

Programming the Open Blockchain

Mastering Bitcoin

Andreas M. Antonopoulos • Second Edition

These notes are intended as a resource for the contributors; or past, present, or future readers of this book, and anyone interested in the material. The goal is to provide an end-to-end resource that covers all material discussed in the book displayed in an organized manner. If you spot any errors or would like to contribute, please contact the contributors directly.

1 Introduction


1.1 What is Bitcoin?

"...akin to Internet of money ... a network of propagating value ... securing ownership of digital assets through computational work"

- Bitcoin users communicate using **bitcoin protocol**, primarily via Internet
 - + available across many devices \implies easy access
 - + perfect form of money for Internet, because fast, secure, borderless
- No physical coins (or digital coins per se), but implied through **transactions** that transfer value from sender to recipient
- Prerequisite for spending bitcoin: **keys** to sign transactions and hence unlock value (spent by transferring to new owner) and **digital wallet** used to store keys
- Network **distributed**, **peer-to-peer** and new bitcoin created through **mining**
 - ◇ anyone running full bitcoin protocol stack may operate as miner
 - + mining decentralized currency-issuance and clearing function \implies no central bank required
- Computational work req. to mine dynamically adjusted with time so that block added every ~ 10 minutes (avg)
- Protocol also halves number of BTC created every 4 years \implies fixed number in circulation at 21 million (~ 2140). Diminishing rate of issuance means bitcoin **deflationary** (therefore, inflating by "printing" cannot happen)
- In summary, bitcoin consists of: (1) a decentralized peer-to-peer network (the bitcoin protocol), (2) a public transaction ledger (**the blockchain**), (3) a set of rules for independent transaction validation and currency issuance (**consensus rules**), (4) a mechanism for reaching global decentralized consensus on the valid blockchain (**POW algorithm**)

Digital Currencies Before Bitcoin

- Cryptography, bits, and the problem of digital money:
 1. Can I trust that money is authentic and not counterfeit?
 2. "Double spend" problem
 3. Can I be sure that no one else can claim this money belongs to them and not me?

Paper money handles (1) and (2) easily (also with help from digital storage and transmission by clearing through central banks. Think  over global circulation). Digital money handles (1) (and apparently (2)) through **cryptographic digital signatures**

- Late 1980s: cryptography for digital currencies (at first backed by national currency or gold) \rightarrow worked but was centralized (central clearinghouse) \rightarrow target of governments and hackers (at times

A Solution to a Distributed Computing Problem

- POW algorithm can be used to solve "Byzantine Generals' Problem" → consensus activated w/o central trusted authority → breakthrough in distributing computing and wide applicability beyond currency (ex: to prove fairness of elections, for lotteries, asset registries, digital monetization, etc.)

1.3 Bitcoin Uses, Users, and Their Stories

- *North American low-value retail*: story introduces to software, exchanges, and basic transactions of a retail consumer
- *North American high-value retail*: story introduces to "**51 percent consensus attack**" for retailers of high-value items
- *Offshore contract services*: story examines the use of bitcoin for outsourcing, contract services, and international wire transfers (?)
- *Web store*
- *Charitable donations*: story shows use of bitcoin for global fundraising across currencies and borders, and use of open ledger for transparency ((+) quick distribution of funds)
- *Import/export*: story shows use of bitcoin for large business-to-business international payments tied to physical goods ((+) accelerate process of payment for import)
- *Mining for bitcoin*: story will examine "industrial" base of bitcoin (the specialized equipment used to secure bitcoin network and issue new currency)

1.4 Getting Started

- Client application must be used to access bitcoin protocol → bitcoin wallet is UI for bitcoin system (web browsers and HTTP protocol analogy). Different wallets vary in quality, performance, security, privacy, and reliability
 - ◇ "Satoshi Client" or "Bitcoin Core" is reference implementation of bitcoin protocol that includes wallet. It is derived from original implementation

1.4.1 Choosing Your Bitcoin Wallet

1-5 by platform, 6-8 by degree of autonomy and type of interaction with bitcoin network

- *Desktop wallet*: first (reference implementation); (+) features, autonomy, control; (-) security disadvantage from OS
- *Mobile wallet*: most common; (+) designed to be simple and easy to use (some fully featured for power users)
- *Web wallet*: accessed through web browser (wallet stored on third-party server); (+) ease-of-use; (-) third party keeps control of bitcoin keys for user, so don't store large amounts of bitcoin (though some use client-side code running in web browser)
- *Hardware wallet*: wallet self-contained on special-purpose hardware (USB for web browser or NFC for mobile); (+) very secure, can store large amounts
- *Paper wallet*: for long term storage (offline storage also called "cold storage"); (+) very secure

- *Full-node client*: entire history of all transactions ever stored, manages users' wallets, direct initiation of transactions on network, handles all aspects of protocol, can independently validate entire blockchain and any transaction; (+) complete autonomy and independent transaction verification; (-) consume substantial computing resources (<125 GB disk, 2GB RAM)
- *Lightweight client (or SPV)*: connects to full nodes for access to bitcoin transaction info, wallet stored locally, creates and validates and transmits transactions independently, interact directly with bitcoin network (no intermediary)
- *Third-party API client*: interacts with network through third-party system or API (indirect), wallet can be stored by user or third party servers, but all transactions go through third party

1.4.2 Quick Start

"a wallet is simply a collection of addresses and the keys that unlock funds within"

- How can Alice get started? → get a bitcoin wallet (book uses mobile wallet "Mycelium" for Android. When application first runs, it creates a wallet, as seen in Figure 1-1 of the book). Important features here: bitcoin address and associated QR code, private key (generated alongside address)
 - ◊ bitcoin address starts with a 1 or 3, acts like an email address (can be shared w/o security risk for users to send bitcoin to wallet. Unlike email, can create as many addresses as you like (many new wallets create a new address for each transaction to maximize privacy))
 - ◊ bitcoin not part of network or any externally identifiable information (including user's name) until associated with transaction

1.4.3 Getting Your First Bitcoin

- "Cannot yet buy bitcoin at a bank or foreign exchange kiosk"
- Bitcoin transactions are **irreversible** → introduces risk of defrauding by reversing electronic payments → to mitigate, companies e-payment for bitcoin require identity verification and credit-worthiness checks ⇒ as new user, cannot instantly buy with credit card
- As new user, you can: find a friend or use <https://meetup.com>, use classified service like localbitcoins.com to find seller locally and meet up in person, use bitcoin ATM (<http://coinatmradar.com>), earn bitcoin by selling products or services, use a bitcoin currency exchange linked to your bank account (online)

Aside: advantage of bitcoin is privacy (not required to divulge sensitive and personally identifiable info to third parties). This is not true where bitcoin touches traditional systems (ex: currency exchanges). Also, once bitcoin address attached to identity, all associated transactions easy to track, which is one reason to unlink exchange account to wallet

1.4.4 Finding the Current Price of Bitcoin

- Must first agree on **exchange rate** between BTC and USD (done automatically on most wallets). But who sets the bitcoin price? → Short answer: markets → Long answer: Bitcoin has **floating exchange rate** → price of bitcoin in USD calculated in each market based on most recent trade (from BTC to USD) → price fluctuates minutely several times per second → pricing service calculates volume-weighted average of all these trades across various markets to find broad market exchange rate
- Most popular pricing services: Bitcoin Average (exchange rate for bitcoin), CoinCap (market capitalization and exchange rate for hundreds of cryptocurrencies), CMEBRR (a reference rate used for institutional and contractual reference, provided as part of investment data feeds by CME)

1.4.5 Sending and Receiving Bitcoin

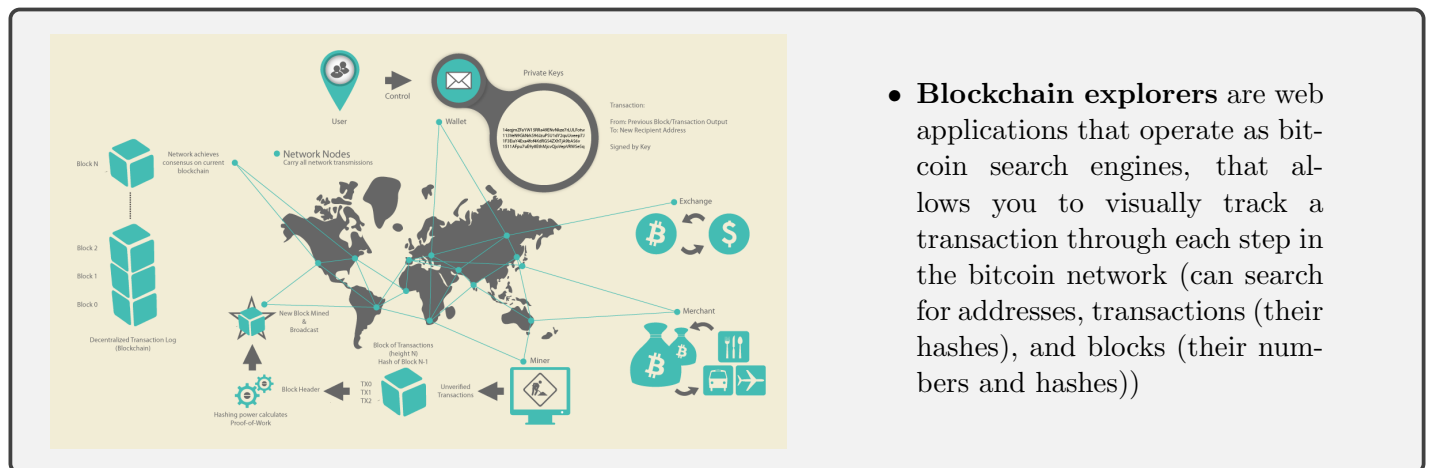
- Alice-Joe: Alice (Mycelium) clicks receive to "listen" to published transactions on bitcoin network with one that matches her address. Joe (Airbits) inputs destination address (can scan QR code) and amount (in BTC or USD) and sends → mobile wallet constructs transaction that assigns amount to address and signs with private keys to tell bitcoin network that Joe has authorized transfer of value to Alice's new address → transmission propagates through peer-to-peer protocol across bitcoin network, where in less than a second almost all of the well connected nodes in network receive transaction and see Alice's address for first time → few seconds later, Alice's wallet indicates that it is receiving x BTC
- At first, Alice sees transaction as "unconfirmed", meaning that it has propagated to network but not yet recorded on blockchain. Clearing, as is known in traditional finance, happens every ~ 10 minutes (avg.)

2 How Bitcoin Works

2.1 Transactions, Blocks, Mining, and the Blockchain

- Unlike traditional banking, bitcoin system based on *decentralized trust*, where trust is achieved as an *emergent property* from interactions of different participants in the bitcoin system

2.1.1 Bitcoin Overview



2.1.2 Buying a Cup of Coffee

- Alice buys from Bob, who has added bitcoin to point of sale system → displays payment request QR code (components of URL: a bitcoin address, payment amount, label for recipient address, description for payment) → Bob sees transaction on register after a few seconds
- Bitcoin network can transact in fractional values, from 1 satoshi ($1/100000000$) to 21,000,000

2.2 Bitcoin Transactions

- *Chain of ownership* concept and authorization

2.2.1 Transaction Inputs and Outputs

- Input contains info to confirm ownership, output to assign new owners

Transaction as Double-Entry Bookkeeping

Inputs	Value	Outputs	Value
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:		Total Outputs:	
	0.55 BTC		0.50 BTC
<div style="display: flex; justify-content: space-between;"> <div>Inputs</div> <div>0.55 BTC</div> </div> <div style="display: flex; justify-content: space-between;"> <div>Outputs</div> <div>0.50 BTC</div> </div> <div style="display: flex; justify-content: space-between;"> <div>Difference</div> <div>0.05 BTC (implied transaction fee)</div> </div>			

- Transaction also contains proof of ownership for each input (through a digital signature), which can be independently validated by anyone. In bitcoin terms, "spending" is signing a transaction that transfers value from previous transaction over to new owner identified by a bitcoin address

2.2.2 Transaction Chains

transaction 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18

INPUTS From: m (previous transactions Joe has received): Joe 0.1000 BTC

OUTPUTS To: Output #0 Alice's Address 0.1000 BTC (spent), Transaction Fees: 0.0000 BTC

transaction 0627052b6f28912f2703066a912ea577f2ce4da94ca5a5fbd8a57286c345c2f2

INPUTS From: Alice 0.1000 BTC

OUTPUTS To: Output #0 Bob's Address 0.0150 BTC (spent), Output #1 Alice's Address (change) 0.0845 BTC (unspent), Transaction Fees: 0.0005 BTC

transaction 2bbac8bb3a57a2363407ac8c16a67015ed2e88a4388af58cf90299e0744d3de4

INPUTS From: Bob 0.0150 BTC

OUTPUTS To: Output #0 Gopesh's Address 0.0100 BTC (unspent), Output #1 Bob's Address (change) 0.0045 BTC (unspent), Transaction Fees: 0.0005 BTC

