->nHeight+1) % params.DifficultyAdjustmentInterval() != 0){
        if (params.fPowAllowMinDifficultyBlocks){ // Special difficulty rule for testnet:
            // If the new block's timestamp is more than 2* 10 minute then allow mining of a min-difficulty block.
            if (pblock->GetBlockTime() > pindexLast->GetBlockTime() + params.nPowTargetSpacing*2)
                return nProofOfWorkLimit;
            else{
                // Return the last non-special-min-difficulty-rules-block
                const CBlockIndex* pindex = pindexLast;
                    while (pindex->pprev && pindex->nHeight % params.DifficultyAdjustmentInterval() != 0
                                         && pindex->nBits == nProofOfWorkLimit)
                        pindex = pindex->pprev;
                return pindex->nBits;
            }
        }
        return pindexLast->nBits;
    }
    // Go back by what we want to be 14 days worth of blocks
    int nHeightFirst = pindexLast->nHeight - (params.DifficultyAdjustmentInterval()-1);
    const CBlockIndex* pindexFirst = pindexLast->GetAncestor(nHeightFirst);
    return CalculateNextWorkRequired(pindexLast, pindexFirst->GetBlockTime(), params);
}
```

```
//2.2.2
bool ContextualCheckBlockHeader(const CBlockHeader& block,
                                CValidationState& state,   CBlockIndex * const
pindexPrev)
{
    const Consensus::Params& consensusParams = Params().GetConsensus();

    // Check proof of work
 if (block.nBits != GetNextWorkRequired(pindexPrev, &block, consensusParams))//2.2.2.1
        return state.DoS(100, error("%s: incorrect proof of work", __func__),
                         REJECT_INVALID, "bad-diffbits");
    // Check timestamp against prev
    if (block.GetBlockTime() <= pindexPrev->GetMedianTimePast())
        return state.Invalid(error("%s: block's timestamp is too early", __func__),
                             REJECT_INVALID, "time-too-old");
    // Reject block.nVersion=1 blocks when 95% (75% on testnet)
    // of the network has upgraded:
    if (block.nVersion < 2 &&
        IsSuperMajority(2, pindexPrev,
            consensusParams.nMajorityRejectBlockOutdated,  consensusParams))//2.2.2.2
        return state.Invalid(error("%s: rejected nVersion=1 block", __func__),
                             REJECT_OBSOLETE, "bad-version");
    // Reject block.nVersion=2 blocks when 95% (75% on testnet)
    // of the network has upgraded:
    if (block.nVersion < 3 &&
        IsSuperMajority(3, pindexPrev,
consensusParams.nMajorityRejectBlockOutdated, consensusParams))//2.2.2.2
        return state.Invalid(error("%s: rejected nVersion=2 block", __func__),
                             REJECT_OBSOLETE, "bad-version");
    // Reject block.nVersion=3 blocks when 95% (75% on testnet)
    // of the network has upgraded:
    if (block.nVersion < 4 && IsSuperMajority(4, pindexPrev,
consensusParams.nMajorityRejectBlockOutdated, consensusParams))//2.2.2.2
        return state.Invalid(error("%s : rejected nVersion=3 block", __func__),
                             REJECT_OBSOLETE, "bad-version");
    return true;
}
```

Blockchain核心算法介绍(3)-Proof of Work-5.

```
//2.2.2.2
static bool IsSuperMajority(int minVersion, const CBlockIndex* pstart,
                            unsigned  nRequired, const Consensus::Params& consensusParams)
{
    unsigned int nFound = 0;
    for (int i = 0; i < consensusParams.nMajorityWindow && nFound < nRequired && pstart != NULL; i++)
    {
        if (pstart->nVersion >= minVersion)
            ++nFound;
        pstart = pstart->pprev;
    }
    return (nFound >= nRequired);
}
```

```
//2.3
void static CheckBlockIndex(const Consensus::Params& consensusParams)
{
    // During a reindex, we read the genesis block and call CheckBlockIndex before ActivateBestChain,
    // so we have the genesis block in mapBlockIndex but no active chain.  (A few of the tests when
    // iterating the block tree require that chainActive has been initialized.)
    if (chainActive.Height() < 0) {
        assert(mapBlockIndex.size() <= 1);
        return;
    }
    // Build forward-pointing map of the entire block tree.
    std::multimap<CBlockIndex*,CBlockIndex*> forward;
    for (BlockMap::iterator it = mapBlockIndex.begin(); it != mapBlockIndex.end(); it++) {
        forward.insert(std::make_pair(it->second->pprev, it->second));
    }
    std::pair<std::multimap<CBlockIndex*,CBlockIndex*>::iterator,std::multimap<CBlockIndex*,
                                CBlockIndex*>::iterator> rangeGenesis = forward.equal_range(NULL);
    CBlockIndex *pindex = rangeGenesis.first->second;
    rangeGenesis.first++;
    assert(rangeGenesis.first == rangeGenesis.second); // There is only one index entry with parent NULL.
    // Iterate over the entire block tree, using depth-first search.
    // Along the way, remember whether there are blocks on the path from genesis
    // block being explored which are the first to have certain properties.
    size_t nNodes = 0;    int nHeight = 0;
    while (pindex != NULL) {
        nNodes++;
```

```
if (pindexFirstInvalid == NULL && pindex->nStatus & BLOCK_FAILED_VALID)
pindexFirstInvalid = pindex;
 if (pindexFirstMissing == NULL && !(pindex->nStatus & BLOCK_HAVE_DATA))
pindexFirstMissing = pindex;
if (pindexFirstNeverProcessed == NULL && pindex->nTx == 0)
   pindexFirstNeverProcessed = pindex;
if (pindex->pprev != NULL && pindexFirstNotTreeValid == NULL && (pindex->nStatus &
BLOCK_VALID_MASK) < BLOCK_VALID_TREE)
    pindexFirstNotTreeValid = pindex;
 if (pindex->pprev != NULL && pindexFirstNotTransactionsValid == NULL && (pindex-
>nStatus & BLOCK_VALID_MASK) < BLOCK_VALID_TRANSACTIONS)
pindexFirstNotTransactionsValid = pindex;
 if (pindex->pprev != NULL && pindexFirstNotChainValid == NULL && (pindex->nStatus &
BLOCK_VALID_MASK) < BLOCK_VALID_CHAIN)
     pindexFirstNotChainValid = pindex;
 if (pindex->pprev != NULL && pindexFirstNotScriptsValid == NULL && (pindex->nStatus &
BLOCK_VALID_MASK) < BLOCK_VALID_SCRIPTS)
     pindexFirstNotScriptsValid = pindex;
 // Begin: actual consistency checks.
 if (pindex->pprev == NULL) {
    // Genesis block checks.
    // Genesis block's hash must match.
    assert(pindex->GetBlockHash() == consensusParams.hashGenesisBlock);
    // The current active chain's genesis block must be this block.
    assert(pindex == chainActive.Genesis());
 }
// nSequenceId can't be set for blocks that aren't linked
if (pindex->nChainTx == 0) assert(pindex->nSequenceId == 0);
if (!fHavePruned) {
            // If we've never pruned, then HAVE_DATA should be equivalent to nTx > 0
            assert(!(pindex->nStatus & BLOCK_HAVE_DATA) == (pindex->nTx == 0));
            assert(pindexFirstMissing == pindexFirstNeverProcessed);
} else {
            // If we have pruned, then we can only say that HAVE_DATA implies nTx > 0
            if (pindex->nStatus & BLOCK_HAVE_DATA) assert(pindex->nTx > 0);
 }
if (pindex->nStatus & BLOCK_HAVE_UNDO)
     assert(pindex->nStatus &BLOCK_HAVE_DATA);
     assert(((pindex->nStatus & BLOCK_VALID_MASK) >=
BLOCK_VALID_TRANSACTIONS) == (pindex->nTx > 0)); // This is pruning-independent.
```

Blockchain核心算法介绍(3)-Proof of Work-6.

