Lecture 11

Alternative Mining Puzzles
Puzzles are the core of Bitcoin

- Determine the incentive system, and nature of puzzles determines behavior of miners
- Basic features of Bitcoin’s proof-of-work puzzle (recap)
  - Puzzle is difficult to solve, so large-scale attacks are difficult
  - ... but not too hard, so honest miners are compensated
- What other features could a puzzle have?
This lecture (and later)

- Alternative puzzle designs
  Used in practice, and research proposals

- Variety of possible goals
  ASIC resistance, pool resistance, environmental-friendliness, intrinsic benefits...

- Essential security requirements
Basic Puzzle Requirements
Puzzle requirements

- **Cheap to Verify**
  - since other users have to verify solutions

- **Adjustable difficulty**
  - E.g., due on increase in hash rate or more users

- **In PoW puzzles, chance of winning should be proportional to computing power (e.g., hash power in Bitcoin)**
  - Large players get only proportional advantage
  - Even small players get proportional compensation
Bad PoW puzzle: a *sequential* puzzle

Consider a puzzle that takes N steps to solve a “Sequential” Proof of Work
Bad PoW puzzle: a *sequential* puzzle

Consider a puzzle that takes $N$ steps to solve

a “Sequential” Proof of Work
Bad PoW puzzle: a *sequential* puzzle

Consider a puzzle that takes $N$ steps to solve a “Sequential” Proof of Work
Bad PoW puzzle: a *sequential* puzzle

Consider a puzzle that takes \( N \) steps to solve a “Sequential” Proof of Work
Bad PoW puzzle: a *sequential* puzzle

Consider a puzzle that takes $N$ steps to solve a “Sequential” Proof of Work
Bad PoW puzzle: a *sequential* puzzle

Consider a puzzle that takes $N$ steps to solve a “Sequential” Proof of Work
Bad PoW puzzle: a \textit{sequential} puzzle

Consider a puzzle that takes \( N \) steps to solve a “Sequential” Proof of Work
Bad PoW puzzle: a *sequential* puzzle

Problem: fastest miner *always* wins the race!
Bad PoW puzzle: a *sequential* puzzle

Problem: fastest miner *always* wins the race!
Bad PoW puzzle: a *sequential* puzzle

Problem: fastest miner *always* wins the race!
Bad PoW puzzle: a *sequential* puzzle

Problem: fastest miner *always* wins the race!

Solution Found!
Good PoW puzzle → Weighted sample

This property is sometimes called “progress-free”
ASIC Resistant (PoW) Puzzles
ASIC resistance - Why? (1 of 2)

Goal: Ordinary people with idle laptops, PCs, or even mobile phones can mine!

Lower barrier to entry

Approach: Reduce the gap between custom hardware and general purpose equipment
ASIC resistance - Why? (2 of 2)

Goal: Prevent large manufacturers from dominating the game

“Burn-in” advantage
In-house designs

Approach: reduce the “gap” between future hardware and the custom ASICs we already have
Memory hard puzzles

Premise: the cost and performance of memory is more stable than for processors

![Graph showing the performance of processors, memory, and storage over time. The graph indicates a "performance gap" between processors and memory.]
scrypt  Colin Percival, 2009

- Memory hard hash function
  - *Constant time/memory tradeoff*
  - Memory consumes a large amount of on-chip area. High memory requirement => small number of hashing engines on special-purpose chips
- Widely used alternative PoW puzzle (e.g., Litecoin)
- Also used in Password-hashing

1. Fill memory with random values
2. Read from the memory in random order
scrypt - step 1 of 2 (write)

Input: \( x \)

\( V_1 = H(X) \)

\( V_2 = H(V_1) = H(H(X)) \)

\( V_3 = H(V_2) = H^3(X) \)

\( \ldots \)

\( V_N = H^N(x) \)
scrypt - step 1 of 2 (write)

Input: \( x \)

\[
\begin{align*}
V_1 & = H(X) \\
V_2 & = H(V_1) = H(H(X)) \\
V_3 & = H(V_2) = H^3(X) \\
\vdots \\
V_N & = H^N(x)
\end{align*}
\]
scrypt - step 1 of 2 (write)

Input: \( x \)

\( V_1 = H(X) \)

\( V_2 = H(V_1) = H(H(X)) \)

\( V_3 = H(V_2) = H^3(X) \)

... 