- Output of one transaction is input of next transaction
- Alice's key provides signature that unlocks previous transaction output, proving to network that she owns funds. She attaches payment to Bob's address, transferring value from Alice to Bob (Bob must also sign to spend)

2.2.3 Making Change

- Unless wallet can aggregate exact amount (including transaction fees), change exists. This is because transaction inputs, like currency notes, cannot be divided.
- Change address* does not have to be same address as that of input, and for privacy reasons is often a new address from owner's wallet
- Depending on wallet software, can aggregate many small inputs or few to one large input for payments (predicament is exactly as with cash. Bitcoin wallet developers strive to find balance)

2.2.4 Common Transaction Forms on Bitcoin Ledger

Common Transaction

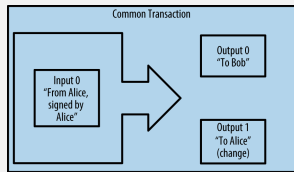
Input 0
"From Alice,
signed by Alice"

➔

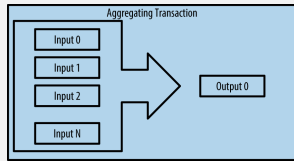
Output 0
"To Bob"

Output 1
"To Alice"
(change)

- Most common transaction



- Transaction aggregating funds (ex: wallets to clean up change – equivalent to exchanging change for bill)



- Transaction distributing funds (ex: commercial entities distributing funds for payroll payments to employees)

2.3 Constructing a Transaction

- Solution of appropriate inputs and outputs handled by wallet (i.e. HOW to aggregate transaction inputs (see above) and what will change amount and transaction fee be). User need only specify destination address and amount
- Transactions (including signatures) can be constructed offline

2.3.1 Getting the Right Inputs

- For (total) transaction inputs, need to find BTC \geq payment amount \rightarrow keep track of unspent OUTPUTS To associated to **your** address in transactions (full node clients see **all** unspent OUTPUTS To on bitcoin network (this allows them to also verify whether inputs from incoming transactions satisfy condition(s) to spend referenced outputs) \rightarrow do not have this functionality? query bitcoin network/ask full node client through API call (example in book has API call constructed as an HTTP GET command to a specific URL, while response received through command-line HTTP client cURL). Information obtained: unspent amounts under ownership of specified address with reference to transaction that contains it).
 - single unspent output preferred, otherwise "rummaging" occurs

2.3.2 Creating the Outputs

- Output created in form of a script that encumbers output value with demand for signature for redemption (signature is a solution to script). Additional outputs in same transaction also sometimes made for change. Transaction fees incurred implicitly (in difference between I/O) to help ensure transaction processed by network in timely fashion (transaction fee provides incentive for miners)

2.3.3 Adding the Transaction to the Ledger

"...new blocks (once added to blockchain) increasingly trusted by network as more blocks added"

Transmitting the Transaction

- Transaction contains within itself everything needed to process, so doesn't matter how or where it is transmitted to bitcoin network
- Bitcoin network *peer-to-peer* (each client connected to several other clients). A primary purpose of network is propagate transactions and blocks to all clients

How it Propagates

- *bitcoin node* anything that participates in network by speaking bitcoin protocol. Wallet can send transaction to any node over any time of connection (wired, Wifi, etc.). Not required to connect to sender/receiver directly or over same Wifi network because of propagation technique called *flooding*

Bob's View

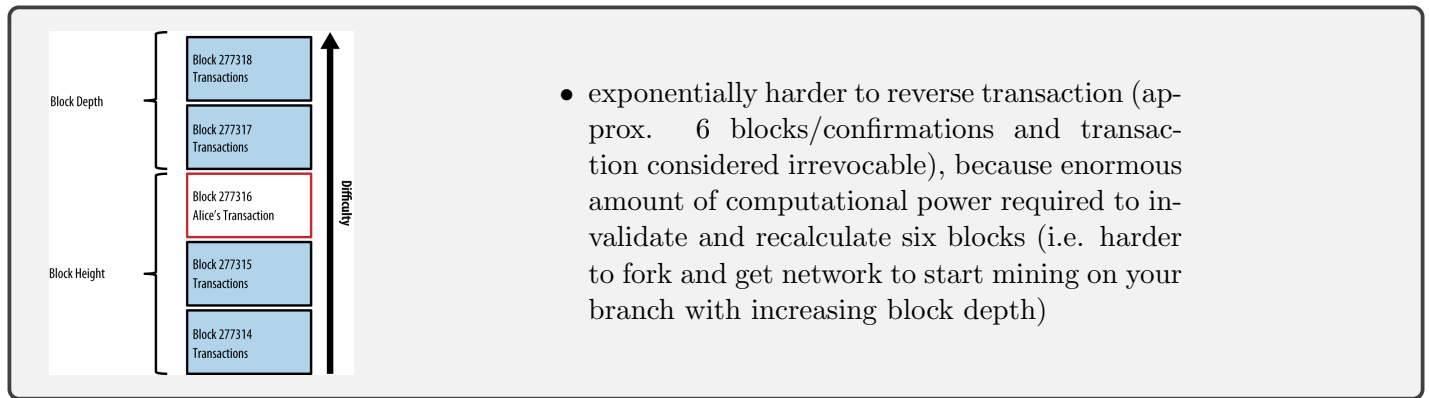
- Bob's wallet identifies transaction (constructed by Alice's wallet) as incoming payment because contains output redeemable by Bob's keys
- Bob's wallet can also independently verify that transaction is well formed (proper syntax, structure – see all other consensus rules), uses previously unspent inputs (no double-spend), and contains sufficient transaction fees to be included in next block (can assume, with little risk, that transaction will shortly be included on blockchain and confirmed (next block or few blocks after)
 - Can only spend BTC after confirmation (i.e. after put on blockchain). For small-value transactions, the risk of exchanging product/service for BTC prior to confirmation is usually lower than risk of delaying service

2.4 Bitcoin Mining

- Higher computation to prove, lower computation to verify as proven (part of broader topic of *Proof-of-Work* and *computational trust*) + adjustable difficulty
- Mining process serves two purposes: (1) provides security for bitcoin transactions by rejecting invalid transactions using bitcoin *consensus rules* (2) created new bitcoin in each block (like "printing money")
- Delicate balance between *cost* (electricity and other costs of operation) and *reward* (a new bitcoin + transaction fee – only collected after block validated through rules of consensus (provides additional layer of security)) required to remove need for central authority
- Algorithm for POW involves repeatedly hashing header of block and some random number with SHA256 cryptographic algorithm until a solution "matching predetermined pattern emerges" (i.e. a specified number of leading zeroes)
- As number of miners increase and with intro of more advanced/powerful hardware (ex: ASICs), hashing power of network also increases → must increase target difficulty to keep confirmation rate at 10 minutes (avg). Use of mining pools can be used to help offset this exponential increase in hashing power and, by extension, difficulty

2.5 Mining Transactions in Blocks

- Transaction received. Each node maintains a *temporary pool* of unconfirmed transactions → miners add transactions to *candidate block*, prioritized by highest transaction fees first (created block contains "fingerprint" of parent block). Each miner includes coinbase transaction in own candidate block (first transaction), containing reward to his wallet (newly created bitcoin (same for all, decreases over time) + sum of transaction fees) → mining begins as soon as previous block received on network → solution to POW algorithm found, block broadcast, verified by nodes, and then added to blockchain (process takes 10 minutes, on average). Reward now spendable and divisible to pool based on amount of contributed work



2.6 Spending the Transaction

- Once in blockchain, transaction spendable. All nodes maintain local perspectives of blockchain that re-converge to a global perspective once block propagated and validated (the level of independence and decentralization each node has in maintaining blockchain depends on capacity of client (ex: full nodes vs. lightweight clients)). Regardless, once on the blockchain, client able to independently verify transaction as valid and spendable. Full-node clients can track source of funds through entire transaction chain. Lightweight clients verify transactions through Simplified Payment Verification by confirming that transaction is in blockchain and has several blocks mined after it.

3 Keys, Addresses

- Bitcoin based on cryptographic techniques used to produce, for example, digital signatures (knowing without receiving), digital fingerprints (prove authenticity of data), etc. Encryption not an important part of bitcoin.

3.1 Introduction

- Ownership of bitcoin done through:
 - *Digital keys*: enables ownership attestation and, more broadly, decentralized trust and control (cryptographic-proof security model based on fact that functions are one-way); exists as *public/private key pair* (former like BA number, latter like pin number); on wallet not network, no bitcoin protocol required (no Internet)
 - *Bitcoin addresses*: hash of the public key (most cases) (is the "digital fingerprint" of public key) (like beneficiary name on cheque). In most cases, generated from and corresponds to public key → abstraction allows for flexible destination address (company accounts, paying for bills, pay to cash) (ex: see pay to script hash)
 - *Digital signatures (or witness)*: can only be generated by private key, used to spend funds and testify true ownership; In most cases, a valid digital signature required to be in blockchain

3.1.1 Public Key Cryptography

Aside: 384-bits deemed enough for T.S. documents in elliptic curve cryptography

3.1.2 Private and Public Keys

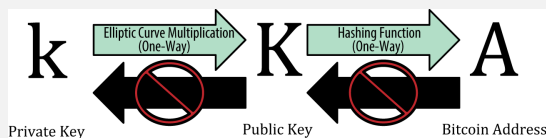


Figure 1: cryptographically generated key-pair used to control bitcoin

- Private key used to sign transactions and used to spend bitcoin. Public key used to receive funds
- Mathematical relation between s.k. and p.k. allows use of s.k. to generate signatures on messages and allows transactions to be validated against p.k. w/o revealing s.k.
- To spend, present digital signature and p.k. (digital signature different each time but from same s.k.) so ownership to spender confirmed and transaction validated on network
- Wallets can store p.k./s.k. as key pair or just s.k. since p.k. derived from it
- Asymmetric cryptography (public/private key pair) used to allow for digital signatures (s.k. applied to digital fingerprint of transaction to produce un-forgable signature, however anyone with access to public key and digital signature can independently (again, going to the theme of decentralized) verify signature and hence transaction as valid)

3.1.3 Private Keys

- Picked at random (ex: toss coin 256 times to generate binary); control of s.k. \implies control of funds, since purpose is to spend by signing (proves ownership). Thus, must keep secret and back up.

Generating a Private Key from a Random Number

- Question: best way to produce 256 bits of entropy (or number from 1 to 2^{256})? (more precise: s.k. any number from 1 to $n-1$, $n = 1.158 \times 10^{77}$ (slightly less than 2^{256}), defined as the order of the elliptical curve used in bitcoin)
- Answer: Use CSPRNG to produce string of 256 bits \rightarrow feed into SHA256 algorithm to produce 256-bit number \rightarrow check if less than $n-1$

3.1.4 Public Keys

Elliptical Curve Cryptography

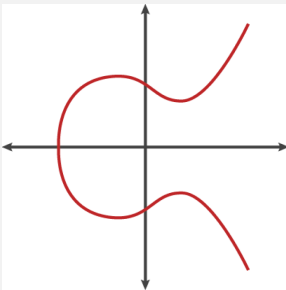


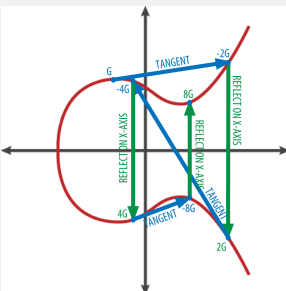
Figure 2: function is asymmetric; x approaches p

- **Function:** $y^2 = (x^3 + 7)$ over finite field of prime order $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^4 - 1 \approx 2^{256}$ (mathematics same as over real numbers)

- **Elliptic Curve Math:**

- Three cases: (1) line through P1 and P2 – must intersect third point P3 prime on curve; (2) P1 = P2 – intersects second point P3; (3) P1 and P2 same x coordinate, opposite y coordinate – vertical line
- Operation – *ADD*: $P3 = P1 + P2$
 - * In case (3), P3 is "point at infinity" and plays role of zero (ex: $P1 + P3 = P1$)
 - * *ADD* operation associative, meaning $(A+B)+C = A+(B+C)$
- Operation – *MULTIPLY*: $k * P = P + P + P + \dots + P$ (k times), where k is whole number and P is point on curve

Generating a Public Key



- $K = k * G$: K is public key, k is randomly generated s.k., G is generator point (same for everyone \Rightarrow a s.k. multiplied by G always results in same p.k.). G predetermined (specified by secp256k1 standard)
- Function is "trap door" fn (i.e. best chance at reverse operation (division) is brute force search) \Rightarrow p.k. can be shared freely

