Bitcoin: 10,000 foot view

- New bitcoins are “created” every ~10 min, owned by “miner” (more on this later)
- Thereafter, just keep record of transfers
  - e.g., Alice pays Bob 1 BTC
- Basic protocol:
  - Alice signs transaction: $\text{txn} = \text{Sign}_{\text{Alice}}(\text{BTC}, \text{PK}_{\text{Bob}})$
  - Alice shows transaction to others...

Problem: Equivocation!

Can Alice “pay” both Bob and Charlie with same bitcoin?

(Known as “double spending”)

How traditional e-cash handled problem

- When Alice pays Bob with a coin, Bob validates that coin hasn’t been spend with trusted third party
- Introduced “blind signatures” and “zero-knowledge protocols” so bank can’t link withdrawals and deposits
How traditional e-cash handled problem

- When Alice pays Bob with a coin, Bob validates that coin hasn’t been spend with trusted third party

Bank maintains linearizable log of transactions

Problem: Equivocation!

Goal: No double-spending in decentralized environment

Approach: Make transaction log
1. public
2. append-only
3. strongly consistent

Bitcoin: 10,000 foot view

- Public
  - Transactions are signed: \( \text{txn} = \text{Sign}_{\text{Alice}}(\text{BTC, PK}_{\text{Bob}}) \)
  - All transactions are sent to all network participants

- No equivocation: Log append-only and consistent
  - All transactions part of a hash chain
  - Consensus on set/order of operations in hash chain

Intro to crypto in 5 minutes
Public-Key Cryptography

- Each party has (public key, private key)
- Alice’s public key PK
  - Known by anybody
  - Bob uses PK to encrypt messages to Alice
  - Bob uses PK to verify signatures from Alice
- Alice’s private/secret key: sk
  - Known only by Alice
  - Alice uses sk to decrypt ciphertexts sent to her
  - Alice uses sk to generate new signatures on messages

Cryptography Hash Functions I

- Take message $m$ of arbitrary length and produces fixed-size (short) number $H(m)$
- One-way function
  - Efficient: Easy to compute $H(m)$
  - Hiding property: Hard to find an $m$, given $H(m)$
    - Assumes “m” has sufficient entropy, not just (“heads”, “tails”)
    - Random: Often assumes for output to “look” random

Cryptography Hash Functions II

- Collisions exist: $|\text{possible inputs}| >> |\text{possible outputs}|$
  … but hard to find
- Collision resistance:
  - Strong resistance: Find any $m \neq m'$ such that $H(m) = H(m')$
  - Weak resistance: Given $m$, find $m'$ such that $H(m) = H(m')$
  - For 160-bit hash (SHA-1)
    - Finding any collision is birthday paradox: $2^{(160/2)} = 2^{80}$
    - Finding specific collision requires $2^{160}$
Tamper-evident logging

- Hash chain creates “tamper-evident” log of txns
- Security based on collision-resistance of hash function
  - Given \( m \) and \( h = \text{hash}(m) \), difficult to find \( m' \) such that \( h = \text{hash}(m') \) and \( m \neq m' \)

Blockchain: Append-only hash chain

Problem remains: forking
**Goal: Consensus**

- Recall Byzantine fault-tolerant protocols to achieve consensus of replicated log
  - Requires: \( n >= 3f + 1 \) nodes, at most \( f \) faulty

- Problem
  - Communication complexity is \( n^2 \)
  - Requires **strong view** of network participants

**Consensus susceptible to Sybils**

- All consensus protocols based on membership…
  - ... assume independent failures ...
  - ... which implies strong notion of identity

- “Sybil attack” (p2p literature ~2002)
  - Idea: one entity can create many “identities” in system
  - Typical defense: 1 IP address = 1 identity
  - Problem: IP addresses aren’t difficult / expensive to get, esp. in world of botnets & cloud services

**Consensus based on “work”**

- Rather than “count” IP addresses, bitcoin “counts” the amount of CPU time / electricity that is expended

  “The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.”
  - Satoshi Nakamoto

- Proof-of-work: Cryptographic “proof” that certain amount of CPU work was performed

**Key idea: Chain length requires work**

- Generating a new block requires “proof of work”
- “Correct” nodes accept longest chain
- Creating fork requires rate of malicious work >> rate of correct
  - So, the older the block, the “safer” it is from being deleted
Use hashing to determine work!

- Recall hash functions are one-way / collision resistant
  - Given $h$, hard to find $m$ such that $h = hash(m)$

- But what about finding partial collision?
  - $m$ whose hash has most significant bit = 0?
  - $m$ whose hash has most significant bit = 00?
  - Assuming output is randomly distributed, complexity grows exponentially with # bits to match

Bitcoin proof of work

- Find nonce such that
  $$hash(\text{nonce} \ || \ \text{prev_hash} \ || \ \text{block data}) < \text{target}$$
i.e., hash has certain number of leading 0’s

What about changes in total system hashing rate?

- Target is recalculated every 2 weeks
- Goal: One new block every 10 minutes

Historical hash rate trends of bitcoin

Currently: 2 Exahash/s
2 x 10^{18}

Tech: CPU → GPU → FPGA → ASICs

Why consume all this energy?

