PoW

A proof of work function allows to prove that you have spend a certain amount of (time/compute resources)

Def: $proof = f_{PoW}(data, d)$ — hard to compute (difficulty d)

$bool \leftarrow verify(data, proof)$ — easy to compute

Example: For a web api, require a PoW.
→ rate limit.
- a user doing individual request will not notice
- a bot doing tons of requests will notice.
  eg. crawling a social media graph.

Def: $(hash, nonce) \leftarrow f_{PoW}(Data)$

verify: $hash \stackrel{?}{=} H(Data \,||\, nonce)$ and first d bits in hash are 0.

difficulty.

Compute f: Try different nonces until one works.

Lemma: For two different nonces, the probability that they solve PoW is independent.

Thm: If p is Probability to find a nonce then the expected number of trials is $\frac{1}{p}$

increasing/decreasing d -> double/half expected work needed.

→ Exercise

Def: better version

$$(h, \text{nonce}) \leftarrow f_{PoW}(\text{Data})$$

verify: $h \stackrel{?}{=} H(\text{Data} \| \text{nonce})$

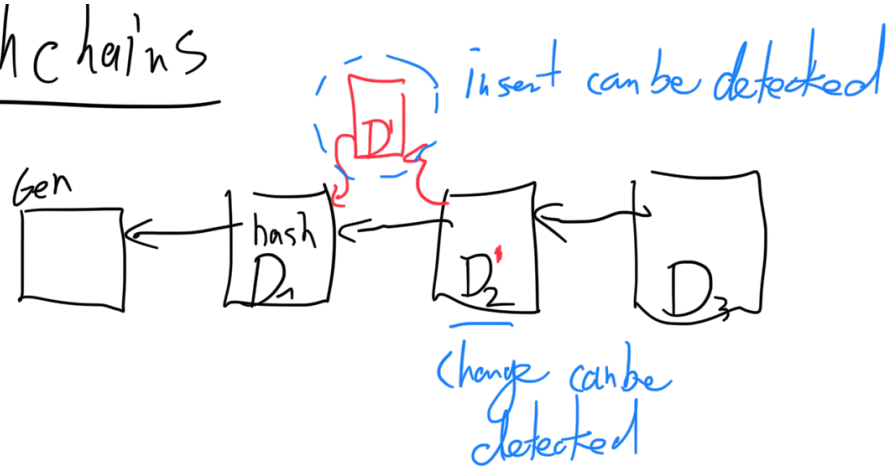
and

$h \leq d$

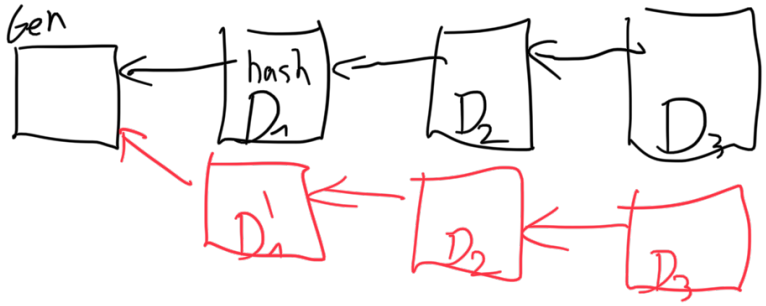
as hexadecimal number

Advantage: We can now exactly tune difficulty

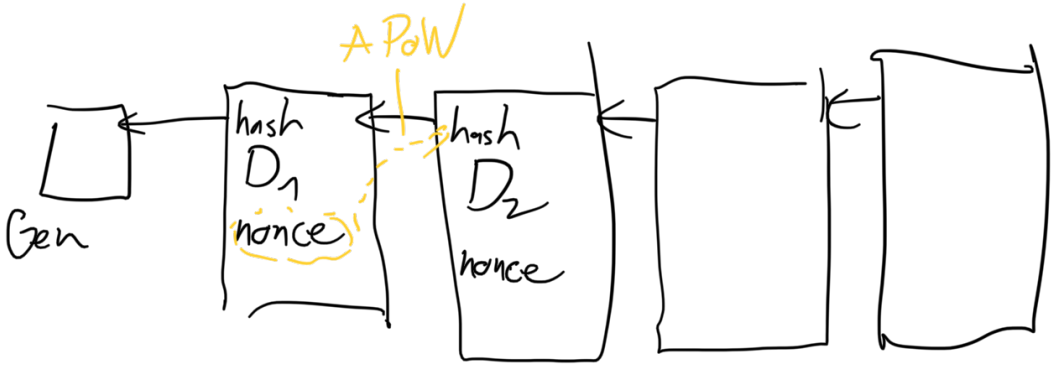
Hash chains



But can create new chain.



Idea: Add a pow. to every block.

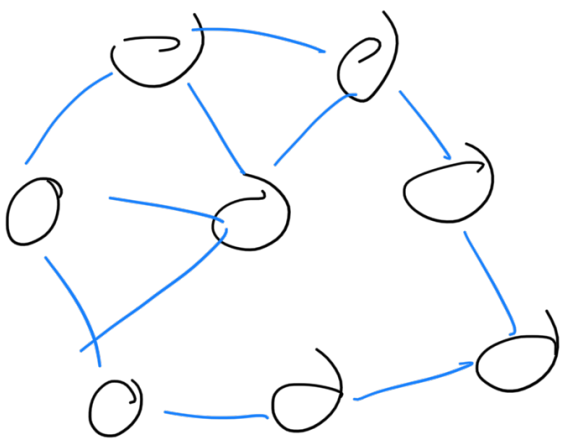


If I want to create new chain
~> recompute all proofs.

Who computes PoW and who creates the block?

A bunch of nodes / computers

A network of computers



- all store Blockchain
- all collect transactions into a block
- " try to find a nonce
- the first one to find a nonce publishes his block to everybody.
- all apply transactions in the block

Censorship resistance:

One node cannot prevent a transaction from ending in a block.

Fault tolerance:

Individual nodes may fail

Rate control:

Difficulty of PoW ensures slow block

creation.

Conflicting blocks are unlikely:

With large difficulty the probability to find two blocks at the same time is small.