3.2 Bitcoin Address

1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy : (1) Public Key \rightarrow (2) SHA256 \rightarrow (3) RIPEMD160 \rightarrow Public Key Hash (160 bits) \rightarrow (4) Base58Check Encode with 0x00 version prefix \rightarrow Bitcoin Address (Base58Check Encoded Public Key Hash)

- Hashing algorithms used extensively in bitcoin: bitcoin addresses, script addresses, mining P.O.W. algorithm
- A "good" hashing fn meets the following requirements: (1) Easy to compute, (2) Uniform distribution, (3) Collision resistant (must be unique), (4) Pre-image resistant, (5) Second pre-image resistant

Table 1: My caption

Type	Version Prefix (hex)	Base58 Result Prefix
Bitcoin address	0x00	1
Pay-to-script hash address	0x05	3
Bitcoin testnet address	0x6F	m or n
Private key WIF	0x80	5, K, or L
BIP-38 encrypted private key	0x0142	6P
BIP-38 extended public key	0x0488B21E	xpub

Table 2: My caption

Type	Prefix	Description
Raw	None	32 bytes (256 bits)
Hex	None	64 hexadecimal digits
WIF	5	Base58Check encoding: Base58 w/ version prefix of 128 and 32 bit checksum
WIF-compressed	K or L	As above, w/ added suffix 0x01 before encoding

3.3 Base58 and Base58Check Encoding

Base58 alphabet: 123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

- Base58 a text-based binary encoding format that offers balance between compact representation, readability, and error detection and prevention. Is a subset of Base64 (no '0', 'O', 'l', 'I', '+', and '/')

Base58Check Encoding: Payload → (1) add version prefix → Version—Payload → (2) Hash (version prefix + payload) → SHA256 → SHA256 → (3) add checksum (first 4 bytes) of 32 bytes → Version—Payload—Checksum → (4) Base58 encode → Base58Check Encoded Payload

- Base58Check adds extra security by being able to better detect and prevent transcription and typing errors. Decoding software calculates checksum of data and compares with checksum included (this works because checksum derived from hash of data)(match required, otherwise loss of funds)
- Version byte helps humans to identify type of data:

3.3.1 Key Formats

Private Key encoding formats (+ Hex compressed)

- (1) and (2) rarely shown to user. Used internally in software. (3) and (4) used to import/export between wallets and for QR code
- Important point: all representations of same 256-bit number
- In Bitcoin Explorer, can use command: "wif-to-ec" to show that WIF and WIF-compressed show same private key

Decode from Base58Check

- In Bitcoin Explorer, use "base58check-decode" command (in example from book, version prefix in WIF format. the suffix 01 in payload signals that derived public key is to be compressed)

Encode from Hex to Base58Check (opposite of previous)

- Use "base58check-encode" and provide hex private key. Resulting Base58Check encoded key has prefix "K" to indicate that "compressed" private key is to be used to produce compressed public key

Public Key formats

- Recall: p.k. a point (x,y) on elliptic curve: x equals 256-bit number, y equals 256-bit number $\rightarrow K$ equals **04 (uncompressed) or 02 or 03 (compressed (not plus 520))** plus 520-bit number (130 hex)($x?y$)
- Like with private keys, many encoding formats available

Compressed public keys

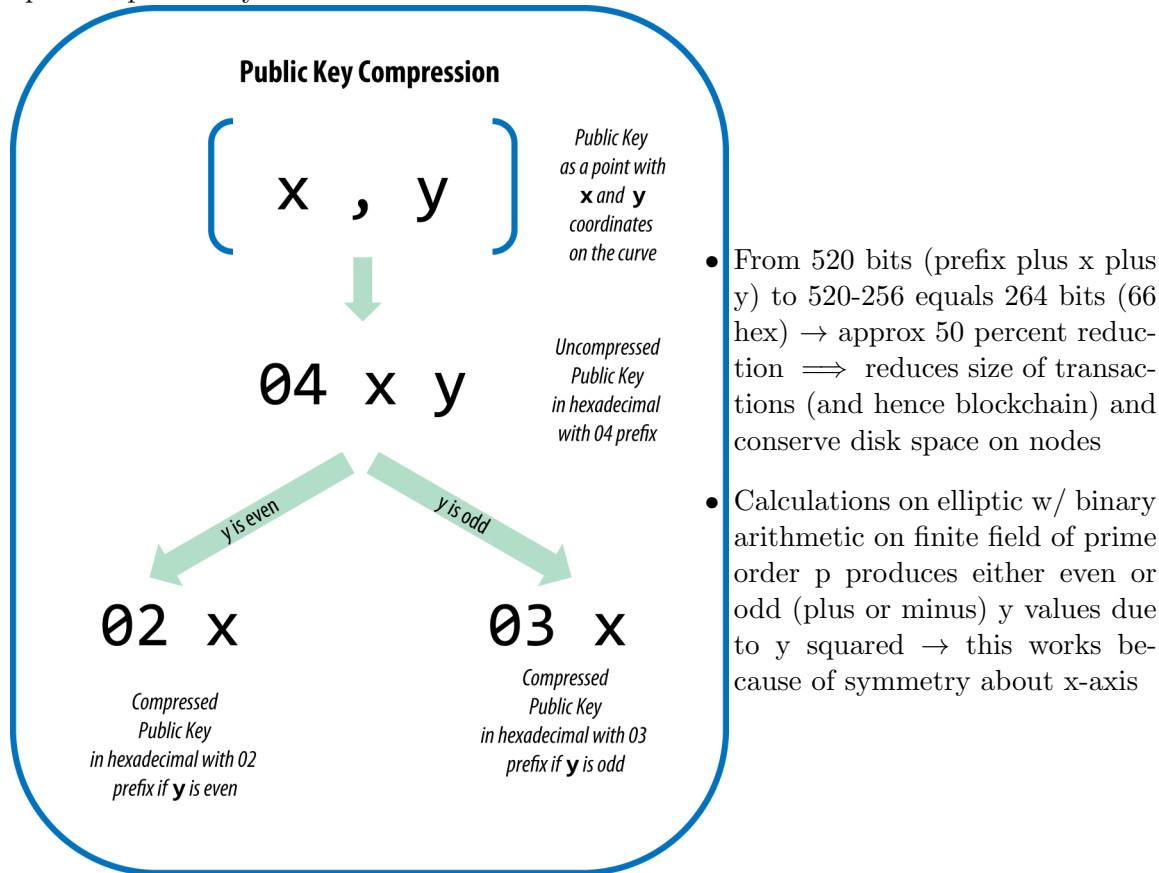


Figure 3: public key compression

- Problem: single private key can produce two different bitcoin addresses (compressed or uncompressed)(however, private keys identical for both addresses), so how do you account for clients that do not support compressed public keys? For example, when importing private keys from one wallet application to another, because the new wallet needs to scan blockchain to find transactions corresponding to imported key, which address to scan for? (both valid but different) \rightarrow when exporting, WIF used to represent s.k. imported differently to indicate whether comes from older wallet (uncompressed p.k. and hence uncompressed address) or newer wallet (compressed p.k. and hence com. address (see next section))

Compressed private keys

- Irony: when s.k. exported as WIF-compressed, it is 1-byte longer (plus 01 suffix for hex compressed). Private key "compressed" signifies that it belongs to newer wallet and really means "private key from which only compressed p.k. should be derived"
 - addition of one-byte suffix results in first character of Base58 encoding to change from 5 to K or L
- With goal of signalling to wallet importing s.k. of whether it must search blockchain for compressed or uncompressed public keys and addresses, newer wallets only ever export as WIF-compressed (thus resulting in compressed public keys and addresses) and older wallets only ever export as WIF (thus resulting in uncompressed public keys and addresses)

3.4 Advanced Keys and Addresses

3.4.1 Encrypted Private Keys (standardized by BIP-38)(section uses bitaddress.org to exemplify

- Goal: balance between **confidentiality** and **availability**, Solution: BIP-38 encrypted private keys (standard of encryption: AES (see block cyphering), established by NIST)
- Technique: s.k. (usually WIF (prefix less than or equal to) -encoded) and passphrase as input → Base58Check-encoded encrypted s.k. w/ prefix 6P as output. Passphrase now required to decrypt and hence use/display s.k. (many wallets now recognize format and have automatic prompts)
- BIP38 encrypted s.k. plus paper wallets for extra security

3.4.2 Pay-to-Script Hash (P2SH) and Multisig Addresses (standardized by BIP-16)

- Recall: traditional bitcoin addresses have prefix "1", and designate an owner of a public key (P2PKH?). When bitcoin sent to address, funds accessed and able to be spent through presentation of s.k. signature and p.k. hash to confirm ownership
- Now: pay-to-script hash (P2SH) addresses have prefix "3", and designate a hash of a script. Purpose: move the responsibility for supplying the conditions to redeem a transaction from sender of funds to redeemer
- Encoding: public key hash (NO) script has (YES) equals RIPEMD160(SHA256(public key (NO) script (YES))
 - script defines who can spend a transaction output
 - script has encoded in Base58Check w/ version prefix '5', which results in an encoded address starting with a '3'
 - In Bitcoin Explorer, can use following commands to derive P2SH address: "script-encode" → "sha256" → "ripemd160" → "base58check-encode"

Multisignature addresses and P2SH (most common, but not only, implementation of P2SH function)

- script requires more than one signature to prove ownership and therefore spend funds
 - **M-of-N multisig:** M signatures (called threshold) required from total of N keys, M less than or equal to N (ex: Bob the coffee shop owner, 1-of-2 signatures (1 key for him, 1 for wife) → "joint account")

4 Wallets

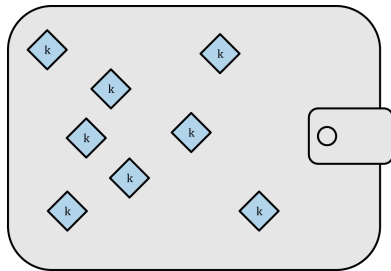
- Wallet definition: application and primary UI, controls access to user money, manages keys and addresses, tracks balance, creates and signs transactions (high-level); data structure used to store and manage user's keys (low level (programming)) – focus of chapter

4.1 Wallet Technology Overview

- bitcoin wallets contain only keys, not bitcoin. Bitcoin recorded in network blockchain in the form of transaction outputs (often noted as vout or txout). User control of bitcoin done by signing transactions with keys, thereby proving ownership of transaction outputs.
- Primary classification:
 - Nondeterministic (JBOK) (ex: Bitcoin Core (100))
 - Deterministic

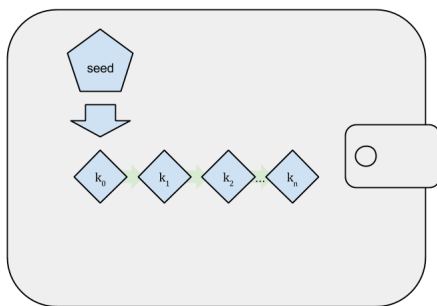
4.1.1 Nondeterministic (Random) Wallet

- Description: each s.k. independently generated from random (no relation). Bitcoin Core started with 100, used each key once, generated more as needed
 - Adhering to principle of avoiding address reuse (privacy through (lack of) association) cumbersome if you want to back up every key



4.1.2 Deterministic (Seeded) Wallet

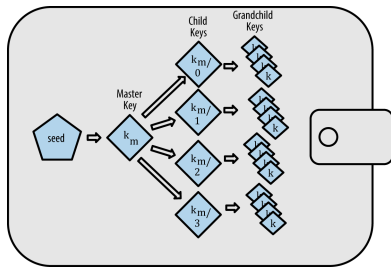
- seed is sufficient to derive all keys → one input required → easy migration of keys between wallet implementation (import/export seed only)
- HD wallet most advanced form



4.1.3 HD Wallets (standardized by BIP-32/BIP-44)

- Advantages: create sequence of pks w/o access to sk (can use on insecure server, use wallet for receive only capacity, issuance a different public key for each transaction. p.k.s do not need to be preloaded or derived in advance.