```
// All parents having had data (at some point) is equivalent to all parents being
//VALID_TRANSACTIONS, which is equivalent to nChainTx being set.
// nChainTx != 0 is used to signal that all parent blocks have been processed
//(but may have been pruned).
assert((pindexFirstNeverProcessed != NULL) == (pindex->nChainTx == 0));
assert((pindexFirstNotTransactionsValid != NULL) == (pindex->nChainTx == 0));
assert(pindex->nHeight == nHeight); // nHeight must be consistent.
 // For every block except the genesis block, the chainwork must be larger than the parent's.
assert(pindex->pprev == NULL || pindex->nChainWork >= pindex->pprev->nChainWork);
// The pskip pointer must point back for all but the first 2 blocks.
assert(nHeight < 2 || (pindex->pskip && (pindex->pskip->nHeight < nHeight)));
 // All mapBlockIndex entries must at least be TREE valid
assert(pindexFirstNotTreeValid == NULL);
if ((pindex->nStatus & BLOCK_VALID_MASK) >= BLOCK_VALID_TREE)
assert(pindexFirstNotTreeValid == NULL); // TREE valid implies all parents are TREE valid
 if ((pindex->nStatus & BLOCK_VALID_MASK) >= BLOCK_VALID_CHAIN)
assert(pindexFirstNotChainValid == NULL); // CHAIN valid implies all parents are CHAIN valid
if ((pindex->nStatus & BLOCK_VALID_MASK) >= BLOCK_VALID_SCRIPTS)
 // SCRIPTS valid implies all parents are SCRIPTS valid
assert(pindexFirstNotScriptsValid == NULL);
if (pindexFirstInvalid == NULL) {
       // Checks for not-invalid blocks.
      // The failed mask cannot be set for blocks without invalid parents.
        assert((pindex->nStatus & BLOCK_FAILED_MASK) == 0);
}
if (!CBlockIndexWorkComparator()(pindex, chainActive.Tip())
                                      && pindexFirstNeverProcessed == NULL) {
    if (pindexFirstInvalid == NULL) {
        // If this block sorts at least as good as the current tip and
        // is valid and we have all data for its parents, it must be in
       // setBlockIndexCandidates.  chainActive.Tip() must also be there
       // even if some data has been pruned.
          if (pindexFirstMissing == NULL || pindex == chainActive.Tip()) {
                assert(setBlockIndexCandidates.count(pindex));
          }
      // If some parent is missing, then it could be that this block was in
      // setBlockIndexCandidates but had to be removed because of the missing data.
      // In this case it must be in mapBlocksUnlinked -- see test below.
     }
 }
```

```
else {
// If this block sorts worse than the current tip or some ancestor's block has never been seen,
// it cannot be in setBlockIndexCandidates.
       assert(setBlockIndexCandidates.count(pindex) == 0);
 }
// Check whether this block is in mapBlocksUnlinked.
std::pair<std::multimap<CBlockIndex*,CBlockIndex*>::iterator,std
       ::multimap<CBlockIndex*,CBlockIndex*>::iterator> rangeUnlinked =
mapBlocksUnlinked.equal_range(pindex->pprev);
        bool foundInUnlinked = false;
        while (rangeUnlinked.first != rangeUnlinked.second) {
            assert(rangeUnlinked.first->first == pindex->pprev);
            if (rangeUnlinked.first->second == pindex) {
                foundInUnlinked = true;
                break;
            }   rangeUnlinked.first++;
        }
if (pindex->pprev && (pindex->nStatus & BLOCK_HAVE_DATA) &&
pindexFirstNeverProcessed != NULL && pindexFirstInvalid == NULL) {
      // If this block has block data available, some parent was never received,
      // and has no invalid parents, it must be in mapBlocksUnlinked.
    assert(foundInUnlinked);
 }
// Can't be in mapBlocksUnlinked if we don't HAVE_DATA
if (!(pindex->nStatus & BLOCK_HAVE_DATA)) assert(!foundInUnlinked);
// We aren't missing data for any parent -- cannot be in mapBlocksUnlinked.
 if (pindexFirstMissing == NULL)    assert(!foundInUnlinked);
if (pindex->pprev && (pindex->nStatus & BLOCK_HAVE_DATA) &&
pindexFirstNeverProcessed == NULL && pindexFirstMissing != NULL) {
           // We HAVE_DATA for this block, have received data for all parents at some point,
           //  but we're currently missing data for some parent.
      assert(fHavePruned); // We must have pruned.
// So if this block is itself better than chainActive.Tip() and it wasn't in
// setBlockIndexCandidates, then it must be in mapBlocksUnlinked.
 if (!CBlockIndexWorkComparator()(pindex, chainActive.Tip()) &&
setBlockIndexCandidates.count(pindex) == 0) {
              if (pindexFirstInvalid == NULL) {   assert(foundInUnlinked); }
           }
}
```

Blockchain核心算法介绍(3)-Proof of Work-7.

```
// assert(pindex->GetBlockHash() == pindex->GetBlockHeader().GetHash());
// Perhaps too slow End: actual consistency checks.
// Try descending into the first subnode.
std::pair<std::multimap<CBlockIndex*,CBlockIndex*>::iterator,std::multimap<CBlockIndex*,CBlockInde
x*>::iterator> range = forward.equal_range(pindex);
if (range.first != range.second) {
        // A subnode was found.
        pindex = range.first->second;
        nHeight++;
        continue;
}// This is a leaf node. Move upwards until reach a node of which we have not yet visited the last child.
while (pindex) { // We are going to either move to a parent or a sibling of pindex.
        // If pindex was the first with a certain property, unset the corresponding variable.
        if (pindex == pindexFirstInvalid) pindexFirstInvalid = NULL;
        if (pindex == pindexFirstMissing) pindexFirstMissing = NULL;
        if (pindex == pindexFirstNeverProcessed) pindexFirstNeverProcessed = NULL;
        if (pindex == pindexFirstNotTreeValid) pindexFirstNotTreeValid = NULL;
        if (pindex == pindexFirstNotTransactionsValid) pindexFirstNotTransactionsValid = NULL;
        if (pindex == pindexFirstNotChainValid) pindexFirstNotChainValid = NULL;
        if (pindex == pindexFirstNotScriptsValid) pindexFirstNotScriptsValid = NULL;
        CBlockIndex* pindexPar = pindex->pprev;  // Find our parent.
        // Find which child we just visited.
std::pair<std::multimap<CBlockIndex*,CBlockIndex*>::iterator,std::multimap<CBlockIndex*,CBlockInde
x*>::iterator> rangePar = forward.equal_range(pindexPar);
        while (rangePar.first->second != pindex) {
            // Our parent must have at least the node we're coming from as child.
             assert(rangePar.first != rangePar.second);
             rangePar.first++;
        }
 // Proceed to the next one.
        rangePar.first++;
        if (rangePar.first != rangePar.second) {
            pindex = rangePar.first->second;   // Move to the sibling.
            break;
        } else { // Move up further.
            pindex = pindexPar;
            nHeight--;
            continue;
        }}}  // Check that we actually traversed the entire map.
    assert(nNodes == forward.size());
```

