cost

How to prevent sybil attacks?

Hard problem! Arms race; Twitter / FB / etc have tons of bots

Also, don't want anyone in charge

rules out SSN, phone num, captchas
work

pset01 (how's that going) needs many attempts to forge a signature if hash functions have random output, then there's no shortcut. We know what we want, but only way to get it is to keep trying.
what do you want out of work?
time consuming - like homework, but hw has problems. Trusted setup
deterministic verification
scalable - $O(1)$ to verify
memoryless - everyone gets a chance
homework work

in this case, 5 bits of 8 are constrained

need to try $2^5$ messages to find a forgery

must match:

```
x x 1 0 1 x 1 0
```
homework work

in this case, 5 bits of 8 are constrained

need to try $2^5$ messages to find a forgery

must match:

```
1 1 0 0 1 1 0 0
```
homework work

in this case, 5 bits of 8 are constrained

need to try $2^5$ messages to find a forgery

must match:

```
0 1 1 0 1 0 1 0
```

```
0 1 1 0 1 0 1 0
```

```
collisions are work

"Tadge forge 1 154262107" is a message that can be forged given the 4 signatures in the pset

Did I do 154M attempts (work)?

Maybe I did more

Maybe I got lucky and started at 150M
"proof" of work
Maybe not quite a proof; for one proof, lots of chance
For many proofs, averages out
Must estimate collision difficulty;
in this case $2^{31} = 2\text{GH}$
in this case, the "1" is thread number; 8 threads, so 1.2G, which is pretty good luck but within 2X
simpler proof of work
Signature collision is complex
Has specific target
Some kind of "universal" work
Collide with a fixed string
Collide with... zero?
hashcash
1997, idea was to stop e-mail spam
put a nonce (that's the 154262107 number) in the mail header, try to
get a *low* hash output
partial collision with 0
simpler proof of work

$ echo "Tadge 4233964024" | sha256sum
000000007e9f5bb5a25b6a0d1512095bd415840a94e2f2fe93386898947dcb07
That's 8 zeros! 4 bytes. $2^{32}$
I'm a hard worker. Will put this on my resume.
partial collision work
increased costs of equivocation / sybil resistance
scalable:
$O(n)$ work takes $O(1)$ space to prove and $O(1)$ time to verify
why work? to keep time
Big new idea in Bitcoin 9+ years ago:
Use chained proof of work as distributed time-stamping
Achieves consensus on message sequence
Solves double spend problem
block chain

message m, nonce r, target t

hash(m, r) = h; h < t
block chain
message m, nonce r, target t

hash(m, r) = h; h < t

\[ m_n = (\text{data}, h_{n-1}) \]

e.g. \[ m_2 = (\text{data}_2, \text{hash}(\text{data}_1, r)) \]
block chain
flip any bit in any block . . .
block chain
flip any bit in any block
and the chain is broken

hash: 00df
prev: 00ce
msg: hi
nonce: 5fffc

hash: 97b2
prev: 00df
msg: oh hey
nonce: 8c90

hash: 0094
prev: 002c
msg: how ru
nonce: 10ba

hash: 008a
prev: 0094
msg: good
nonce: 562e
chain forks can have two branches at a given height (number of blocks from origin)
chain forks
Highest (most work) wins
Everyone uses chain with most work

hash: 00db
hash: 002c
hash: 0094
hash: 008a
hash: 0061
hash: 00f2
hash: 00a3
chain forks
Less work chains can be discarded after the fact. "Reorg"
## Pros & Cons of PoW

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous</td>
<td>~All nonces fail</td>
</tr>
<tr>
<td>Memoryless</td>
<td>Uses watts</td>
</tr>
<tr>
<td>Scalable</td>
<td>Uses chips</td>
</tr>
<tr>
<td>Non-interactive</td>
<td>51% attacks</td>
</tr>
<tr>
<td>Tied to real world</td>
<td>People hate it</td>
</tr>
</tbody>
</table>
pros: anonymous
no pre-known key / signature
anyone can go for it
all attempts equally likely
not limited to humans
pros: memoryless
no progress. 10T failed nonces, next nonce just as likely to fail

Poisson process: always expect next block in 10 min

2X attempts / sec means 2X chance of finding next block (linear)
pros: scalable

Look at those 0s! 18 of em! 9 bytes!
(Seriously, that is $10^{22}$ attempts. Almost a mole.)

Takes just as long to check as with the 4 bytes of my name & nonce

But it's $2^{40}$ times more work!
(that's 1 trillion times more)
pros: non-interactive
1000 chips all trying once
or 1 chip trying 1000 times
equal chances; only communication is
when a block is found
pros: real world resources
Like a captcha (turing test)
Prove usage of real world resource
Can't get that time / energy back
cons: ~all nonces fail
"inefficient" - almost all attempts fail. That's no fun.

$2^{72}$ attempts needed? You will ~never find a valid proof.

Granularity is high; small players pushed out of the game
cons: uses watts & chips
Lots of electricity
Could use that to charge your car
Uses fabs, which could make more CPUs
Affects markets: GPU prices
Someday could affect electric prices
cons: irregular

Poisson process means sometimes a solution is found in a few seconds
Sometimes it takes an hour
Can deal with it but annoying;
Precludes some use cases
cons: 51% attacks
Anonymous: don't know who's got hash power. Maybe an attacker!
Attacker with 51% of total network power can write a chain faster than everyone else
Attacker can potentially rewrite history!
cons: people hate it
Not a quantitative / objective reason, but lots of people really don't like proof of work.
"The whole point of sha256 is you can't find collisions!"
"Wastes so much electricity"
"Totally pointless computation!"
Proof of Work: it Works
It's been working for 9 years
Blocks keep coming
In practice, infeasible to re-write old messages; tons of work on top
Bitcoin: very few block reorgs (rewrites), most 1 or 2 blocks deep
Next: build on it!
Fun facts
How to estimate total work done in the network?
Just look at lowest hash
Can prove total work ever with 1 hash
Can prove close calls as well to other people and show you're working