- tree structure can be used to represent additional organizational meaning (ex: specific branch of subkeys to receive incoming payments and different branch to receive change from outgoing payments, corporate settings (branches to apartments, subsidiaries, specific fns, or accounting categories))



4.1.4 Seeds and Mnemonic Codes (standardized by BIP-39)

- representing seeds from mnemonics makes easier to transcribe, record/read without error, export and import to another wallet for backup and recovery (mnemonics now inter-operable across most wallets)
 - 0C1E24E5917779D297E14D45F14E1A1A → army van defense carry jealous true garbage clan echo media make cruch

4.1.5 Wallet Best Practices

- common standard for bitcoin wallets to make them inter-operable, easy to use, secure, and flexible: mnemonic code word (BIP-39), HD wallets (BIP-32), multipurpose HD wallet structure (BIP-43), multi-currency and multiaccount wallets (BIP-44)
 - Bread wallet, Copay, Multibit HD, Mycelium; hardware wallets: keepkey, Ledger, Trezor

4.1.6 Using a Bitcoin Wallet

- Gabriel and his Trezor wallet (USB device with two buttons, that stores keys (in form of HD wallet), that signs transactions. All industry standards (above) met. When first used, mnemonic seed generated from built in random number generator (backup created through paper backup, recovery now possible (both hardware and software)). Note, mnemonic in form of numbered spaces, so sequence important (12 or 24 common). For first implementation of web store, one bitcoin address used for all orders by all customers (has drawbacks, can be improved with HD wallet)

4.2 Wallet Technology Details

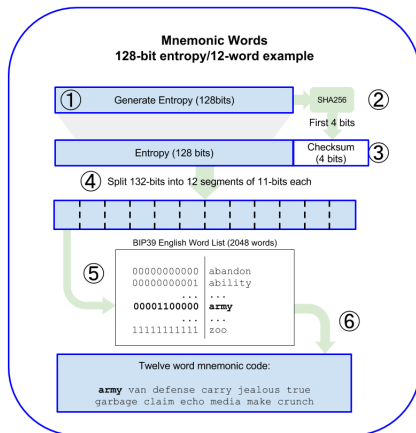
4.2.1 Mnemonic Code Words (BIP-39)

- Definition: word sequence used to encode a random number seed to derive wallet through its seeds
 - Aside: brainwallets does not equal mnemonic words → brainwallets consist of words chosen by user (not secure (poor randomness))
 - Though BIP-39 industry standard, Electrum implements different mnemonic code standard (predates)

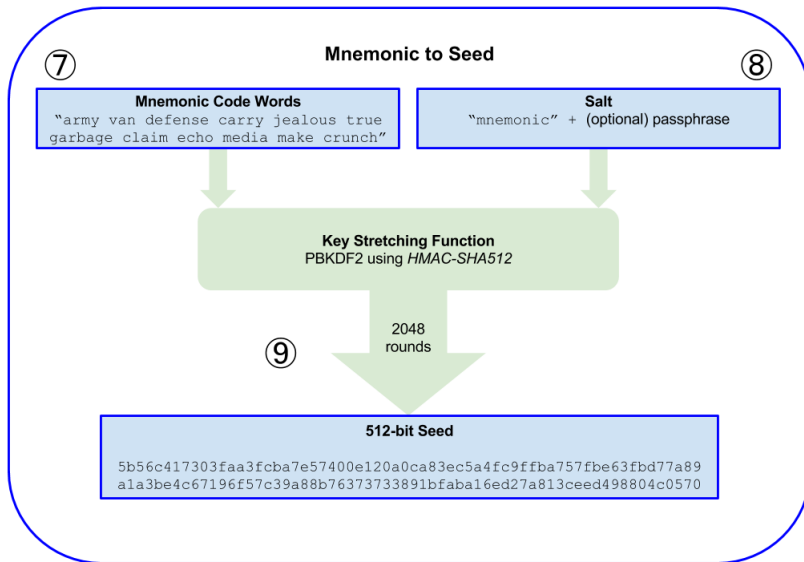
Generating mnemonic words

Table 3: My caption

Entropy (bits)	Checksum (bits)	Entropy + checksum (bits)	Mnemonic length (words)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
286	8	264	24



From mnemonic to seed



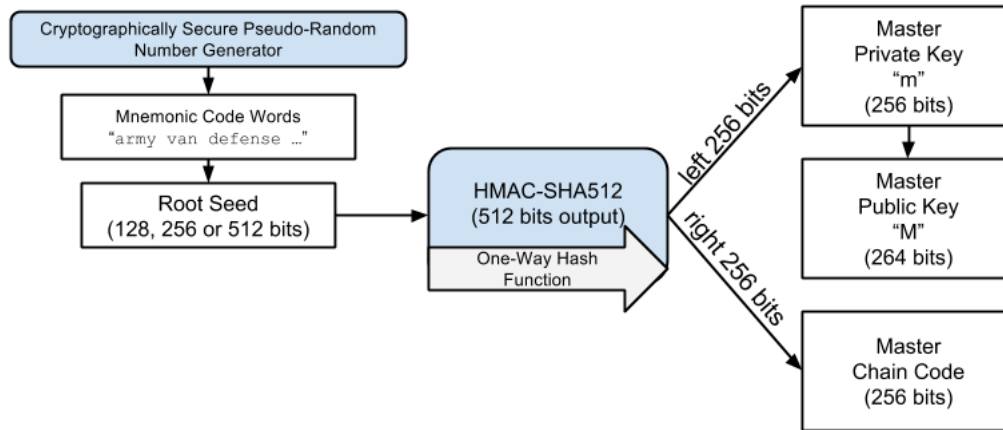
Optional passphrase in BIP39

- no passphrase versus passphrase leads to a different seed. For given mnemonic, every possible passphrase leads to a different seed \implies no "wrong passphrase"
- Passphrase features: (1) a second factor (protects against mnemonic backups), (2) duress wallets for distraction
- Risk of loss: (1) wallet owner dead \rightarrow passphrase lost \rightarrow seed useless and wallet funds lost forever, (2) backup passphrase in same place as seed \rightarrow purpose of second factor defeated

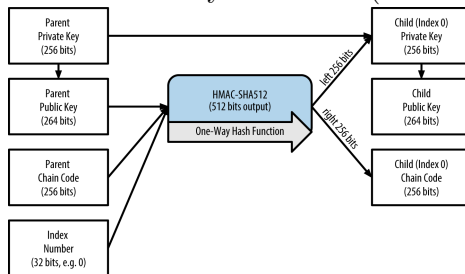
Working with mnemonic codes

- BIP-39 implemented as a library in many programming languages: python-menomic for Python, bitcoinjs/bip39 for JavaScript, libbitcoin/menominic for C++
- BIP-39 implemented as a standalone webpage (offline (in a browser) or online: <https://iancoleman.github.io/bip39>)

4.2.2 Creating an HD wallet from the seed



Private child key derivation (CKD function)



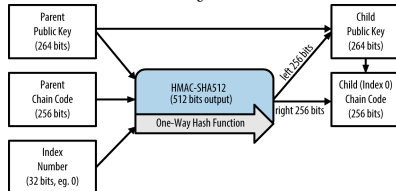
Using derived child keys

- child sk indistinguishable from nondeterministic keys (fact that part of key and sequence invisible outside HD wallet)
- from above child sk \rightarrow (???) parent sk, cant know siblings from one child \rightarrow need parent and parent chain code to derive all children, and then child equivalent for grandchildren
 - Use of child sk \rightarrow derive pk and bitcoin address to sign transactions to spend anything paid to that address

Extended keys:

- An extended key ("extensible key") consists of a private or public key and chain code (the essentials)(stored and represented as concatenation of 256-bit key and 256-bit chain code into 512-bit sequence)
 - extended sk can create complete branch (child sk and pk), extended pk
 - * give extended key \rightarrow give access to while branch

Public child key derivation



Application: very secure public key-only deployments (server has copy of epk and no sks, able to produce infinite number of pks and bitcoin addresses to spend money to, esk stored on more secure server to derive all corresponding sks to sign transactions and spend money)

- ex: ecommerce, cold storage, or hardware wallet usage for esk (epk can be online w/o risk (to sign and spend, can use offline signing bitcoin client or directly on hardware device (ex:Trezor))

4.2.3 Using an extended public key on a web store

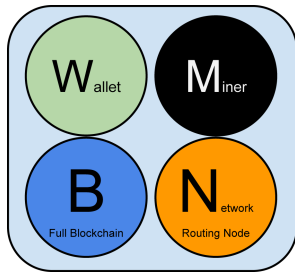
- Gabriel → one address for all orders (overwhelming and lack of security) → export on xpub from Trezor to website via Mycelium Gear (open-source webstore plugin) to device. Unique address for every order (esk stays on Trezor)

5 The Bitcoin Network

5.1 Peer-to-Peer Network Architecture

- Characteristics: built on top of Internet, equality and shared burden, mesh network with flat topology (
- to access, must run protocol

5.2 Node Types and Roles



- Routing function common to all (validation of transactions and blocks and discover + maintenance of connections to peers)

Figure 4: full node

- Full nodes: maintain complete and update copy of blockchain → authoritative and anonymous verification of any transaction w/o external reference
- Light-weight nodes: contain subset of blockchain, verify transactions through SPV method Mining nodes: run specialized hardware for POW algorithm. Can be full or lightweight node (latter depends on pool server to maintain full node) Wallets: can be part of full node (most btc desktop clients) or lightweight
- In addition to P2P protocol, specialized mining pool protocols and lightweight-client access protocol

5.3 The Extended Bitcoin Network

- Main bitcoin network runs bitcoin P2P protocol through various versions of bitcoin reference client (Bitcoin Core) or other implementations (ex: Bitcoin Classic, Bitcoin Unlimited, BitcoinJ, Libbitcoin, btcd, and bcoin) split between listening functions, mining, network edge routing (ex: exchanges, wallets, block explorers, merchant payment processing), etc.
- Extended: bitcoin p2p protocol + specialized protocols. Attached to the main bitcoin P2P network are a number of pool servers and protocol gateways that connect nodes running other protocols. These other protocol nodes are mostly pool mining nodes (see [mining]) and lightweight wallet clients, which do not carry a full copy of the blockchain.

5.4 Bitcoin Relay Network

- Problem: Bitcoin P2P network exhibits too high of network latency for miners (competition time sensitive, network latency in propagating winning block to start of next round directly related to profit margins)
- Solution: Bitcoin Relay Network (network that minimizes latency and quickly synchronize blocks between miners)(consists special nodes hosted on Amazon Web Services infratstructure to connect miners and mining pools)

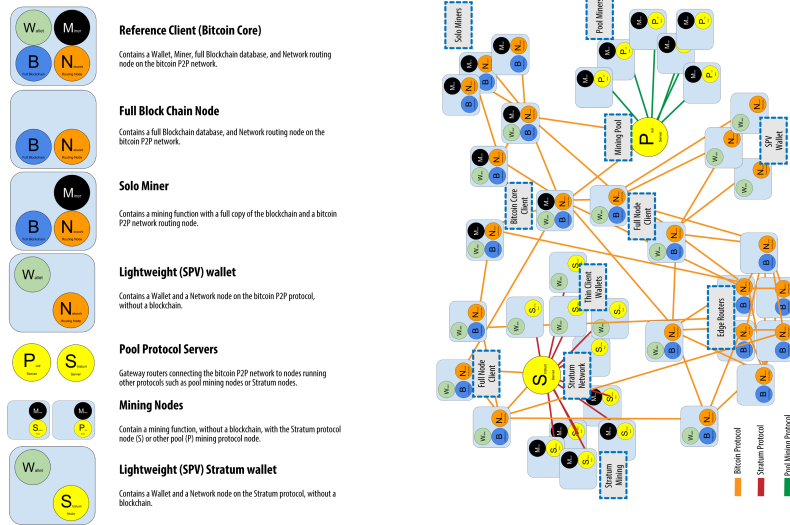


Figure 5: Different types of nodes on the extended bitcoin network

Figure 6: The extended bitcoin network showing various node types, gateways, and protocols

- BRN replaces by Fast Internet Bitcoin Relay Engine (FIBRE) (UDP-based relay network)(implements compact block optimization to reduce size of block (i.e. reduce data transmitted) and network latency))
- Falcon (proposal): "cut-through-routing" instead of "store-and-forward"
- Relay networks are not replacements for bitcoin's P2P network. Instead they are overlay networks that provide additional connectivity between nodes with specialized needs. Like freeways are not replacements for rural roads, but rather shortcuts between two points with heavy traffic, you still need small roads to connect to the freeways.