```
//2.4
/**
 * Make the best chain active, in multiple steps. The result is either failure
 * or an activated best chain. pblock is either NULL or a pointer to a block
 * that is already loaded (to avoid loading it again from disk).
 */
bool ActivateBestChain(CValidationState &state, const CChainParams& chainparams,
                                const CBlock *pblock)
{    CBlockIndex *pindexMostWork = NULL;
     do {
       boost::this_thread::interruption_point();
       CBlockIndex *pindexNewTip = NULL;
       const CBlockIndex *pindexFork;
       bool fInitialDownload;
       CBlockIndex *pindexOldTip = chainActive.Tip();
       pindexMostWork = FindMostWorkChain();
       // Whether we have anything to do at all.
       if (pindexMostWork == NULL || pindexMostWork == chainActive.Tip())
             return true;
       if (!ActivateBestChainStep(state, chainparams, pindexMostWork, pblock &&
                    pblock->GetHash() == pindexMostWork->GetBlockHash() ? pblock : NULL))
             return false;
       pindexNewTip = chainActive.Tip();
       pindexFork = chainActive.FindFork(pindexOldTip);
       fInitialDownload = IsInitialBlockDownload();
       if (pindexFork != pindexNewTip) {
          uiInterface.NotifyBlockTip(fInitialDownload, pindexNewTip);
          if (!fInitialDownload) {
             // Find the hashes of all blocks that weren't previously in the best chain.
             std::vector<uint256> vHashes;
             CBlockIndex *pindexToAnnounce = pindexNewTip;
             while (pindexToAnnounce != pindexFork) {
                vHashes.push_back(pindexToAnnounce->GetBlockHash());
                pindexToAnnounce = pindexToAnnounce->pprev;
                if (vHashes.size() == MAX_BLOCKS_TO_ANNOUNCE) {
                   // Limit announcements in case of a huge reorganization.
                   // Rely on the peer's synchronization mechanism in that case.
                   break;
                }
             }
```

Blockchain核心算法介绍(3)-Proof of Work-8.

```
// Relay inventory, but don't relay old inventory during initial block download.
            int nBlockEstimate = 0;
            if (fCheckpointsEnabled)
                nBlockEstimate = Checkpoints::GetTotalBlocksEstimate(chainparams.Checkpoints());
            {
                LOCK(cs_vNodes);
                BOOST_FOREACH(CNode* pnode, vNodes) {
if (chainActive.Height() > (pnode->nStartingHeight != -1 ? pnode->nStartingHeight - 2000 :  nBlockEstimate))
{
                    BOOST_REVERSE_FOREACH(const uint256& hash, vHashes) {
                        pnode->PushBlockHash(hash);
                    }
                }}}
// Notify external listeners about the new tip.
            if (!vHashes.empty()) {
                GetMainSignals().UpdatedBlockTip(pindexNewTip);
            }}}
    } while(pindexMostWork != chainActive.Tip());
    CheckBlockIndex(chainparams.GetConsensus());
    // Write changes periodically to disk, after relay.
    if (!FlushStateToDisk(state, FLUSH_STATE_PERIODIC)) {
        return false;
    }
    return true;
}
```

```
struct CMainSignals {
    /** Notifies listeners of updated block chain tip */
    boost::signals2::signal<void (const CBlockIndex *)> UpdatedBlockTip;
    /** Notifies listeners of updated transaction data (transaction, and optionally the block it is found in. */
    boost::signals2::signal<void (const CTransaction &, const CBlock *)> SyncTransaction;
    /** Notifies listeners of an updated transaction without new data (for now: a coinbase potentially becoming
visible). */
    boost::signals2::signal<void (const uint256 &)> UpdatedTransaction;
    /** Notifies listeners of a new active block chain. */
    boost::signals2::signal<void (const CBlockLocator &)> SetBestChain;
    /** Notifies listeners about an inventory item being seen on the network. */
    boost::signals2::signal<void (const uint256 &)> Inventory;
    /** Tells listeners to broadcast their data. */
    boost::signals2::signal<void (int64_t nBestBlockTime)> Broadcast;
    /** Notifies listeners of a block validation result */
    boost::signals2::signal<void (const CBlock&, const CValidationState&)> BlockChecked;
    /** Notifies listeners that a key for mining is required (coinbase) */
    boost::signals2::signal<void (boost::shared_ptr<CReserveScript>&)> ScriptForMining;
    /** Notifies listeners that a block has been successfully mined */
    boost::signals2::signal<void (const uint256 &)> BlockFound;
};
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/validationinterface.cpp
static CMainSignals g_signals;
CMainSignals& GetMainSignals(){  return g_signals;  }

//https://github.com/bitcoin/bitcoin/blob/master/src/zmq/zmqnotificationinterface.cpp
void CZMQNotificationInterface::UpdatedBlockTip(const CBlockIndex *pindex)
{
    for (std::list<CZMQAbstractNotifier*>::iterator i = notifiers.begin(); i!=notifiers.end(); )
    {
        CZMQAbstractNotifier *notifier = *i;
        if (notifier->NotifyBlock(pindex)){    i++;   }
        else{
            notifier->Shutdown();
            i = notifiers.erase(i);
        }
    }
}
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/checkpoints.cpp
int GetTotalBlocksEstimate(const CCheckpointData& data){
        const MapCheckpoints& checkpoints = data.mapCheckpoints;
        if (checkpoints.empty())    return 0;
        return checkpoints.rbegin()->first;
 }
```

