Lecture 10 - Alternative Consensus

Role of PoW: Puzzle

- Expensive to find blocks
- Extremely wasteful to create orphaned blocks in chains
Abstract puzzle

\[ P(c, r, d, n) \rightarrow \text{Valid/Invalid} \]

- Puzzle verification algorithm
- Challenge
- Nonce
- Difficulty

Bitcoin

\[ P(c, r, d) \rightarrow SHA-256^2(c, r) \leq 2^{256-d} \]
Bitcoin puzzle

- SHA-256² (block) < target

Problems:
- Wastes energy (no purpose)
- ASICs
- Pools
- Limited # of transactions (scalability)
ASIC resistance

Goal: minimize

\[
\frac{\text{Performance (ASIC)}}{\text{Performance (PC)}} \geq 1
\]

Performance = \# hashes / s / $$

Bitcoin 1516-256 10^6 

rabo
def sCrypt(N, seed):
    V = [0] * N  # initialize memory buffer of length N

    # Fill up memory buffer with pseudorandom data
    V[0] = seed
    for i = 1 to N:
        V[i] = SHA-256(V[i-1])

    # Access memory buffer in a pseudorandom order
    X = SHA-256(V[N-1])
    for i = 1 to N:
        j = X % N  # Choose a random index based on X
        X = SHA-256(X ^ V[j])  # Update X based on this index
Idea #1: Use lots of RAM

Memory-hard puzzle: Can't compute without lots of RAM

Memory-bound puzzle: Memory access time dominates cost
```python
1 def scrypt(N, seed):
2     V = [0] * N  # initialize memory buffer of length N
3     # Fill up memory buffer with pseudorandom data
4     V[0] = seed
5     for i = 1 to N:
6         V[i] = SHA-256(V[i-1])
7     # Access memory buffer in a pseudorandom order
8     X = SHA-256(V[N-1])
9     for i = 1 to N:
10        j = X % N  # Choose a random index based on X
11        X = SHA-256(X ^ V[j])  # Update X based on this index
```

8  j = X \% N  // Choose a random index based on X
9  X = SHA-256(X ^ V[j])  // Update X based on this index
Ideal:
memory-hard to compute puzzle-solution, easy to verify

Equihash (Zcash)

Generalized birthday problem:
Find $X_1, X_2, \ldots, X_k$

$\bigoplus_{i=1}^{k} H(X_i) \leq 2^{256-d}

1 \leq i \leq k$
Idea #2: multiple puzzles

- alternate puzzles
- solve all puzzles

XII

Idea #3: moving, target (requires consensus)

unpredictable

"ASIC honeymoon"
Is ASCII-resistance desirable?
Non-otutable puzzles

Can't check solutions w/o knowledge of private key

\[ P(c, r, d, \text{privkey}) \rightarrow \text{Valid/Invalid} \]

Example (almost):

\[ \text{Sign}(c, r) \geq 2^{256} - d \]
Energy use ideas

1) Useful puzzles

2) Virtual mining (proof-of-stake)
<table>
<thead>
<tr>
<th>Project</th>
<th>Founded</th>
<th>Goal</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Great Internet Mersenne Prime Search</td>
<td>1996</td>
<td>Finding large Mersenne primes</td>
<td>Found the new “largest prime number” twelve straight times, including $2^{3785161} - 1$</td>
</tr>
<tr>
<td>distributed.net</td>
<td>1997</td>
<td>Cryptographic brute-force demos</td>
<td>First successful public brute-force of a 64-bit cryptographic key</td>
</tr>
<tr>
<td>SETI@home</td>
<td>1999</td>
<td>Identifying signs of extraterrestrial life</td>
<td>Largest project to date with over 5 million participants</td>
</tr>
<tr>
<td>Folding@home</td>
<td>2000</td>
<td>Atomic-level simulations of protein folding</td>
<td>Greatest computing capacity of any volunteer computing project. More than 118 scientific papers.</td>
</tr>
</tbody>
</table>

Table 8.3: Popular “Volunteer computing” projects
Required properties of puzzles

- no trusted party to hand out work
- easy verification of solutions
- difficulty adjustment
- progress-free
Prime coin

Puzzle:

Cunningham chain:

\[ P_1, P_2, \ldots, P_k \] where \( P_i = 2 \cdot P_{i-1} + 1 \)

Prime coin puzzle:

\[ P_i = \text{challenge} \_ \text{nonce} \]
Miners "vote" on blockchain in proportion to how much cryptocurrency they hold "stake"
Approach 1 (Peercoin) Hybrid approach

\[ H(c, r) \leq 2 \]

\( S = \text{coin-age} \)

\[ X_{BTC, \text{time}} \rightarrow \text{Coinbase} \rightarrow S = \sum Y_{i \cdot t} \]
Approach #2: pure proof-of-stake

H

[Diagram of network with nodes labeled as Alice, Bob, Charlie, etc.]

NXT
Snow White
ALGO ALGORAND
Ouroboros
CASPER