- Creating a new block creates bitcoin!
  - Initially 50 BTC, decreases over time, currently 12.5
  - New bitcoin assigned to party named in new block
  - Called “mining” as you search for gold/coins
Bitcoin is worth (LOTS OF) money!

- 12.5 BTC = $140,000+ today

Incentivizing correct behavior?

- Race to find nonce and claim block reward, at which time race starts again for next block
  
  \[
  \text{hash (nonce} \mid| \mid \text{prev_hash} \mid| \mid \text{block data})
  \]

  - As solution has prev_hash, corresponds to particular chain

- Correct behavior is to accept longest chain
  
  - “Length” determined by aggregate work, not # blocks
  
  - So miners incentivized only to work on longest chain, as otherwise solution not accepted
  
  - Remember blocks on other forks still "create" bitcoin, but only matters if chain in collective conscious (majority)

Form of randomized leader election

- Each time a nonce is found:
  
  - New leader elected for past epoch (~10 min)
  
  - Leader elected randomly, probability of selection proportional to leader’s % of global hashing power
  
  - Leader decides which transactions comprise block

One block = many transactions

- Each miner picks a set of transactions for block
  
  - Builds "block header": prevhash, version, timestamp, txns, ...
  
  - Until hash < target OR another node wins:
    
    - Pick nonce for header, compute hash = SHA256(SHA256(header))
Transactions are delayed

- At some time $T$, block header constructed
- Those transactions had been received $[T-10\text{~min}, T]$
- Block will be generated at time $T+10\text{~min}$ (on average)
- So transactions are from $10-20\text{~min}$ before block creation
- Can be much longer if “backlog” of transactions are long

Commitments further delayed

- When do you trust a transaction?
  - After we know it is “stable” on the hash chain
  - Recall that the longer the chain, the hard to “revert”
- Common practice: transaction “committed” when 6 blocks deep
  - i.e., Takes another $\sim1\text{~hour}$ for txn to become committed

Transaction format: strawman

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø</td>
<td>25.0→PK_Alice</td>
</tr>
</tbody>
</table>

Create 12.5 coins, credit to Alice

- Transfer 3 coins from Alice to Bob
  - SIGNED(Alice)
- Transfer 8 coins from Bob to Carol
  - SIGNED(Bob)
- Transfer 1 coins from Carol to Alice
  - SIGNED(Carol)

How do you determine if Alice has balance?
Scan backwards to time 0!

Transaction format

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H(\text{prevtxn}, 0)$</td>
<td>25.0→PK_Alice</td>
</tr>
<tr>
<td>$H(\text{prevtxn}, 0)$</td>
<td>25.0→PK_Bob</td>
</tr>
<tr>
<td>$H(\text{prevtxn}, 0)$</td>
<td>25.0→PK_Alice</td>
</tr>
</tbody>
</table>

Transaction typically has 1+ inputs, 1+ outputs
- Making change: 1st output payee, 2nd output self
- Output can appear in single later input (avoids scan back)
Transaction format

- Inputs: Ø // Coinbase reward
  Outputs: 25.0 → PK_Alice

- Inputs: H(prevtxn, 0) // 25 BTC from Alice
  Outputs: 25.0 → PK_Bob

- Inputs: H(prevtxn, 0) // 25 BTC from Alice
  Outputs: 5.0 → PK_Bob, 20.0 → PK_Alice

- Inputs: H(prevtxn1, 1), H(prevtxn2, 0) // 10+5 BTC
  Outputs: 14.9 → PK_Bob

- Unspent portion of inputs is “transaction fee” to miner
- In fact, “outputs” are stack-based scripts
- 1 Block = 1MB max

Storage / verification efficiency

- Merkle tree
  - Binary tree of hashes
  - Root hash “binds” leaves given collision resistance

- Using a root hash
  - Block header now constant size for hashing
  - Can prune tree to reduce storage needs over time

- Not panacea of scale as some claim

- Scaling limitations
  - 1 block = 1 MB max
  - 1 block ~ 2000 txns
  - 1 block ~ 10 min
  - So, 3-4 txns / sec
  - Log grows linearly, joining requires full download and verification

- Visa peak load comparison
  - Typically 2,000 txns / sec
  - Peak load in 2013: 47,000 txns / sec
Summary

- Coins xfer/split between “addresses” (PK) in txns
- Blockchain: Global ordered, append-only log of txns
  - Reached through decentralized consensus
    - Each epoch, “random” node selected to batch transactions into block and append block to log
  - Nodes incentivized to perform work and act correctly
    - When “solve” block, get block rewards + tx fees
    - Reward: 12.5 BTC @ ~730 USD/BTC (11-25-16) = $9125 / 10 min
    - Only “keep” reward if block persists on main chain

Rich ecosystem: Mining pools

- Mining == gambling:
  - Electricity costs $, huge payout, low probability of winning
- Development of mining pools to amortize risk
  - Pool computational resources, participants “paid” to mine e.g., rewards “split” as a fraction of work, etc
  - Verification? Demonstrate “easier” proofs of work to admins
  - Prevent theft? Block header (coinbase txn) given by pool

Bitcoin & blockchain intrinsically linked

- security of block chain
- health of mining ecosystem
- value of currency

More than just currency...