```
//https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.h
typedef std::map<int, uint256> MapCheckpoints;
struct CCheckpointData {
    MapCheckpoints mapCheckpoints;
    int64_t nTimeLastCheckpoint;  int64_t nTransactionsLastCheckpoint;
    double fTransactionsPerDay;
};
```

9 区块链协议Script实现

区块链交易指令Script

- Bitcoin 使用脚本系统来交易 。 其脚本语言是一个基于堆栈简单形式。其从左至右执行，其目标不是必需要变成图灵完备，但应不存在死循环。
- 一个脚本实际上就是一列交易指令的记录，其他使用比特币系统的人能够访问到它。对于一个典型的比特币地址转账，需要保证支付者提供一个公钥封装在脚本里，一个签名算出私钥对公钥的回复。
- 脚本提供了一个一个灵活的方式来修改比特币支付转移的参数项。
- 一次交易的有效性体现在没有错误触发。栈顶项非空。

区块链交易Script底层实现

Stack

Word	Opcode	Hex	Input	Output	Description
OP_TOALTSTACK	107	0x6b	x1	(alt)x1	Puts the input onto the top of the alt stack. Removes it from the main stack.
OP_FROMALTSTACK	108	0x6c	(alt)x1	x1	Puts the input onto the top of the main stack. Removes it from the alt stack.
OP_IFDUP	115	0x73	x	x / x x	If the top stack value is not 0, duplicate it.
OP_DEPTH	116	0x74	Nothing	<Stack size>	Puts the number of stack items onto the stack.
OP_DROP	117	0x75	x	Nothing	Removes the top stack item.
OP_DUP	118	0x76	x	x x	Duplicates the top stack item.
OP_NIP	119	0x77	x1 x2	x2	Removes the second-to-top stack item.
OP_OVER	120	0x78	x1 x2	x1 x2 x1	Copies the second-to-top stack item to the top.
OP_PICK	121	0x79	xn ... x2 x1 x0 <n>	xn ... x2 x1 x0 xn	The item *n* back in the stack is copied to the top.
OP_ROLL	122	0x7a	xn ... x2 x1 x0 <n>	... x2 x1 x0 xn	The item *n* back in the stack is moved to the top.
OP_ROT	123	0x7b	x1 x2 x3	x2 x3 x1	The top three items on the stack are rotated to the left.
OP_SWAP	124	0x7c	x1 x2	x2 x1	The top two items on the stack are swapped.
OP_TUCK	125	0x7d	x1 x2	x2 x1 x2	The item at the top of the stack is copied and inserted before the second-to-top item.
OP_2DROP	109	0x6d	x1 x2	Nothing	Removes the top two stack items.
OP_2DUP	110	0x6e	x1 x2	x1 x2 x1 x2	Duplicates the top two stack items.
OP_3DUP	111	0x6f	x1 x2 x3	x1 x2 x3 x1 x2 x3	Duplicates the top three stack items.
OP_2OVER	112	0x70	x1 x2 x3 x4	x1 x2 x3 x4 x1 x2	Copies the pair of items two spaces back in the stack to the front.
OP_2ROT	113	0x71	x1 x2 x3 x4 x5 x6	x3 x4 x5 x6 x1 x2	The fifth and sixth items back are moved to the top of the stack.
OP_2SWAP	114	0x72	x1 x2 x3 x4	x3 x4 x1 x2	Swaps the top two pairs of items.

//https://github.com/bitcoin/bitcoin/blob/master/src/script/script.h

```
enum opcodetype
{  /** Script opcodes */
    // stack ops
    OP_TOALTSTACK = 0x6b,
    OP_FROMALTSTACK = 0x6c,
    OP_2DROP = 0x6d,
    OP_2DUP = 0x6e,
    OP_3DUP = 0x6f,
    OP_2OVER = 0x70,
    OP_2ROT = 0x71,
    OP_2SWAP = 0x72,
    OP_IFDUP = 0x73,
    OP_DEPTH = 0x74,
    OP_DROP = 0x75,
    OP_DUP = 0x76,
    OP_NIP = 0x77,
    OP_OVER = 0x78,
    OP_PICK = 0x79,
    OP_ROLL = 0x7a,
    OP_ROT = 0x7b,
    OP_SWAP = 0x7c,
    OP_TUCK = 0x7d,
```

```
typedef prevector<28, unsigned char>
CScriptBase;
//prevetor见
https://github.com/bitcoin/bitcoin/blob/mas
ter/src/prevector.h

/** Serialized script, used inside
transaction inputs and outputs */
class CScript : public CScriptBase
{
    CScript& push_int64(int64_t n)
    {
        if (n == -1 || (n >= 1 && n <= 16))
            push_back(n + (OP_1 - 1));
        else if (n == 0)
            push_back(OP_0);
        else
            *this << CScriptNum::serialize(n);
        }
        return *this;
    }
```

Tranascation交易指令解释

general format of a Bitcoin transaction (inside a block)

Field	Description	Size
Version no	currently 1	4 bytes
In-counter	positive integer VI = VarInt	1 - 9 bytes
list of inputs	the first input of the first transaction is also called "coinbase" (its content was ignored in earlier versions)	<in-counter>-many inputs
Out-counter	positive integer VI = VarInt	1 - 9 bytes
list of outputs	the outputs of the first transaction spend the mined bitcoins for the block	<out-counter>-many outputs
lock_time	if non-zero and sequence numbers are < 0xFFFFFFFF: block height or timestamp when transaction is final	4 bytes

An input is a reference to an output from a previous transaction. Multiple inputs are often listed in a transaction. All of the new transaction's input values are added up

```
class CTxIn
{
public:
    COutPoint prevout;
    CScript scriptSig;
    uint32_t nSequence;
template <typename Stream, typename Operation>
    inline void SerializationOp(Stream& s, Operation ser_action,
int nType, int nVersion) {
        READWRITE(prevout);
        READWRITE(*(CScriptBase*)(&scriptSig));
        READWRITE(nSequence);
    }
```

```
/** The basic transaction that is broadcasted on the network and contained in
 * blocks.  A transaction can contain multiple inputs and outputs.*/
class CTransaction{
private: /** Memory only. */
    const uint256 hash;
    void UpdateHash() const;
public:
    static const int32_t CURRENT_VERSION=1;
    const int32_t nVersion;
    const std::vector<CTxIn> vin;
    const std::vector<CTxOut> vout;
    const uint32_t nLockTime;
template <typename Stream, typename Operation>
    inline void SerializationOp(Stream& s, Operation ser_action, int nType, int nVersion) {
        READWRITE(*const_cast<int32_t*>(&this->nVersion));
        nVersion = this->nVersion;
        READWRITE(*const_cast<std::vector<CTxIn>*>(&vin));
        READWRITE(*const_cast<std::vector<CTxOut>*>(&vout));
        READWRITE(*const_cast<uint32_t*>(&nLockTime));
        if (ser_action.ForRead())
            UpdateHash();
    }
```

An output contains instructions for sending bitcoins. Value is the number of Satoshi (1 BTC = 100,000,000 Satoshi) that this output will be worth when claimed. ScriptPubKey is the second half of a script (discussed later). There can be more than one output, and they share the combined value of the inputs

```
class CTxOut
{
public:
    CAmount nValue;
    CScript scriptPubKey;
template <typename Stream, typename Operation>
    inline void SerializationOp(Stream& s, Operation ser_action,
int nType, int nVersion) {
        READWRITE(nValue);
        READWRITE(*(CScriptBase*)(&scriptPubKey));
    }
```

Script指令底层调用实现(1)

