Day 8: Proof-of-Work

(building a secure ledger)

Today

1. Recap
2. Blocks
3. Proof-of-Work (PoW)
4. Hash Chains / Block Chain

1 Recap:

- Last time,

  Public Ledger ⟶ Payment System

- Unresolved questions:
  - How is money created?
    - How is scarcity enforced?
  - How do we ensure consensus? ⟵ related
  - How do we ensure the append-only property?
  - How do we incentivize people to maintain / store the ledger?
    (incentivize ↑ to make something in someone's interest)
- Key ideas: a hash-chain & proof-of-work.

2 Blocks

- The ledger is organized into tx groups called "blocks".

  $\underbrace{tx_0, tx_1, tx_2}_{\text{Block 1}}; \underbrace{tx_3, tx_4, tx_5}_{\text{Block 2}}; \ldots$

- Blocks are added to the ledger atomically. (together / all-at-once)
- A system may specify a range of acceptable block sizes.

3. Proof-of-Work

- An idea used to:
  - Create money while,
  - Preserving scarcity, and
  - Incentivizing storage / maintenance of ledger.
- We'll:
  - Reward anyone who adds a block, but
  - make blocks hard to add (using crypto).
- A block
  - Is added to the ledger by a "miner"
  - Contains:
    - the miner's public key
    - a new coin Id for the miner's reward
    - a list of transactions
    - a "grind": a sequence of bytes that the miner sets arbitrarily (more on this later)
    - hash of previous block (more on this later)
  - In addition to the UTXOs of its txs
    - A block creates a new UTXO for the miner:
      - id = miner's reward id (For example: "1")
      - pk = miner's pk
      - amt = REWARD_AMT + $\sum$ transaction surpluses
  - So, adding to the ledger gives $.
    - How do we preserve scarcity
    - By making blocks hard to add
      - By adding extra constraints

- Requirement: first d (← integer) bits of H(block.to_bytes()) must be 0.
  - Meet this requirement by trying different grinds
    - H(Block(_, _, ..., grind=x).to_bytes()) = ??
    - " grind=y
    - " grind=z

    ← try random values until requirement is met.

Q: If H is like a random fn, what is the prob. that the last d bits are zero?

A: Pr[one bit is 0] = $\frac{1}{2}$, so $\frac{1}{2^d}$.

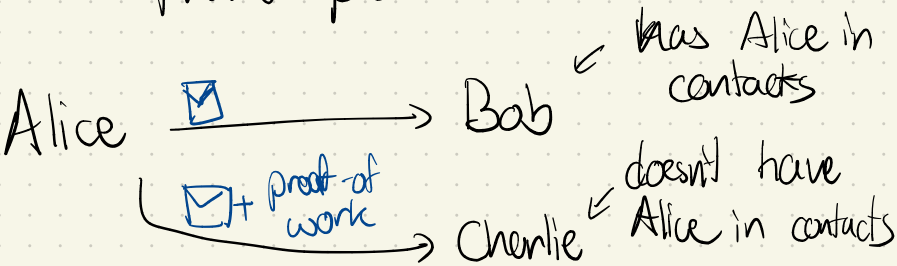
- Thus, the expected # of guessed grind values is 2^d. ← "difficulty" parameter: higher → harder to add blocks.

- The idea to require the last d bits of H(data) to be zero is called "proof of work".
  - Invented by Cynthia Dwork and Moni Naor in '93

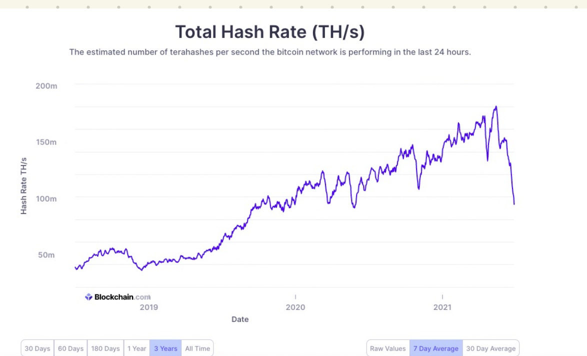
    ↑ also "differential privacy" (Cynthia Dwork)

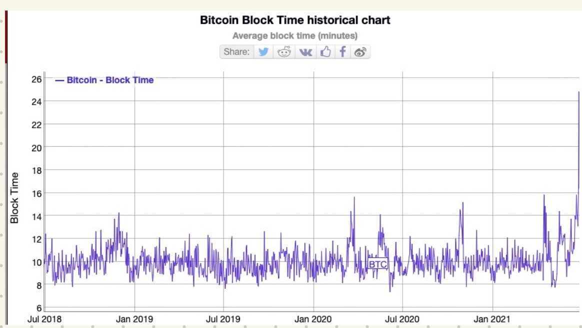
    ↑ RSA '22 Math award (Moni Naor)

- Invented for stopping email spam:
  - To email someone who doesn't know you you must provide proof-of-work.

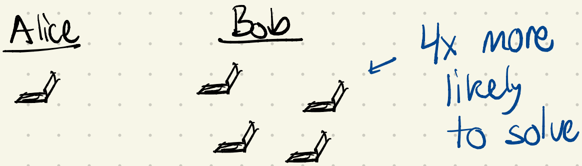


- For us:
  - moderates block production rate.
  - randomizes block production authority.
  - distributes block production authority proportionally to computational resources.
- Moderating block production rate:
  - d is raised / lowered to pace out block every 10 minutes.





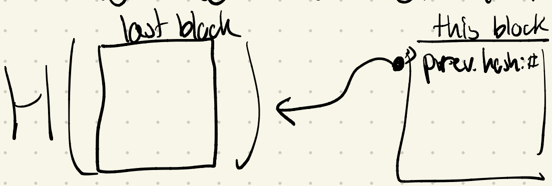
- Distributing block production authority:
  - Construct a proof-of-work → authority to add a block
  - Chance of solving PoW ~ # of computers



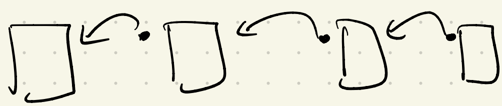
  - So as long as honest people have >50% of computers, the system behaves as intended
- "51% Attack": An adversary w/ >50% of computers can do many bad things including:
  - Refuse the transactions of people they don't like

[4] Hash-Chain ("Block Chain")

- Remaining problem: how do we agree upon the <u>order</u> of previous blocks?
  - Connect <u>all</u> previous blocks to proof-of-work.
    - Set "previous hash" field of each block to the hash of last block.



A "block chain"



- Benefits of hash-chaining:
  - Establishes a clear order of blocks / transactions
  - Ties the PoW to the order
    - new order → new PoW
    - Important, because re-ordering txs could be an attack

Example (Stocks):

1. Buy 2 shares of JPM for Alice ← $102
2. Buy 2 shares of JPM for Bob ↖ $110

switching them gives Alice a worse price!

Sell offers
1 JPM @ 50
1 JPM @ 52
1 JPM @ 53
1 JPM @ 57