\( V_N = H^N(x) \)
scrypt - step 1 of 2 (write)

Input: \( x \)

\[
\begin{align*}
V_1 &= H(X) \\
V_2 &= H(V_1) = H(H(X)) \\
V_3 &= H(V_2) = H^3(X) \\
&\quad \vdots \\
V_N &= H^N(x)
\end{align*}
\]
scrypt - step 1 of 2 (write)

Input: \( x \)

\[ V_1 = H(X) \]

\[ V_2 = H(V_1) = H(H(X)) \]

\[ V_3 = H(V_2) = H^3(X) \]

... 

\[ V_N = H^N(x) \]
Input: $x$

$A := H^{N+1}(X)$

For $N$ iterations:

- $i := A \mod N$
- $A := H(A \text{xor } V_i)$

Output: $A$
scrypt - step 2 of 2 (read)

Input: x
A := H^{N+1}(x)

For N iterations:

i := A \mod N
A := H(A \text{ xor } V_i)

Output: A
scrypt - step 2 of 2 (read)

Input: x
A := \( H^{N+1}(X) \)

For N iterations:

\( i := A \mod N \)
A := \( H(A \oplus V_i) \)

Output: A
scrypt - step 2 of 2 (read)

Input: x
A := H^{N+1}(x)

For N iterations:

\[ i := A \mod N \]
\[ A := H(A \text{ xor } V_i) \]

Output: A
**scrypt** - time/memory tradeoff

Why is this memory-hard?
Reduce memory by half, 1.5x the # steps

Need to access $V_i$ where $i$ is even?

Access $V_{i-1}$

Compute $V_i = H(V_{i-1})$
scrypt

Disadvantages: Also requires N steps, N memory to check

Is it actually ASIC resistant?

scrypt ASICs are already available
Exploit time-memory trade-offs, lower values of N, etc.
Academic research

- Many subsequent candidates: Argon2i (winner of PW-hashing contest), Ballon-Hashing, etc.
- Proofs of memory hardness in various models using graph pebbling techniques (see, e.g., Alwen-Serbeninko’15 and many subsequent works)
- See talk at Theory Seminar this Wednesday (Malone 228, 12-1pm) on this subject
Cuckoo hash cycles  

Memory hard puzzle that’s cheap to verify

Input: X

For i = 1 to E:

\[ a := H_0(X + i) \]
\[ b := N + H_1(X + i) \]

\( \text{edge}(a \mod N, b \mod N) \)

Is there a cycle of size K?  If so, Output: X, K edges
Cuckoo hash cycles  

Memory hard puzzle that’s cheap to verify

Input: X

For i = 1 to E:

\[ a := H_0(X + i) \]

\[ b := N + H_1(X + i) \]

\[ \text{edge}(a \mod N, b \mod N) \]

Is there a cycle of size K? If so, Output: X, K edges
Cuckoo hash cycles

Memory hard puzzle that’s cheap to verify

Input: X

For $i = 1$ to $E$:

$a := H_0(X + i)$

$b := N + H_1(X + i)$

$\text{edge}(a \mod N, b \mod N)$

Is there a cycle of size $K$? If so, Output: $X$, $K$ edges
Cuckoo hash cycles  
John Tromp, 2014

Memory hard puzzle that’s cheap to verify

Input: X

For i = 1 to E:

\[ a := H_0(X + i) \]

\[ b := N + H_1(X + i) \]

\[ \text{edge}(a \mod N, b \mod N) \]

Is there a cycle of size K? If so, Output: X, K edges
Cuckoo hash cycles

Memory hard puzzle that’s cheap to verify

Input: X

For $i = 1$ to $E$:

$a := H_0(X + i)$

$b := N + H_1(X + i)$

$edge(a \mod N, b \mod N)$

Is there a cycle of size $K$? If so, Output: $X, K$ edges
Cuckoo hash cycles  

John Tromp, 2014

Memory hard puzzle that’s cheap to verify

Input: X

For i = 1 to E:

\[ a := H_0(X + i) \]

\[ b := N + H_1(X + i) \]

\[ \text{edge}(a \mod N, b \mod N) \]

Is there a cycle of size K? If so, Output: X, K edges
Cuckoo hash cycles  

Memory hard puzzle that’s cheap to verify

Input: X

For $i = 1$ to $E$:
- $a := H_0(X + i)$
- $b := N + H_1(X + i)$

```
    edge(a mod N, b mod N)
```