```
//https://github.com/bitcoin/bitcoin/blob/master/src/script/interpreter.cpp
//执行指令
bool EvalScript(vector<vector<unsigned char> >& stack,
                const CScript& script, unsigned int flags,
                const BaseSignatureChecker& checker, ScriptError* serror)
{
    static const CScriptNum bnZero(0);
    static const CScriptNum bnOne(1);
    static const CScriptNum bnFalse(0);
    static const CScriptNum bnTrue(1);
    static const valtype vchFalse(0);
    static const valtype vchZero(0);
    static const valtype vchTrue(1, 1);
    CScript::const_iterator pc = script.begin();
    CScript::const_iterator pend = script.end();
    CScript::const_iterator pbegincodehash = script.begin();
    opcodetype opcode;
    valtype vchPushValue;
    vector<bool> vfExec;
    vector<valtype> altstack;
try {
        while (pc < pend)
        {
            bool fExec = !count(vfExec.begin(), vfExec.end(), false);
            // Read instruction
            if (!script.GetOp(pc, opcode, vchPushValue))
                return set_error(serror, SCRIPT_ERR_BAD_OPCODE);
            if (vchPushValue.size() > MAX_SCRIPT_ELEMENT_SIZE)
                return set_error(serror, SCRIPT_ERR_PUSH_SIZE);
            // Note how OP_RESERVED does not count towards the opcode limit.
            if (opcode > OP_16 && ++nOpCount > MAX_OPS_PER_SCRIPT)
                return set_error(serror, SCRIPT_ERR_OP_COUNT);
```

```
if (opcode == OP_CAT ||
                opcode == OP_SUBSTR ||
                opcode == OP_LEFT ||
                opcode == OP_RIGHT ||
                opcode == OP_INVERT ||
                opcode == OP_AND ||
                opcode == OP_OR ||
                opcode == OP_XOR ||
                opcode == OP_2MUL ||
                opcode == OP_2DIV ||
                opcode == OP_MUL ||
                opcode == OP_DIV ||
                opcode == OP_MOD ||
                opcode == OP_LSHIFT ||
                opcode == OP_RSHIFT)
return set_error(serror, SCRIPT_ERR_DISABLED_OPCODE); // Disabled opcodes
if (fExec && 0 <= opcode && opcode <= OP_PUSHDATA4) {
                if (fRequireMinimal && !CheckMinimalPush(vchPushValue, opcode)) {
                    return set_error(serror, SCRIPT_ERR_MINIMALDATA);
                }stack.push_back(vchPushValue);
                } else if (fExec || (OP_IF <= opcode && opcode <= OP_ENDIF))
            switch (opcode)
            {// Push value
                case OP_1NEGATE:
                case OP_1:
                case OP_2:
                case OP_3:
                case OP_4:
                case OP_5:
                case OP_6:
                case OP_7:
                case OP_8:
                case OP_9:
```

Script指令底层调用实现(2)

```
case OP_10:  case OP_11:  case OP_12:
case OP_13:  case OP_14:  case OP_15:
case OP_16:{
                CScriptNum bn((int)opcode - (int)(OP_1 - 1));
                stack.push_back(bn.getvch());
// The result of these opcodes should always be the minimal way to push the
data
}break;
// Stack ops
            case OP_TOALTSTACK:{
                if (stack.size() < 1)
                    return set_error(serror,
SCRIPT_ERR_INVALID_STACK_OPERATION);
                altstack.push_back(stacktop(-1));
                popstack(stack);
            }break;
            case OP_FROMALTSTACK:{
                if (altstack.size() < 1)
                    return set_error(serror,
SCRIPT_ERR_INVALID_ALTSTACK_OPERATION);
                stack.push_back(altstacktop(-1));
                popstack(altstack);
            }
            break;
            case OP_2DROP:{
                // (x1 x2 -- )
                if (stack.size() < 2)
                    return set_error(serror,
SCRIPT_ERR_INVALID_STACK_OPERATION);
                popstack(stack);
                popstack(stack);
            }break;
```

```
case OP_2DUP:{
                // (x1 x2 -- x1 x2 x1 x2)
                if (stack.size() < 2)
                    return set_error(serror,
SCRIPT_ERR_INVALID_STACK_OPERATION);
                valtype vch1 = stacktop(-2);
                valtype vch2 = stacktop(-1);
                stack.push_back(vch1);
                stack.push_back(vch2);
            }break;
            case OP_3DUP:{
                // (x1 x2 x3 -- x1 x2 x3 x1 x2 x3)
                if (stack.size() < 3)
                    return set_error(serror,
SCRIPT_ERR_INVALID_STACK_OPERATION);
                valtype vch1 = stacktop(-3);
                valtype vch2 = stacktop(-2);
                valtype vch3 = stacktop(-1);
                stack.push_back(vch1);
                stack.push_back(vch2);
                stack.push_back(vch3);
            }break;
case OP_2OVER:{
                // (x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2)
                if (stack.size() < 4)
                    return set_error(serror,
SCRIPT_ERR_INVALID_STACK_OPERATION);
                valtype vch1 = stacktop(-4);
                valtype vch2 = stacktop(-3);
                stack.push_back(vch1);
                stack.push_back(vch2);
            }break;
  ........
```

Script指令底层调用实现(3)