5.5 Network Discovery

- Connecting to known peer: (1) discover at least one existing node and connect to it (random selection – BTC network topology not geographically defined) → (2) establish TCP connection (usually port 8333) → (3) initialize "handshake" by transmitting **version** message containing: **nVersion**, **nLocalServices**, **nTime**, **addrYou**, **addrMe**, **subver**, **BestHeight** → (4) peer looks at nVersion. If compatible, sends **verack**
- Finding peers (Method 1): query DNS using DNS seeds (BCC contains 5) for list of IP addresses of BTC nodes (returns static list of stable listening node addresses or random subset of list of node addresses from crawler or long-running BTC node (custom implementation of BIND)). DNS seeds have diverse implementations and ownership ⇒ high reliability for initial bootstrapping process.
- Finding peers (Method 2): Use command-line argument -seednode to connect to one node, using it as seed. Seed node introduces to new peers, afterwhich client disconnects from seed node
- After one (or more) connections established (part of address propagation and discovery): (1) Send **addr** message to peer → peer fwds **addr** to its neighbours. (2) send **getaddr** request to peer → peer returns **addr** of its peers
- Only handful of nodes required to establish diverse paths (otherwise wasteful and unnecessary), though too little connections unreliable (nodes come and go all the time). As seen above, only one connection needed to bootstrap, afterwhich a node will remember most recent successful peer connections (quick reestablishment after reboot) (if none of former peers respond to connection request, the node can use the seed nodes to bootstrap again)

- If there is no traffic on a connection, nodes will periodically send a message to maintain the connection. If a node has not communicated on a connection for more than 90 minutes, it is assumed to be disconnected and a new peer will be sought. Thus, the network dynamically adjusts to transient nodes and network problems, and can organically grow and shrink as needed without any central control.
- Aside: `getpeerinfo` and `-connect=IPAddress` commands in BCC

5.6 Full Nodes

- full nodes actually full blockchain nodes (early on, all nodes were (BCC is currently)) – in dependant and authoritative maintenance of blockchain (build and verification) \implies verification of any transaction w/o any reliance or trust on other nodes ("full bitcoin experience"). Relies on network to receive updates about new blocks, which it then verifies and incorporates into its local copy of blockchain
- Price of complete independance and freedom from central authority \rightarrow requires more than 100 GB of persistent storage (disk space) to store full blockchain. It also takes two to three days to sync to the network
- Aside: more than 75 percent of full nodes run reference implementation (BCC also called Satoshi client), though alternative implementations of full blockchain bitcoin clients exist

5.7 Exchanging "Inventory"

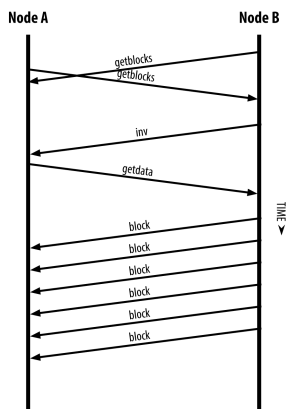


Figure 7: Full-node first action (after connections to peers) – Node synchronization of blockchain – happens any time any node goes offline for any period of time

- Brand-new nodes START w/ genesis block (statically embedded in client software)
- (1) `getblocks` message contains hash (fingerprint) of the top block on their local blockchain (who's longer?)(can identify which blocks the other node needs in order to "catch up") \rightarrow (2) identify the first 500 blocks to share and transmit their hashes (`inv`) \rightarrow (3) node missing blocks will retrieve the rough series of `getdata` messages. The node missing these blocks will then retrieve them, by issuing a series of `getdata` messages (full block data) and identifying the requested blocks using the hashes from the `inv` message (here, load spread across peers. Note: The node keeps track of how many blocks are "in transit" per peer connection, meaning blocks that it has requested but not received, checking that it does not exceed a limit ($MAX_BLOCKS_IN_TRANSIT_PER_PEER$). This way, if it needs a lot of blocks, it will only request new ones as previous requests are fulfilled, allowing the peers to control the pace of updates and not overwhelm the network).

5.8 Simplified Payment Verification (SPV) Nodes

- Most common for space- and power- constrained devices
- Only block header, not contained transactions, downloaded \implies chain of blocks 1000 times smaller (size in kB) AND \implies cannot construct full UTXO set. Verifying that transaction not double-spend done through proof-by-proxy using block *depth*, not height (full nodes construct full database of transactions, and hence full database of UTXO set. Thus, they can validate any transaction directly and confirm UTXO remains unspent) (see tourist analogy) — "A full blockchain node verifies a transaction by checking the entire chain of thousands of blocks below it in order to guarantee that the UTXO is not spent, whereas an SPV node checks how deep the block is buried by a handful of blocks above it."

- Existence of transaction in block proven through merkle path (establishing a link between the transaction and the block that contains it) proof and by validating the POW in the chain of blocks BUT cannot verify that a transaction, such as double spend of the same UTXO, doesn't exist \Rightarrow vulnerability to denial-of-service or double-spend attacks \rightarrow connect randomly to several nodes to increase probability that in contact with at least one honest node \Rightarrow vulnerable to partitioning attacks or Sybil attacks (usually secure enough – balance between resource needs, practicality, and security)
- SPV node synchronization of block headers: process same as full nodes use to retrieve full blocks (see above), except SPV nodes use a getheaders message instead of getblocks (returns headers). The responding peer will send up to 2,000 block headers using a single headers message. SPV nodes also set a filter on the connection to peers, to filter the stream of future blocks and transactions sent by the peers. Any transactions of interest are retrieved using a getdata request. The peer generates a tx message containing the transactions, in response.

6 Bloom Filters

- Problem: request of specific data creates privacy risk (ex: association of bitcoin addresses to user's wallet). Solution: Bloom filters
- Bloom filters (probabilistic search filter) allow SPV node to specify a search pattern for transactions (w/o revealing exactly which addresses, keys, or transactions they are searching for) that can be tuned towards precision or privacy

6.1 How Bloom Filters Work

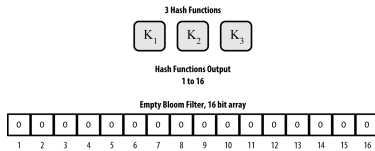


Figure 8: Bloom filter w/ 16 (N)-bit field and 3 (M) hash functions (deterministic)

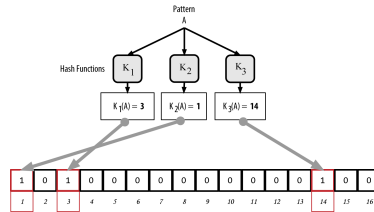


Figure 9: Adding pattern 'A' to data structure (less accurate as more patterns added). The accuracy depends on the number of patterns added versus the size of the bit array (N) and number of hash functions (M).

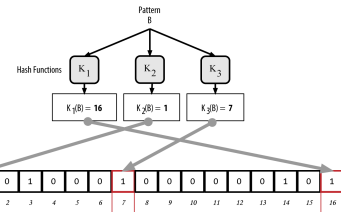


Figure 10: Adding pattern 'B' to bloom filter. Overlapping bits decreases accuracy of filter \Rightarrow bloom filter probabilistic data structure (less accurate as more patterns added). The accuracy depends on the number of patterns added versus the size of the bit array (N) and number of hash functions (M).

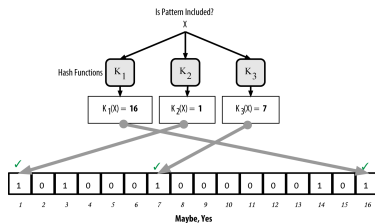


Figure 11

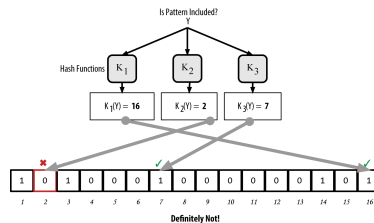


Figure 12

6.2 How SPV Nodes Use Bloom Filters

- SPV node makes list of all addresses, keys, and hashes it is interested in by extracting the public key hash and script hash and transaction IDs from any UTXO controlled by its wallet → add each to bloom filter → send filterload message to peer, containing the bloom filter → bloom filter checked against each incoming transaction. Full node checks several parts of transaction against the bloom filter, looking for a match including: the transaction ID, the data components from the locking scripts of each of the transaction outputs (every key and hash in the script), each of the transaction inputs, and each of the input signature data components (or witness scripts). By checking against all these components, bloom filters can be used to match public key hashes, scripts, *OP_RETURN* values, public keys in signatures, or any future component of a smart contract or complex script.

6.3 SPV Nodes and Privacy

- However, even with bloom filters, an adversary monitoring the traffic of an SPV client or connected to it directly as a node in the P2P network can collect enough information over time to learn the addresses in the wallet of the SPV client.

7 Encrypted and Authenticated Connections

- network communications of a bitcoin node in original implementation NOT encrypted – not an issue for full nodes, major privacy issue for SPV nodes. Solution: encrypted communication through: Tor Transport and P2P Authentication and Encryption

7.1 Tor Transport

- Tor (the Onion Routing network) is network that offers encryption and encapsulation of data through randomized network paths that offer anonymity, untraceability and privacy. See also Tor hidden services offered through Bitcoin Core

7.2 Peer-to-Peer Authentication and Encryption (BIP-150/BIP-151)

- BIP-151 enables negotiated encryption for all communications between two nodes that support BIP-151. BIP-150 offers optional peer authentication that allows nodes to authenticate each other's identity using ECDSA and private keys. BIP-150 requires that prior to authentication the two nodes have established encrypted communications as per BIP-151.
- Additionally, authentication can be used to create networks of trusted bitcoin nodes and prevent Man-in-the-Middle attacks. Finally, P2P encryption, if deployed broadly, would strengthen the resistance of bitcoin to traffic analysis and privacy-eroding surveillance, especially in totalitarian countries where internet use is heavily controlled and monitored.

8 Transaction Pools

- Almost every node maintains a memory pool/mempool/transaction pool (received, verified, relayed but not put on blockchain)
- Some nodes also maintain orphan pool (missing parent referenced in input, when a transaction is added to the transaction pool, the orphan pool is checked for any orphans that reference this transaction's outputs (its children) → validated then added to transaction pool). In light of the newly added transaction, which is no longer an orphan, the process is repeated recursively looking for any further descendants, until no more descendants are found. Through this process, the arrival of a parent transaction triggers a cascade reconstruction of an entire chain of interdependent transactions by re-uniting the orphans with their parents all the way down the chain.

9 The Blockchain

9.1 Introduction

9.2 Structure of a Block

- A block is a container data structure
- Aside: for perspective, average transaction at least 400 bytes, and average block contains more than 1900 transactions

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1-9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

9.3 Block Header

- Contains three sets of metadata: previous block hash, difficulty and timestamp and nonce (relates to mining competition), and the merkle tree root

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

9.4 Block Identifiers: Block Header Hash and Block Height

- Primary identifier: block *header* cryptographic hash (double-SHA256) (32 bytes) (a digital fingerprint (one-way, unique, unambiguous, can be independently derived by anyone))
- Secondary identifier: block height (block height 0 is genesis block) (not unique identifier (specific and invariant, but see forking))
- Neither block header hash or block height is, at any point, stored within the block. They are instead computed/dynamically identified when received by each node. Can be stored as part of block's metadata through indexed database table for faster retrieval of blocks from disk

9.5 The Genesis Block

- Statistically encoded within the bitcoin client software \implies common ancestor for ALL blocks in blockchain, cannot be altered, every node (new nodes incl.) contain it within their local copies of blockchain (so genesis block serves as "secure" starting point)

9.6 Linking Blocks in the Blockchain

- As (full)node receives incoming blocks, first validates block, then links to existing blockchain using the "previous block hash" in block header (Blocks linked in a chain by reference to the previous block header hash) (child-parent relationship)

9.7 Merkle Trees

- Merkle trees (or binary hash trees) are data structures (binary trees containing cryptographic hashes (double-SHA256) that provide an efficient summary of large sets of data (in this case, all the transactions in a block). Used by SPV nodes to verify that a transaction is included in a block

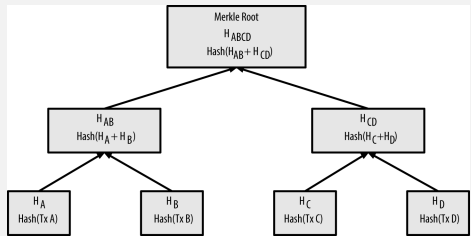


Figure 13: Merkle tree construction

- Bottom-up construction (parent-children). Note: transactions not ever included in merkle tree. Although concatenation occurs to produce 64 bytes from 32 bytes, after hashing, every leaf node contains 32 bytes (incl. root). Recall: Only root stored in block header, since provides summary of all transactions

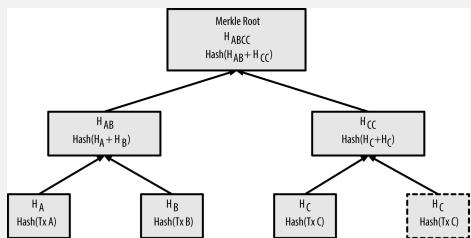


Figure 14: Creating a balanced tree

- Because merkle tree is a binary tree, it needs an even number of leaf nodes, meaning that if there is an odd number of transactions, last transaction hash will be duplicated to create an even number of leaf nodes, producing what's called a **balanced tree**

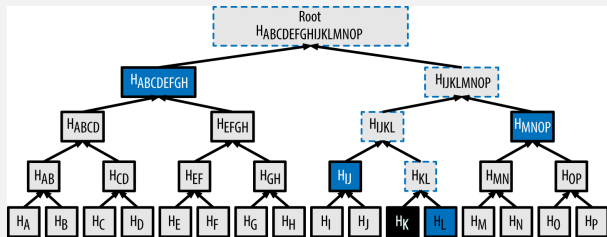


Figure 15: A merkle path or authentication path used to prove the inclusion of a data element

- For a transaction K (or, more appropriately, its transaction hash, and by consequence K?), merkle path is only four 32-byte hashes long (128 bytes total). For any merkle tree, proving requires at most $2 \cdot \log_2(N)$ calculations. Here, need to only produce $\log_2(N)$ 32-byte hashes for proof.