```
// Crypto
case OP_RIPEMD160:
case OP_SHA1:
case OP_SHA256:
case OP_HASH160:
case OP_HASH256: {
     // (in -- hash)
    if (stack.size() < 1)
        return set_error(serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
valtype& vch = stacktop(-1);
valtype vchHash((opcode == OP_RIPEMD160 || opcode == OP_SHA1 || opcode ==
OP_HASH160) ? 20 : 32);
if (opcode == OP_RIPEMD160)
  CRIPEMD160().Write(begin_ptr(vch),vch.size()).Finalize(
                                                        begin_ptr(vchHash));
else if (opcode == OP_SHA1)
  CSHA1().Write(begin_ptr(vch), vch.size()).Finalize(begin_ptr(vchHash));
else if (opcode == OP_SHA256)
 CSHA256().Write(begin_ptr(vch), vch.size()).Finalize(begin_ptr(vchHash));
else if (opcode == OP_HASH160)
                CHash160().Write(begin_ptr(vch),vch.size()).Finalize(
                                                        begin_ptr(vchHash));
else if (opcode == OP_HASH256)
                CHash256().Write(begin_ptr(vch),vch.size()).Finalize(
                                                        begin_ptr(vchHash));
popstack(stack);
stack.push_back(vchHash);
}break;
```

10 secp256k1（ECDSA椭圆曲线加密算法）实现

2. secp256k1

比特币地址和私钥是由ECDSA椭圆曲线加密算法计算出来的，

由ECDSA私钥计算出我们常用的Bitcoin-qt格式比特币地址需要有十个步骤：

第一步，产生ECDSA私钥，如：18E14A7B6A307F426A94F8114701E7C8E774E7F9A47E2C2035DB29A206321725

第二步，计算出ECDSA公钥，0450863AD64A87AE8A2FE8….82BA6

第三步，对公钥进行SHA256运算600FFE422B4E00731A59557A5CCA46CC183944191006324A447BDB2D98D4B408

第四步，对第三步结果进行RIPEMD-160运算010966776006953D5567439E5E39F86A0D273BEE

第五步，在第四步结果上加上版本号00010966776006953D5567439E5E39F86A0D273BEE

第六步，对第五步结果进行SHA256运算445C7A8007A93D8733188288BB320A8FE2DEBD2AE1B47F0F50BC10BAE845C094

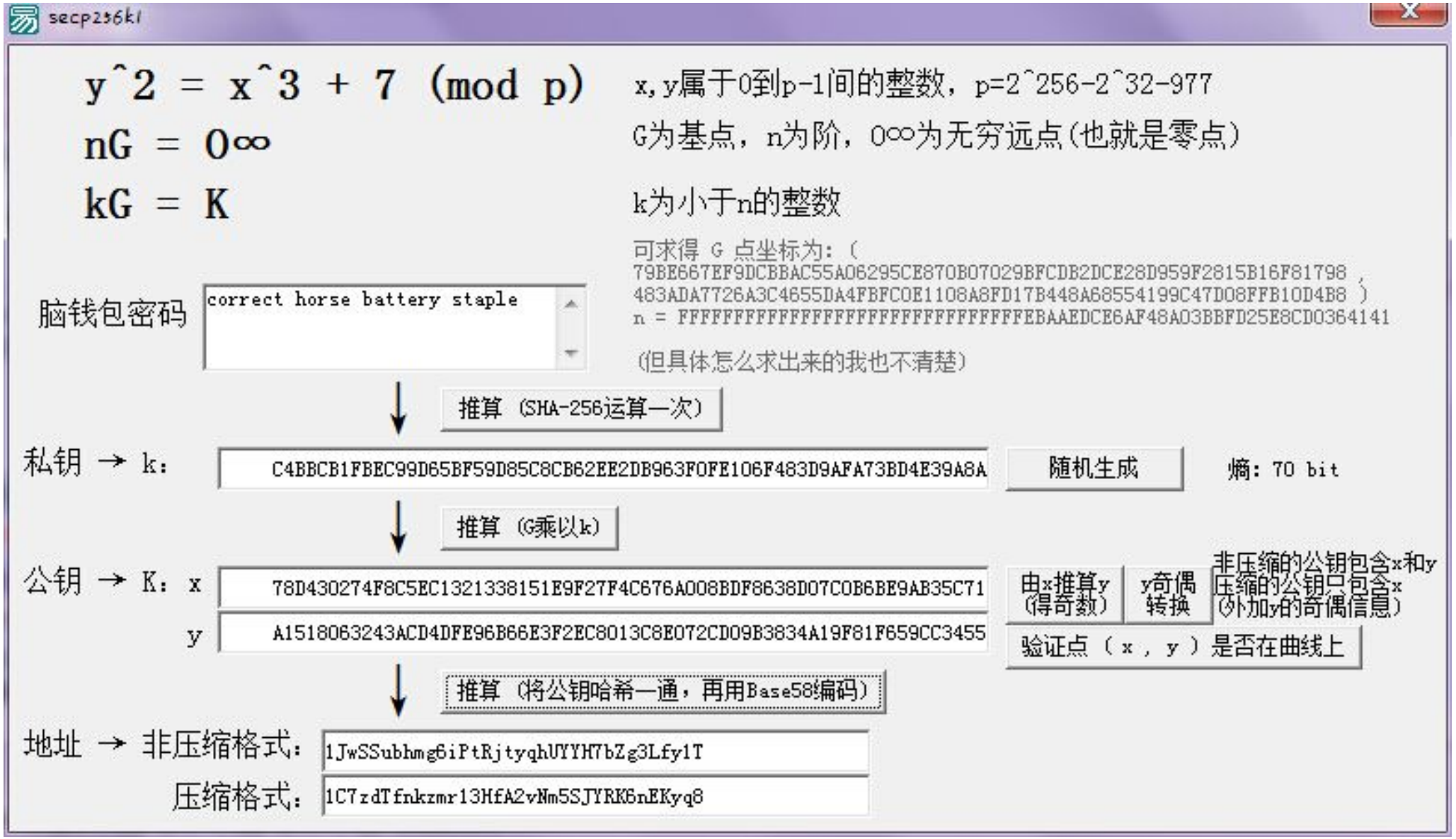
第七步，对第六步结果进行SHA256运算D61967F63C7DD183914A4AE452C9F6AD5D462CE3D277798075B107615C1A8A30

第八步，提出第七步结果的前四个字节D61967F6

第九步，将第八步的结果加到第五步结果最后面00010966776006953D5567439E5E39F86A0D273BEED61967F6

第十步，对第九步结果进行Base58编码16UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM

1. 比特币生成secp256k1(ECC四尝)



secp256k1 pubkey源码实现(1)

```
int ec_privkey_export_der(const secp256k1_context *ctx, unsigned char *privkey, size_t *privkeylen, const unsigned char *key32, int
compressed) {
    secp256k1_pubkey pubkey;
    size_t pubkeylen = 0;
    if (!secp256k1_ec_pubkey_create(ctx, &pubkey, key32)) {
        *privkeylen = 0;   return 0;
    }
    if (compressed) {
        static const unsigned char begin[] = {
            0x30,0x81,0xD3,0x02,0x01,0x01,0x04,0x20
        };
        static const unsigned char middle[] = {
            0xA0,0x81,0x85,0x30,0x81,0x82,0x02,0x01,0x01,0x30,0x2C,0x06,0x07,0x2A,0x86,0x48,
            0xCE,0x3D,0x01,0x01,0x02,0x21,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
            0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
            0xFF,0xFF,0xFE,0xFF,0xFF,0xFC,0x2F,0x30,0x06,0x04,0x01,0x00,0x04,0x01,0x07,0x04,
            0x21,0x02,0x79,0xBE,0x66,0x7E,0xF9,0xDC,0xBB,0xAC,0x55,0xA0,0x62,0x95,0xCE,0x87,
            0x0B,0x07,0x02,0x9B,0xFC,0xDB,0x2D,0xCE,0x28,0xD9,0x59,0xF2,0x81,0x5B,0x16,0xF8,
            0x17,0x98,0x02,0x21,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
            0xFF,0xFF,0xFF,0xFF,0xFE,0xBA,0xAE,0xDC,0xE6,0xAF,0x48,0xA0,0x3B,0xBF,0xD2,0x5E,
            0x8C,0xD0,0x36,0x41,0x41,0x02,0x01,0x01,0xA1,0x24,0x03,0x22,0x00
        };
        unsigned char *ptr = privkey; memcpy(ptr, begin, sizeof(begin)); ptr += sizeof(begin);
        memcpy(ptr, key32, 32); ptr += 32; memcpy(ptr, middle, sizeof(middle)); ptr += sizeof(middle);
        pubkeylen = 33;
        secp256k1_ec_pubkey_serialize(ctx, ptr, &pubkeylen, &pubkey, SECP256K1_EC_COMPRESSED);
        ptr += pubkeylen;   *privkeylen = ptr - privkey;
    } else { static const unsigned char begin[] = {0x30,0x82,0x01,0x13,0x02,0x01,0x01,0x04,0x20 };
    static const unsigned char middle[] = {
            0xA0,0x81,0xA5,0x30,0x81,0xA2,0x02,0x01,0x01,0x30,0x2C,0x06,0x07,0x2A,0x86,0x48,
            0xCE,0x3D,0x01,0x01,0x02,0x21,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
            0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
            0xFF,0xFF,0xFE,0xFF,0xFF,0xFC,0x2F,0x30,0x06,0x04,0x01,0x00,0x04,0x01,0x07,0x04,
            0x41,0x04,0x79,0xBE,0x66,0x7E,0xF9,0xDC,0xBB,0xAC,0x55,0xA0,0x62,0x95,0xCE,0x87,
            0x0B,0x07,0x02,0x9B,0xFC,0xDB,0x2D,0xCE,0x28,0xD9,0x59,0xF2,0x81,0x5B,0x16,0xF8,
            0x17,0x98,0x48,0x3A,0xDA,0x77,0x26,0xA3,0xC4,0x65,0x5D,0xA4,0xFB,0xFC,0x0E,0x11,
            0x08,0xA8,0xFD,0x17,0xB4,0x48,0xA6,0x85,0x54,0x19,0x9C,0x47,0xD0,0x8F,0xFB,0x10,
            0xD4,0xB8,0x02,0x21,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
            0xFF,0xFF,0xFF,0xFF,0xFE,0xBA,0xAE,0xDC,0xE6,0xAF,0x48,0xA0,0x3B,0xBF,0xD2,0x5E,
            0x8C,0xD0,0x36,0x41,0x41,0x02,0x01,0x01,0xA1,0x44,0x03,0x42,0x00};
        unsigned char *ptr = privkey;
        memcpy(ptr, begin, sizeof(begin)); ptr += sizeof(begin);
        memcpy(ptr, key32, 32); ptr += 32;
        memcpy(ptr, middle, sizeof(middle)); ptr += sizeof(middle);
        pubkeylen = 65;
    secp256k1_ec_pubkey_serialize(ctx, ptr, &pubkeylen, &pubkey, SECP256K1_EC_UNCOMPRESSED);
        ptr += pubkeylen;     *privkeylen = ptr - privkey;
    }
    return 1;
}
```

```
int secp256k1_ec_pubkey_create(const secp256k1_context* ctx, secp256k1_pubkey *pubkey,
                               const unsigned char *seckey)   //代码1
{   secp256k1_gej pj;
    secp256k1_ge p;
    secp256k1_scalar sec;
    int overflow;    int ret = 0;
    VERIFY_CHECK(ctx != NULL);
    ARG_CHECK(pubkey != NULL);
    memset(pubkey, 0, sizeof(*pubkey));
    ARG_CHECK(secp256k1_ecmult_gen_context_is_built(&ctx->ecmult_gen_ctx));
    ARG_CHECK(seckey != NULL);
    secp256k1_scalar_set_b32(&sec, seckey, &overflow); //代码1.1
    ret = (!overflow) & (!secp256k1_scalar_is_zero(&sec));
    if (ret) {  secp256k1_ecmult_gen(&ctx->ecmult_gen_ctx, &pj, &sec); //代码1.2
                secp256k1_ge_set_gej(&p, &pj); //代码1.3
                secp256k1_pubkey_save(pubkey, &p); //代码1.4
    }
    secp256k1_scalar_clear(&sec);
    return ret;
}
```

```
int secp256k1_ec_pubkey_serialize(const secp256k1_context* ctx, unsigned char *output, size_t
*outputlen, const secp256k1_pubkey* pubkey, unsigned int flags) {  //代码2
    secp256k1_ge Q;
    size_t len;   int ret = 0;
    (void)ctx;  VERIFY_CHECK(ctx != NULL);
    ARG_CHECK(outputlen != NULL);
    ARG_CHECK(*outputlen >= ((flags & SECP256K1_FLAGS_BIT_COMPRESSION) ? 33 : 65));
    len = *outputlen;   *outputlen = 0;
    ARG_CHECK(output != NULL);
    memset(output, 0, len);
    ARG_CHECK(pubkey != NULL);
    ARG_CHECK((flags & SECP256K1_FLAGS_TYPE_MASK) ==
SECP256K1_FLAGS_TYPE_COMPRESSION);
    if (secp256k1_pubkey_load(ctx, &Q, pubkey)) { //代码2.1
        ret = secp256k1_eckey_pubkey_serialize(&Q, output, &len, flags &
SECP256K1_FLAGS_BIT_COMPRESSION); //代码2.2
        if (ret) {   *outputlen = len;  }
    }  return ret;
}
```

secp256k1 pubkey源码实现(2)

```
//代码1.1
static void secp256k1_scalar_set_b32(secp256k1_scalar *r,
                                     const unsigned char *b32, int *overflow)
{
    int over;
    r->d[0] = (uint64_t)b32[31] | (uint64_t)b32[30] << 8 | (uint64_t)b32[29] << 16 | (uint64_t)b32[28] <<
24 | (uint64_t)b32[27] << 32 | (uint64_t)b32[26] << 40 | (uint64_t)b32[25] << 48 | (uint64_t)b32[24] <<
56;
    r->d[1] = (uint64_t)b32[23] | (uint64_t)b32[22] << 8 | (uint64_t)b32[21] << 16 | (uint64_t)b32[20] <<
24 | (uint64_t)b32[19] << 32 | (uint64_t)b32[18] << 40 | (uint64_t)b32[17] << 48 | (uint64_t)b32[16] <<
56;
    r->d[2] = (uint64_t)b32[15] | (uint64_t)b32[14] << 8 | (uint64_t)b32[13] << 16 | (uint64_t)b32[12] <<
24 | (uint64_t)b32[11] << 32 | (uint64_t)b32[10] << 40 | (uint64_t)b32[9] << 48 | (uint64_t)b32[8] <<
56;
    r->d[3] = (uint64_t)b32[7] | (uint64_t)b32[6] << 8 | (uint64_t)b32[5] << 16 | (uint64_t)b32[4] << 24 |
(uint64_t)b32[3] << 32 | (uint64_t)b32[2] << 40 | (uint64_t)b32[1] << 48 | (uint64_t)b32[0] << 56;
    over = secp256k1_scalar_reduce(r, secp256k1_scalar_check_overflow(r));
    if (overflow) {  *overflow = over; }
}

//代码1.1.1
SECP256K1_INLINE static int secp256k1_scalar_reduce(secp256k1_scalar *r,
                                                    unsigned int overflow)
{
    uint128_t t;
    VERIFY_CHECK(overflow <= 1);
    t = (uint128_t)r->d[0] + overflow * SECP256K1_N_C_0;
    r->d[0] = t & 0xFFFFFFFFFFFFFFFFULL; t >>= 64;
    t += (uint128_t)r->d[1] + overflow * SECP256K1_N_C_1;
    r->d[1] = t & 0xFFFFFFFFFFFFFFFFULL; t >>= 64;
    t += (uint128_t)r->d[2] + overflow * SECP256K1_N_C_2;
    r->d[2] = t & 0xFFFFFFFFFFFFFFFFULL; t >>= 64;
    t += (uint64_t)r->d[3];
    r->d[3] = t & 0xFFFFFFFFFFFFFFFFULL;
    return overflow;
}
```