9.8 Merkle Trees and Simplified Payment Verification (SPV)

- Example: SPV node interested in incoming payment to an address contained in its wallet → SPV node establishes bloom filter on its connection to peers → when peer identifies a transaction that matches bloom filter, it sends that block using a merkleblock message (contains block header as well as a merkle path that links the transaction of interest to the merkle root in the block) → SPV node uses merkle path (and root contained in block header) to verify that transaction is included in the block. SPV node also uses the block header to link the block to the rest of the blockchain. The combination of these two links, between the transaction and block, and between the block and blockchain, proves that the transaction is recorded in the blockchain.

9.9 Bitcoin's Test Blockchains

Four bitcoin blockchains: *mainnet*, *testnet*, *segnet*, and *regtest*

- *mainnet*: the "main" bitcoin blockchain, created by Satoshi Nakamoto on January 3rd, 2009, containing the genesis block discussed earlier
- *testnet* ... Bitcoin's Testing Playground: test blockchain, network, and currency (testnet coins) used for testing purposes. Possesses all features of mainnet except for two differences: testnet coins worthless and mining easy (keeping them worthless). These two differences not easy to maintain, because with the increase in use of advanced mining equipment on testnet (GPUs and ASICs), mining difficulty also increases, making testnet coins not worthless. Solution: testnet periodically scrapped starting from a new genesis block, resetting the difficulty (now on testnet3, blockchain actually 20GB or more). Aside: "One good way to run a testnet node is as a virtual machine image (e.g., VirtualBox, Docker, Cloud Server, etc.) dedicated for that purpose." Any software development intended for production use on mainnet should first be tested through testnet (protects developers against monetary loss due to bugs, protects network from unintended behaviour due to bugs).
- *segnet* ... The Segregated Witness Testnet: In 2016, a special-purpose testnet was launched to aid in development and testing of Segregated Witness (aka segwit; see [segwit]). This test blockchain is called segnet and can be joined by running a special version (branch) of Bitcoin Core. Since segwit was added to testnet3, it is no longer necessary to use segnet for testing of segwit features. In the future it is likely we will see other testnet blockchains that are specifically designed to test a single feature or major architectural change, like segnet.
- *Regtest*: The Local Blockchain: Regtest, which stands for "Regression Testing," is a Bitcoin Core feature that allows you to create a local blockchain for testing purposes. Unlike testnet3, which is a public and shared test blockchain, the regtest blockchains are intended to be run as closed systems for local testing. You launch a regtest blockchain from scratch, creating a local genesis block. You may add other nodes to the network, or run it with a single node only to test the Bitcoin Core software.

9.10 Using Test Blockchains for Development

Bitcoin's various blockchains (regtest, segnet, testnet3, mainnet) offer a range of testing environments for bitcoin development. Use the test blockchains whether you are developing for Bitcoin Core, or another full-node consensus client; an application such as a wallet, exchange, ecommerce site; or even developing novel smart contracts and complex scripts.

You can use the test blockchains to establish a development pipeline. Test your code locally on a regtest as you develop it. Once you are ready to try it on a public network, switch to testnet to expose your code to a more dynamic environment with more diversity of code and applications. Finally, once you are confident your code works as expected, switch to mainnet to deploy it in production. As you make changes, improvements, bug fixes, etc., start the pipeline again, deploying each change first on regtest, then on testnet, and finally into production.

10 Mining and Consensus

10.1 Introduction

10.2 Decentralized Consensus

Question: How is it possible for an authoritative, trusted, public, and global ledger/record of ownership (i.e. the blockchain) to occur without a central authority (or w/o any trust at all, really)? **Answer:** Decentralized, *emergent consensus* from the interplay of four processes that occur independently on nodes across the network:

- Independent verification of each transaction, by every full node, based on a comprehensive list of criteria
- Independent aggregation of those transactions into new blocks by mining nodes, coupled with demonstrated computation through a Proof-of-Work algorithm

- Independent verification of the new blocks by every node and assembly into a chain
- Independent selection, by every node, of the chain with the most cumulative computation demonstrated through Proof-of-Work

10.3 Independent Verification of Transactions

Every node independently verifies propagated transactions before propagating it further so as to build a pool of valid but unconfirmed transactions called *transaction pool*, *memory pool*, or *mempool*. The (dynamic) criteria for transaction validation (otherwise discarded): syntax and data structure correct, neither I/O empty, $100 \leq \text{size (bytes)} < \text{MAX_BLOCK_SIZE}$, dust threshold $<$ each output (as well as total) $<$ 21m coins, none of inputs have hash=0, N=-1 (coinbase transactions should not be relayed),

- The transaction's syntax and data structure must be correct.
- Neither lists of inputs or outputs are empty.
- The transaction size in bytes is less than MAX_BLOCK_SIZE .
- Each output value, as well as the total, must be within the allowed range of values (less than 21m coins, more than the dust threshold).
- None of the inputs have hash=0, N=-1 (coinbase transactions should not be relayed).
- nLocktime is equal to INT_MAX , or nLocktime and nSequence values are satisfied according to MedianTimePast.
- The transaction size in bytes is greater than or equal to 100.
- The number of signature operations (SIGOPS) contained in the transaction is less than the signature operation limit.
- The unlocking script (scriptSig) can only push numbers on the stack, and the locking script (scriptPubkey) must match IsStandard forms (this rejects "nonstandard" transactions).
- A matching transaction in the pool, or in a block in the main branch, must exist.
- For each input, if the referenced output exists in any other transaction in the pool, the transaction must be rejected.
- For each input, look in the main branch and the transaction pool to find the referenced output transaction. If the output transaction is missing for any input, this will be an orphan transaction. Add to the orphan transactions pool, if a matching transaction is not already in the pool.
- For each input, if the referenced output transaction is a coinbase output, it must have at least COINBASE_MATURITY (100) confirmations.
- For each input, the referenced output must exist and cannot already be spent.
- Using the referenced output transactions to get input values, check that each input value, as well as the sum, are in the allowed range of values (less than 21m coins, more than 0).
- Reject if the sum of input values is less than sum of output values.
- Reject if transaction fee would be too low (minRelayTxFee) to get into an empty block.
- The unlocking scripts for each input must validate against the corresponding output locking scripts.

10.4 Mining Nodes

- Recall: not all miners are full nodes (see mining pools)
- Like every other node, miners receive and propagate unconfirmed transactions on the network, but, in addition to this, aggregate these transactions into new blocks
- Like every other node, miners listen for blocks propagated on network, with the added meaning that mined block is simultaneously end of last round and beginning of new round of mining

10.5 Aggregating Transactions into Blocks

Recall: Nodes (including miners) collect, validate (if successful, added and kept in memory pool until on blockchain. Memory pool not global), and propagate transactions. Miners also aggregate transactions into *candidate block*. Miner simultaneously listens for transactions, tries to mine new block, and listens for blocks discovered by other nodes. While doing this, it was also collecting transactions in preparation for the next block. When block received, validated, compared against mempool transactions for removal from mempool. Empty candidate block immediately constructed.

10.5.1 The Coinbase Transaction

First transaction in block is coinbase transaction. Contains reward for mining effort (coinbase reward (halved every 210,000 blocks) and transaction fees of all transactions included in block (implicitly derived between input and output)). Transaction created by miner, payment to own wallet. Transaction does not consume UTXO as input, but has only one input (the coinbase) which creates bitcoin from nothing, and one output to miner's own address

10.5.2 Coinbase Rewards and Fees

Constructing coinbase transaction: (1) $\text{Total Fees} = \text{Sum}(\text{Inputs}) - \text{Sum}(\text{Outputs})$, (2) calculate reward from block height

Bindary right shift operator used because more efficient than repeated divisions. As a final step, the coinbase reward ($n\text{Subsidy}$) is added to the transaction fees ($n\text{Fees}$), and the sum is returned. Coinbase transaction then constructed with these calculations done. Question: If Jing's mining node writes the coinbase transaction, what stops Jing from "rewarding" himself 100 or 1000 bitcoin? The answer is that an incorrect reward would result in the block being deemed invalid by everyone else, wasting Jing's electricity used for Proof-of-Work. Jing only gets to spend the reward if the block is accepted by everyone.

10.5.3 Structure of the Coinbase Transaction

In a coinbase transaction, the first two fields are set to values that do not represent a UTXO reference.

10.5.4 Coinbase Data

Except for first few bytes (BIP-34, version 2 blocks), coinbase data is arbitrary. Currently, miners use the coinbase data to include extra nonce values and strings identifying the mining pool. Beginning of field must contain block height index as a script "push" operation. See example coinbase data: 03443b0403858402062f503253482f. 03: instructs script execution engine to push the next three bytes onto the script stack. Next three bytes (0x443b04) are block height encoded in little-endian format (backward, least-significant byte first). Next few (0385840206) are used to encode an extra nonce, or random value, used to find a suitable Proof-of-Work solution. The final part (2f503253482f) is ASCII-encoded string /P2SH/ (versus p2sh/CHV) which indicates that mining node supports P2SH improvement as defined by BIP-16 not BIP-17 (vote: BIP-16 elected as winner).

Table 4: Merkle tree efficiency (effect more pronounced as number of transactions increases, because the base-2 logarithm of the number of transactions increases much more slowly)

Number of transaction	Approx. size of block	Path size (hashes)	Path size (bytes)
16 transactions	4 kilobytes	4 hashes	128 bytes
512 transactions	128 kilobytes	9 hashes	288 bytes
2048 transactions	512 kilobytes	11 hashes	352 bytes
65,535 transactions	16 megabytes	16 hashes	512 bytes

Figure 16: Caption

Table 5: My caption

Size	Field	Description
32 bytes	Transaction Hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent, first one is 0
1–9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking script
4 bytes	Sequence Number	Currently disabled Tx-replacement feature, set to 0xFFFFFFFF

Table 6: My caption

Size	Field	Description
32 bytes	Transaction Hash	All bits are zero: Not a transaction hash reference
4 bytes	Output Index	All bits are ones: 0xFFFFFFFF
1–9 bytes (VarInt)	Coinbase Data Size	Length of the coinbase data, from 2 to 100 bytes
Variable	Coinbase Data	Arbitrary data used for extra nonce and mining tags. In v2 blocks; must be
4 bytes	Sequence Number	Currently disabled Tx-replacement feature, set to 0xFFFFFFFF

10.6 Constructing the Block Header

Recall: 6 fields defining the structure of block header. Added points: At the time that block 277,316 was mined, the version number describing the block structure is version 2, which is encoded in little-endian format in 4 bytes as 0x02000000. For previous block hash, by selecting the specific parent block, indicated by the Previous Block Hash field in the candidate block header, Jing is committing his mining power to extending the chain that ends in that specific block. In essence, this is how Jing "votes" with his mining power for the longest-difficulty valid chain. Jing's node then fills in the target, which defines the required Proof-of-Work to make this a valid block. The target is stored in the block as a "target bits" metric, which is a mantissa-exponent encoding of the target. The encoding has a 1-byte exponent, followed by a 3-byte mantissa (coefficient). In block 277,316, for example, the target bits value is 0x1903a30c. The first part 0x19 is a hexadecimal exponent, while the next part, 0x03a30c, is the coefficient. The final field is the nonce, which is initialized to zero. With all the other fields filled, the block header is now complete and the process of mining can begin. The goal is now to find a value for the nonce that results in a block header hash that is less than the target. The mining node will need to test billions or trillions of nonce values before a nonce is found that satisfies the requirement.