```
//代码1.2
static void secp256k1_ecmult_gen(const secp256k1_ecmult_gen_context *ctx,
                                 secp256k1_gej *r, const secp256k1_scalar *gn) {
    secp256k1_ge add;  secp256k1_ge_storage adds;
    secp256k1_scalar gnb;   int bits;   int i, j;
    memset(&adds, 0, sizeof(adds));
    *r = ctx->initial;
    /* Blind scalar/point multiplication by computing (n-b)G + bG instead of nG. */
    secp256k1_scalar_add(&gnb, gn, &ctx->blind);
    add.infinity = 0;
    for (j = 0; j < 64; j++) {
        bits = secp256k1_scalar_get_bits(&gnb, j * 4, 4);
        for (i = 0; i < 16; i++) {
            secp256k1_ge_storage_cmov(&adds, &(*ctx->prec)[j][i], i == bits);
        }
        secp256k1_ge_from_storage(&add, &adds);
        secp256k1_gej_add_ge(r, r, &add);
    } bits = 0;
    secp256k1_ge_clear(&add);
    secp256k1_scalar_clear(&gnb);
}
//代码1.3
static void secp256k1_ge_set_gej(secp256k1_ge *r, secp256k1_gej *a) {
    secp256k1_fe z2, z3;   r->infinity = a->infinity;
    secp256k1_fe_inv(&a->z, &a->z);  secp256k1_fe_sqr(&z2, &a->z);
    secp256k1_fe_mul(&z3, &a->z, &z2); secp256k1_fe_mul(&a->x, &a->x, &z2);
    secp256k1_fe_mul(&a->y, &a->y, &z3);  secp256k1_fe_set_int(&a->z, 1);
    r->x = a->x;  r->y = a->y;
}
//代码1.4
static void secp256k1_pubkey_save(secp256k1_pubkey* pubkey, secp256k1_ge* ge) {
    if (sizeof(secp256k1_ge_storage) == 64) {
        secp256k1_ge_storage s;
        secp256k1_ge_to_storage(&s, ge);
        memcpy(&pubkey->data[0], &s, 64);
    } else {
        VERIFY_CHECK(!secp256k1_ge_is_infinity(ge));
        secp256k1_fe_normalize_var(&ge->x); secp256k1_fe_normalize_var(&ge->y);
secp256k1_fe_get_b32(pubkey->data, &ge->x);secp256k1_fe_get_b32(pubkey->data + 32, &ge->y);
    }
}
```

secp256k1 pubkey源码实现(3)

```
//代码2.1
static int secp256k1_pubkey_load(const secp256k1_context* ctx, secp256k1_ge* ge,
                                 const secp256k1_pubkey* pubkey) {
    if (sizeof(secp256k1_ge_storage) == 64) {
        /* When the secp256k1_ge_storage type is exactly 64 byte, use its
         * representation inside secp256k1_pubkey, as conversion is very fast.
         * Note that secp256k1_pubkey_save must use the same representation. */
        secp256k1_ge_storage s;
        memcpy(&s, &pubkey->data[0], 64);
        secp256k1_ge_from_storage(ge, &s);
    } else {
        /* Otherwise, fall back to 32-byte big endian for X and Y. */
        secp256k1_fe x, y;
        secp256k1_fe_set_b32(&x, pubkey->data); secp256k1_fe_set_b32(&y, pubkey->data + 32);
        secp256k1_ge_set_xy(ge, &x, &y);
    }
    ARG_CHECK(!secp256k1_fe_is_zero(&ge->x));
    return 1;
}

//代码2.2
static int secp256k1_eckey_pubkey_serialize(secp256k1_ge *elem, unsigned char *pub,
                                            size_t *size, int compressed)
{
    if (secp256k1_ge_is_infinity(elem)) {   return 0;  }
    secp256k1_fe_normalize_var(&elem->x);
    secp256k1_fe_normalize_var(&elem->y);
    secp256k1_fe_get_b32(&pub[1], &elem->x);
    if (compressed) {
        *size = 33;
        pub[0] = 0x02 | (secp256k1_fe_is_odd(&elem->y) ? 0x01 : 0x00);
    } else {
        *size = 65;  pub[0] = 0x04;
        secp256k1_fe_get_b32(&pub[33], &elem->y);
    }
    return 1;
}
```

```
static void secp256k1_fe_normalize_var(secp256k1_fe *r) {
    uint64_t t0 = r->n[0], t1 = r->n[1], t2 = r->n[2], t3 = r->n[3], t4 = r->n[4];
    /* Reduce t4 at the start so there will be at most a single carry from the first pass */
    uint64_t m;  uint64_t x = t4 >> 48;   t4 &= 0x0FFFFFFFFFFFFULL;
    /* The first pass ensures the magnitude is 1, ... */
    t0 += x * 0x1000003D1ULL;
    t1 += (t0 >> 52); t0 &= 0xFFFFFFFFFFFFFULL;
    t2 += (t1 >> 52); t1 &= 0xFFFFFFFFFFFFFULL; m = t1;
    t3 += (t2 >> 52); t2 &= 0xFFFFFFFFFFFFFULL; m &= t2;
    t4 += (t3 >> 52); t3 &= 0xFFFFFFFFFFFFFULL; m &= t3;
    /* ... except for a possible carry at bit 48 of t4 (i.e. bit 256 of the field element) */
    VERIFY_CHECK(t4 >> 49 == 0);
    /* At most a single final reduction is needed; check if the value is >= the field characteristic */
    x = (t4 >> 48) | ((t4 == 0x0FFFFFFFFFFFFULL) & (m == 0xFFFFFFFFFFFFFULL)
                                                  & (t0 >= 0xFFFFEFFFFFC2FULL));
    if (x) {   t0 += 0x1000003D1ULL;
        t1 += (t0 >> 52); t0 &= 0xFFFFFFFFFFFFFULL;  t2 += (t1 >> 52); t1 &= 0xFFFFFFFFFFFFFULL;
        t3 += (t2 >> 52); t2 &= 0xFFFFFFFFFFFFFULL;  t4 += (t3 >> 52); t3 &= 0xFFFFFFFFFFFFFULL;
        /* If t4 didn't carry to bit 48 already, then it should have after any final reduction */
        VERIFY_CHECK(t4 >> 48 == x);
t4 &= 0x0FFFFFFFFFFFFULL; /* Mask off the possible multiple of 2^256 from the final reduction */
    }
    r->n[0] = t0; r->n[1] = t1; r->n[2] = t2; r->n[3] = t3; r->n[4] = t4;
#ifdef VERIFY
    r->magnitude = 1;  r->normalized = 1;
    secp256k1_fe_verify(r);
#endif
}
/** Convert a field element to a 32-byte big endian value. Requires the input to be normalized */
static void secp256k1_fe_get_b32(unsigned char *r, const secp256k1_fe *a) {
    int i;
#ifdef VERIFY
    VERIFY_CHECK(a->normalized);  secp256k1_fe_verify(a);
#endif
    for (i=0; i<32; i++) {  int j;   int c = 0;
        for (j=0; j<2; j++) {
            int limb = (8*i+4*j)/52;  int shift = (8*i+4*j)%52;
            c |= ((a->n[limb] >> shift) & 0xF) << (4 * j);
        } r[31-i] = c;
```