10.7 Mining the Block

10.7.1 Proof-of-Work Algorithm

- Properties of a hash algorithm: arbitrary-length input \rightarrow fixed-length deterministic output (digital fingerprint of input). Deterministic \implies easily verifiable (one hash for POW)
- Key characteristic of a *cryptographic* hash algorithm is that it is *collision-resistant*. Corollary: Virtually impossible to select an input in such a way as to produce a desired output, other than trying random inputs
- SHA256 cryptographic hash algorithm used for proof-of-work. In textbook example, a variable integer (i.e. the *nonce*) is appended to the phrase "I am Satoshi Nakamoto" to vary the output (called the *hash* or *digest*). Again, this works because the output depends on every part of the input, and so any change results in a completely different hash
- *Target* threshold serves as a challenge, which, in numerical terms means finding a hash value less than a certain number (i.e. the target), which translates to a hash value with a certain number of leading zeroes. Adjusting the target means changing the difficulty (target and difficulty are inversely related)
- Since SHA256 is deterministic, the probability of an outcome based on the difficulty imposed by the target can be calculated before-hand using statistics. The successful result is thus a proof-of-work.
- In Bitcoin's Proof-of-Work, miner constructs a candidate block filled with transactions, miner then calculates hash of block header to see if smaller than current target. If not, miner modifies nonce (usually by incrementing by one) and tries again. Current difficulty is set in bitcoin network (in bits, how many of the leading bits must be zero) so that at total network processing power (in hashes per second), block is mined every 10 minutes on average. Every increase in bit approximately doubles time it takes to find a solution. Again, increasing difficulty decreases the acceptable range of hashes.

10.7.2 Target Representation

- Target in "target bits" or just "bits" notation (coefficient/exponent format): $target = coefficient * 2^{(8 * (exponent - 3))}$. In example 0x1903a30c, first 2 hex digits exponent, 6 remaining hex digits coefficient. Recall: result output and must be displayed as 64-digit field (hex) or 256-digit field (binary). Question: How to calculate mining time from hash rate and difficulty?

10.7.3 Retargeting to Adjust Difficulty

- Recall: Target determines difficulty which determines how long it takes to find a solution to the Proof-of-Work algorithm
- Why is the difficulty adjustable? The target is set so that the current mining power (of entire network) will result in a 10-minute block interval (average). Readjusting account for increase in the number of participants and computer power. Block interval underpins frequency of currency issuance and speed of transaction settlement.
- Who adjusts it? Sticking to theme of a completely decentralized network, re-targeting occurs automatically and on every node independently every 2,016 blocks (happens every 2,016 blocks, based on previous 2,015 blocks due to off-by-one error in original Bitcoin Core client, resulting in retargeting bias towards higher difficulty by 0.05 percent)
- How is it adjusted? Equation summarized as: $NewTarget = OldTarget * (ActualTimeofLast2016Blocks / 20160min)$ (see `CalculateNextWorkRequired()` in `pow.cpp` (parameters `Interval` (2,016 blocks) and `TargetTimespan` (two weeks as 1,209,600 seconds) defined in `chainparams.cpp`). To avoid extreme volatility in the difficulty, adjustments must be less than a factor of four. If greater, readjusted by a factor of four and no more. This means that several 2,016 block cycles might be required to (near) remove discrepancies between hashing power and difficulty as imbalances can persist through near-future block cycles.
- Note: target independant of the number of transactions or the value of transactions (recall: the hash of the block header is being made). Therefore, hashing power and electricity expended also independent of these two. This means that Bitcoin can scale up, achieve broader adoption, and remain secure without any increase in hashing power from today's levels. The increase in hashing power represents market forces as new miners enter the market to compete for the reward. As long as enough hashing power is under the control of miners acting honestly in pursuit of the reward, it is enough to prevent "takeover" attacks and, therefore, it is enough to secure bitcoin. The difficulty of mining is closely related to the cost of electricity and the exchange rate of bitcoin vis-a-vis the currency used to pay for electricity. High-performance mining systems are about as efficient as possible with the current generation of silicon fabrication, converting electricity into hashing computation at the highest rate possible. The primary influence on the mining market is the price of one kilowatt-hour of electricity in bitcoin, because that determines the profitability of mining and therefore the incentives to enter or exit the mining market.

10.8 Successfully Mining the Block

- After candidate block prepared for mining, mining node sends block header to mining hardware (aside: Jing has several hardware mining rigs with application-specific integrated circuits, where hundreds of thousands of integrated circuits run the SHA256 algorithm in parallel at incredible speeds. Many of these specialized machines are connected to his mining node over USB or a local area network), which starts testing nonce values. Nonce is only 32 bits, so after exhausting all nonce possibilities, mining hardware changes the block header (adjusting the coinbase extra nonce space or timestamp) and resets the nonce counter, testing new combinations. If found, hardware machine sends solution back to mining node. Hash of nonce inserted into block header should produce result less than target. Mining node immediately transmits block to all its peers, who in turn receive, validate, propagate. Once validated, each node adds to own copy of blockchain and mining node immediately abandon their efforts to mine for block at same height, and start computing next block in chain, using received block as parent. By building on top of Jing's newly discovered block, the other miners are essentially "voting" with their mining power and endorsing Jing's block and the chain it extends.

10.9 Validating a New Block

- Independant validation by every node using the same rules ensures that only valid blocks propagated and ensures that miners cannot cheat (disincentive: cost of electricity without compensation in the form of

reward). Thus, independent validation key component of decentralized consensus. Criteria for validation as set by Bitcoin Core:

- The block data structure is syntactically valid
- The block header hash is less than the target (enforces the Proof-of-Work)
- The block timestamp is less than two hours in the future (allowing for time errors)
- The block size is within acceptable limits
- The first transaction (and only the first) is a coinbase transaction
- All transactions within the block are valid using the transaction checklist discussed in Independent Verification of Transactions

When precisely can winning miner collect reward? After all or a reasonable amount of nodes validate block?

10.10 Assembling and Selecting Chains of Blocks

- Nodes maintain three sets of blocks: main chain, secondary chains, orphans. Invalid blocks rejected and not included anywhere.
 - At any time, main chain is one with most cumulative Proof-of-Work associated with it (usually means chain with most blocks in it, unless there are two equal-length chains and one has more Proof-of-Work)
 - If valid block received and no parent found in existing chains, that block considered an orphan (temporally placed in orphan block pool). Orphan blocks usually occur when two blocks that were mined within a short time of each other are received in reverse order (child before parent). How could this happen when mining interval is approximately 10 minutes?

10.10.1 Blockchain Forks

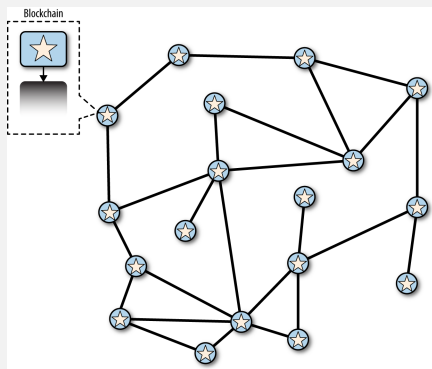


Figure 17: Before the fork – all nodes have the same perspective

- Note: blockchain is a decentralized data structure, meaning that each node has its own perspective of the global blockchain. As each node receives blocks from its neighbours, it updates its own copy of the blockchain. Blockchain forks occur naturally due to transmission delays in the global network, but as long as each node selects the greatest-cumulative-work chain, the global bitcoin network eventually converges to a consistent state.
- For illustration purposes, each node contains a shape that it believes is currently the tip of the main chain

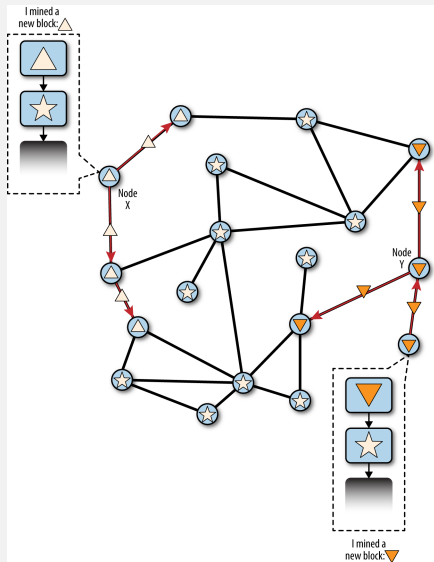


Figure 18: Visualization of a blockchain fork event: two blocks found simultaneously

- Both blocks valid. Both blocks extend same parent. Both blocks likely contain most of the same transactions, with only perhaps a few differences in the order of transactions.

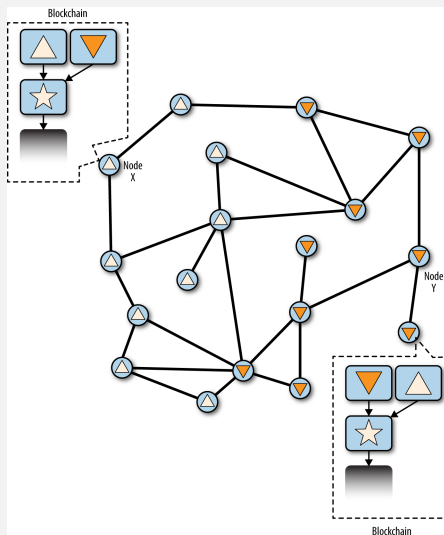


Figure 19: Visualization of a blockchain fork event: two blocks propagate, splitting the network

- Both blocks valid. Both blocks extend same parent. Both blocks likely contain most of the same transactions, with only perhaps a few differences in the order of transactions.
- Block that extends the main chain is whichever was received first. Whichever block that was received second forms a secondary chain.

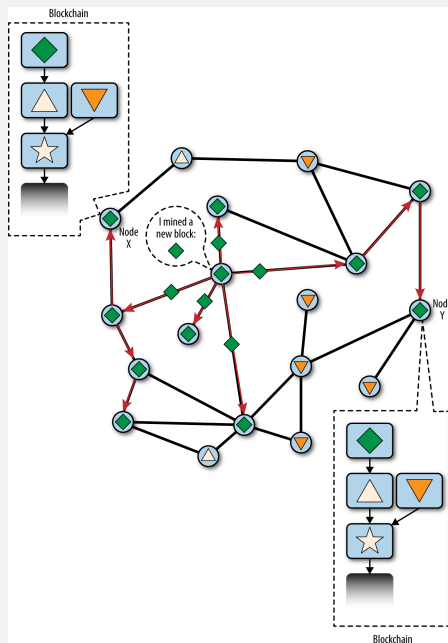


Figure 20: Visualization of a blockchain fork event: a new block extends one fork, reconverging the network

- Nodes are voting with their hashing power in support and in dedication of and to the chain that they have elected as their main chain.
- Forks almost always resolved in one block (forks extending to two blocks very unlikely)

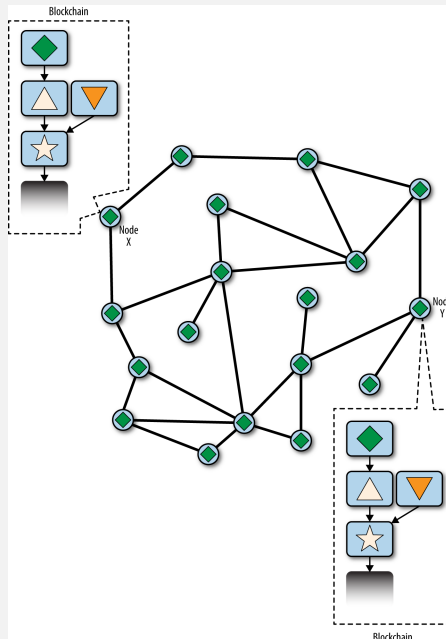


Figure 21: Visualization of a blockchain fork event: the network reconverges on a new longest chain

- Chain reconvergence involves nodes being forced to revise their view of the blockchain to incorporate the new evidence of a longer chain. Upside-down triangle main chain now becomes secondary chain, and main chain switches over to diamond. Work done to extend previous main chain stopped, because their candidate block is an "orphan", as its parent "upside-down-triangle" is no longer on the longest chain? This is a chain reconvergence, because those nodes are forced to revise their view of the blockchain to incorporate the new evidence of a longer chain. Any miners working on extending the chain star-upside-down-triangle will now stop that work because their candidate block is an "orphan," as its parent "upside-down-triangle" is no longer on the longest chain. The transactions within "upside-down-triangle" that are not within "triangle" are re-inserted in the mempool for inclusion in the next block to become a part of the main chain. The entire network reconverges on a single blockchain star-triangle-rhombus, with "rhombus" as the last block in the chain. All miners immediately start working on candidate blocks that reference "rhombus" as their parent to extend the star-triangle-rhombus chain.

Bitcoin's block interval of 10 minutes is a design compromise between fast confirmation times (settlement of transactions) and the probability of a fork. A faster block time would make transactions clear faster but lead to more frequent blockchain forks, whereas a slower block time would decrease the number of forks but make settlement slower.

10.11 Mining and the Hashing Race

- Total hashing power of bitcoin network has grown exponentially every year due to a larger network of participants and technological developments, most notably CPU to GPU mining, FPGAs, and ASICs. The difficulty has been met to match this exponential increase. Aside: We are coming to the forefront of Moore's law, so "here are no more giant leaps left in bitcoin mining". However, exponential increase expected to increase due to higher density data centers.

10.11.1 The Extra Nonce Solution

- With the increase in difficulty, the nonce was iterated through its entirety before a valid solution was found. Need an extra source of change in block header → change timestamp to account for elapsed time. With increase in hashing power, this solution could cause block to be invalid → use coinbase transaction as extra nonce space (also changes merkle root) (8 bytes extra nonce, 4 bytes standard). If iterate through all, change timestamp (could further expand in coinbase in future)

10.11.2 Mining Pools

- Very unlikely for solo miners to offset electricity and hardware costs (aside: commercial systems) (answer to previous question, probability worked out with: $P = (\text{miner's hash rate} / \text{global hash rate at beginning of time period}) * (\text{number blocks in } x \text{ years}) = 0.98$ – however, uncertainty still involved, plus difficulty bound to increase roughly exponentially, so mining rig obsolete in around one year's time). Mining pool splits reward but uncertainty reduced (frequent payout means lower risk). Financial payouts of mining pool "only make sense at very low very electricity cost and only at very large scale"
- Mining pools coordinate over specialized pool-mining protocols. Efforts of each miner synchronized and shared with rest of pool, thereby earning share of reward (based on share of work done, measured and verified using Proof-of-Work algorithm). Successful blocks pay reward to a pool bitcoin address, rather than individual miners (pool server provides percentage fee of rewards for providing pool-mining service) ("The pool server will periodically make payments to the miners' bitcoin addresses, once their share of the rewards has reached a certain threshold."). Pool sets a higher target (lower difficulty) than network target that serves as incentive for smaller pool miners to participate in pool (success gives proof that node has done the hashing work to find that result). More importantly, the work to find shares contributes, in a statistically measurable way, to the overall effort to find a hash lower than the bitcoin network's target.

Managed Pools

- Managed pools means a company or individual (the *pool operator*) runs the pool server (specialized software and pool-mining protocol for coordination). Pool operator charges percentage fee of earnings. Why? Pool operator connects to full node for full blockchain access for transaction and block validation on behalf of miners (good for miners, lots of RAM and disk for pool operator). Plus, maintenance, monitoring, and upgrade of software running on the full node (otherwise hurt profitability of miners). Miners connect to server using a mining protocol such as Stratum (STM) or GetBlockTemplate (GBT). Pool server constructs full candidate block, sends header template to miners.

Peer-to-peer mining pool (P2Pool)

- Problem: pool miner can cheat (contain double-spend transactions or invalidate blocks) and represents a centralized, single-point-of-failure (server can be downed or slowed by denial of service attacks). Solution: decentralize functions of a pool server through a *share chain* (instead of pool server keeping track of shares and rewards, all pool miners keep track of all shares using a decentralized consensus mechanism like bitcoin's blockchain consensus mechanism, using a blockchain at lower difficulty). Disadvantage: pool miners must have enough disk space, memory, and internet bandwidth to support a full bitcoin node and the P2Pool node software (?) and still vulnerable to 51 percent consensus attacks against share chain Local P2Pool node, which sends block templates to miners (miners construct own candidate blocks like solo miners, and mine collaboratively on share chain)

10.12 Consensus Attacks

- Consensus mechanism depends on having the majority of miners acting honestly out of self-interest. Consensus attacks only affect future (denial-of-service attacks) or recent past (tens of blocks) (older blocks practically immutable). Consensus attacks do not affect security of private key and signing algorithm (ECDSA). Consensus attacks cannot steal bitcoin, spend bitcoin without signatures, redirect bitcoin, or otherwise change past transactions or ownership records.
- Example consensus attack – 51 percent attack: (1) Fork to "double-spend" OWN transactions, for which the attacker can produce a valid signature. Invalidates transactions in previous blocks, causing attacker to get an irreversible exchange payment or produce without paying for it. Solution: for high-value purchases, wait at least 6 confirmations, or the merchant should use an escrow multisignature account, again waiting for several confirmations after the escrow account is funded. For high-value items, this is still convenient and efficient to conventional payments? Low-value items not worth it, because the risk of a double-spend

on a cup of coffee is low in comparison to the convenience of rapid customer service. This is similar to the practice of coffee shops that accept credit card payments without a signature for amounts below 25 dollars, because the risk of a credit-card chargeback is low while the cost of delaying the transaction to obtain a signature is comparatively larger. (2) Sustained denial-of-service against specific bitcoin addresses by ignoring those transactions or forking and remining if included on a block mined by another miner. Note: ONLY pool server can have malicious intent (others mine with automated miners and cannot monitor every transaction or block). Note: 51 percent only threshold in which success of attack becomes more likely than not. "Security research groups have used statistical modeling to claim that various types of consensus attacks are possible with as little as 30 percent of the hashing power". More hashing power, longer the fork you can deliberately create, the more blocks in the recent past can invalidate, or the more blocks in the future can control.

- Exponential increase in total network hashing power makes consensus attacks more difficult (practically impossible for a single miner), but possible for mining pools due to the centralization of control to the pool operator (construction of candidate blocks and which transactions included controlled). In addition, non-profit attacks aimed at disrupting bitcoin network possible for well-funded, most likely state-sponsored attackers. Alternatively, a well-funded attacker could attack bitcoin's consensus by simultaneously amassing mining hardware, compromising pool operators, and attacking other pools with denial-of-service. Undoubtedly, a serious consensus attack would erode confidence in bitcoin in the short term, possibly causing a significant price decline. However, the bitcoin network and software are constantly evolving, so consensus attacks would be met with immediate countermeasures by the bitcoin community, making bitcoin more robust (public project).

10.13 Changing the Consensus Rules

- Consensus rules determine the validity of transactions and blocks. Local perspectives converge to a network-wide blockchain. Can be upgraded but difficult.

10.13.1 Hard Forks

- Definition: a fork due to a change in the consensus rules (network DOES NOT re-converge, two chains evolve independently). Change in consensus rules occur either due to a bug in the consensus rules or a deliberate change in the implementation of the consensus rules (software fork). Hard fork not "forward compatible" (invalid transaction AND/OR block) ("valid" blocks that reference invalid block as parent is orphan block and not processed). Coordination required! Network partitioned/split

10.13.2 Hard Forks: Software, Network, Mining, and Chain

- Software forks (open source) precede deliberate hard forks. Software forks become hard forks only after new implementation developed, adopted, and launched (necessary precondition, but not itself sufficient) ("for a hard fork to occur, the competing implementation must be adopted and the new rules activated, by miners, wallets, and intermediary nodes"). Software forks that do not change the consensus rules (barring a bug) can coexist and inter-operate on network without causing hard fork.
- "Consensus rules may differ in obvious and explicit ways, in the validation of transactions or blocks. The rules may also differ in more subtle ways, in the implementation of the consensus rules as they apply to bitcoin scripts or cryptographic primitives such as digital signatures. Finally, the consensus rules may differ in unanticipated ways because of implicit consensus constraints imposed by system limitations or implementation details. An example of the latter was seen in the unanticipated hard fork during the upgrade of Bitcoin Core 0.7 to 0.8, which was caused by a limitation in the Berkley DB implementation used to store blocks."
- Conceptually, we can think of a hard fork as developing in four stages: a software fork, a network fork, a mining fork, and a chain fork. When this forked implementation is deployed in the network, a certain

percentage of miners, wallet users, and intermediate nodes may adopt and run this implementation. A resulting fork will depend upon whether the new consensus rules apply to blocks, transactions, or some other aspect of the system. If the new consensus rules pertain to transactions, then a wallet creating a transaction under the new rules may precipitate a network fork, followed by a hard fork when the transaction is mined into a block. If the new rules pertain to blocks, then the hard fork process will begin when a block is mined under the new rules.

10.13.3 Diverging Miners and Difficulty

- Scenario: Hard fork causing split in mining power across network along with fork occurring immediately after a re-targeting period (recall: difficulty readjusts to produce 10 minute after 2016 blocks)

10.13.4 Contentious Hard Forks

- We are at the dawn of this new frontier called consensus software development (new development tools, practices, methodologies, and communities will emerge) (aside: open source software development)
- Hard fork risky because they risk partitioning system, unless you have near-unanimous consent. Contentious and controversial issue, with people on left, right, and middle sides.

10.13.5 Soft Forks

- Soft fork not actually a fork. Forward-compatible change to consensus rule. Can only be used to constrain consensus rules, not expand them (new rules can only limit what is valid) (otherwise hard fork triggered). Soft forks don't require all nodes to upgrade or force non-upgraded nodes out of consensus. "In order to be forward compatible, transactions and blocks created under the new rules must be valid under the old rules too, but not vice versa."

Soft forks redefining NOP opcodes

- NOP1-NOP10 in Bitcoin Script null-potent operators. Soft fork to give them new meaning (NOP code for any old client) (example: BIP-65, NOP2 interpreted as `OP_CHECKLOCKTIMEVERIFY`)

Other ways to soft fork upgrade

- Segwit (initially envisioned as a hard fork, as it modified a fundamental structure: transactions): an architectural change to the structure of a transaction, which moves the unlocking script (witness) from inside the transaction to an external data structure (segregating it). Was able to be introduced as a soft fork: the mechanism used for this is a modification of the locking script of UTXO created under segwit rules, such that unmodified clients see the locking script as redeemable with any unlocking script whatsoever.

10.13.6 Criticisms of Soft Forks

- NOP opcodes uncontroversial (explicit goal, non-disruptive)
- Common criticism: Technical debt, validation relaxation (applies to NOP upgrades as well), irreversible upgrades

10.14 Soft Fork Signalling with Block Version

10.14.1 BIP-34 Signaling and Activation

- Block version number used to signal readiness (prior to: "1", convention not enforced by consensus). BIP-34 changed consensus rule where coinbase field in coinbase transaction input contained the block height.
- Two-step activation mechanism (based on rolling window of 1000 blocks) (prior to these steps, individual miners who set block version to "2" voice support, but don't have to take any action):

- 75 percent of last 1000: version 2 blocks must contain block height (otherwise invalid), version 1 don't have to (old and new consensus rules coexist during this period)
- 95 percent: all must comply. Version 1 blocks no longer valid
- Used twice more to signal and activate soft forks with BIP-66 and BIP-65. After BIP-65 activation, BIP-9 became standard
- Limitations: Coordination and agreement on sequencing/prioritization of soft fork proposals required, since integer value of block version number meant that only one could be activated at a time; because the block version was incremented, the mechanism didn't offer a straightforward way to reject a change and then propose a different one. If old clients were still running, they could mistake signaling for a new change as signaling for the previously rejected change; Each new change irrevocably reduced the available block versions for future changes

10.14.2 BIP-9 Signaling and Activation

- "BIP-9 interprets the block version as a bit field instead of an integer"; Proposed changes are identified by a data structure that contains the following fields: name, bit (0 through 28 (versions 1 to 4 already used) → independently and simultaneously signal readiness on 29 different proposals), starttime (based on MTP), endtime (based on MTP). TIMEOUT set in proposal. After TIMEOUT, whether proposal activated or rejected, bits can be recycled, even for same proposal. "In short, we should not expect to see reuse until all 29 bits have been used once"
- Difference: for every retarget period, if the sum of blocks signaling for a proposal exceeds 95 percent (1916 of 2016), the proposal will be activated one retarget period later.

10.15 Consensus Software Development

- Evolving, continuous discussion. By its nature, decentralized system where power diffused and unilateral decisions cannot be made makes coordination vital and consensus changes more difficult (weakness? author sees it as one of its greatest strengths). The bar for consensus is also high –Status quo is the stable state of this system with only a few changes possible if there is strong consensus by a very large majority. The 95 percent threshold for soft forks is reflective of this reality. Even if 51 percent of miners can theoretically change the rules by simple majority (51 percent), they are constrained by the consent of the other constituencies. If they act unilaterally, the rest of the participants may simply refuse to follow them, keeping the economic activity on a minority chain. Without economic activity (transactions, merchants, wallets, exchanges), the miners will be mining a worthless coin with empty blocks. Hard forks and soft forks aren't perfect – the one constant characteristic of consensus software development is that change is difficult and consensus forces compromise.

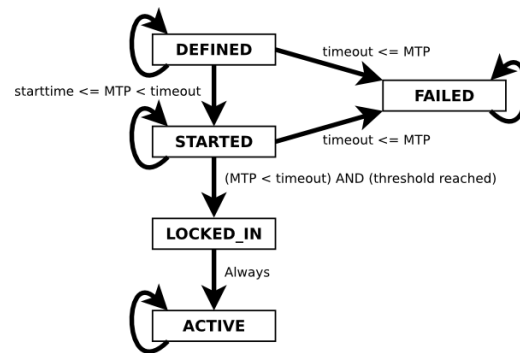


Figure 22: BIP-9 state transition diagram