Is there a cycle of size $K$? If so, Output: $X$, $K$ edges
Cuckoo hash cycles  

Memory hard puzzle that’s cheap to verify

Input: X

For i = 1 to E:

\[ a := H_0(X + i) \]
\[ b := N + H_1(X + i) \]

\[ \text{edge}(a \mod N, b \mod N) \]

Is there a cycle of size K? If so, Output: X, K edges
Cuckoo hash cycles  

Memory hard puzzle that’s cheap to verify

Input: X

For $i = 1$ to $E$:

$a := H_0(X + i)$

$b := N + H_1(X + i)$

$\text{edge}(a \mod N, b \mod N)$

Is there a cycle of size $K$? If so, Output: $X$, $K$ edges
Even more approaches

● More complicated hash functions
  ● X11: 11 different hash functions combined
       (subsequent iterations: X13, X14, X15, X17)

● Moving target
  Change the puzzle periodically
Counter argument: SHA2 is fine

Bitcoin Mining ASICs aren’t changing much
Big ASICs only marginally more performant than small ones
Proof-of-useful-work
Recovering wasted work

Recall: power consumed by Bitcoin network in 2017 ~ power consumed by Denmark

Natural question:
- Can we recycle this and do something useful?
Candidates - needle in a haystack

● Natural choices:
  - Protein folding (find a low energy configuration)
  - Search for aliens (find an anomalous region of a signal)

● Challenges:
  - Randomly chosen instances must be hard

Who chooses the problem?
Puzzle based on finding large prime numbers

Cunningham chain:

\[ p_1, p_2, \ldots, p_n \]

where \( p_i = 2^i a + 1 \)

Each \( p_i \) is a large (probable) prime

\( p_1 \) is divisible by \( H(\text{prev}|| \text{mrkl\_root} || \text{nonce}) \)
Primecoin

- Many of the largest known Cunningham chains have come from Primecoin miners.

- Hard problem? Studied by others (e.g., PrimeGrid).

- Usefulness? Some applications to crypto (e.g., Young-Yung’98).
Recovering wasted hardware

Estimate: more than $100M spent on customized Bitcoin mining hardware

This hardware investment is otherwise useless

Idea: a puzzle where hardware investment is useful, even if the work is wasted?
Permacoin - Mining with storage  

Side effect:  
Massively distributed, replicated storage system
Permacoin

Assume we have a large file $F$ to store

For simplicity: $F$ is chosen globally, at the beginning, by a trusted dealer

Each user stores a random subset of the file
Storage-based puzzle

1. Build a Merkle tree, where each leaf is a segment of the file
Storage-based puzzle

1. Build a Merkle tree, where each leaf is a segment of the file

2. Generate a public signing key \( pk \), which determines a random subset of file segments
Storage-based puzzle

1. Build a Merkle tree, where each leaf is a segment of the file

2. Generate a public signing key pk, which determines a random subset of file segments

3. Each mining attempt:
   a) Select a random nonce
   b) $h_1 := H(prev || mrkl\_root || PK || nonce)$
Storage-based puzzle

1. Build a Merkle tree, where each leaf is a segment of the file

2. Generate a public signing key pk, which determines a random subset of file segments

3. Each mining attempt:
   a) Select a random nonce
   b) $h_1 := H(\text{prev} \ || \ \text{mrkl\_root} \ || \ PK \ || \ \text{nonce})$
   c) $h_1$ selects $k$ segments from subset
Storage-based puzzle

1. Build a Merkle tree, where each leaf is a segment of the file

2. Generate a public signing key pk, which determines a random subset of file segments

3. Each mining attempt:
   a) Select a random nonce
   b) \( h_1 := H(prev || mrkl\_root || PK || nonce) \)
   c) \( h_1 \) selects \( k \) segments from subset
   d) \( h_2 := H(prev || mrkl\_root || PK || nonce || F) \)
   e) Winner if \( h_2 < \text{TARGET} \)
Reducing Bitcoin’s “honesty” cost

“Honest” miners validate every transaction

Validation requires the UTXO database (GBs)

Maintaining the UTXO database doesn’t pay

Idea: use Permacoin to reward UTXO storage
Proofs of Space

- Require non-trivial storage (as opposed to computational power) to solve a puzzle [Dziembowski et al. CRYPTO’15, Ateniese et al. SCN’14]

- More environmental-friendly
- Used in SpaceMint (see also Burstcoin)
Summary

• Useful proof-of-work is a natural goal
  (while maintaining security requirements)
• The benefit must be a pure public good
• Viable approaches include storage, prime-finding, others may be possible
• Realized benefit so far has been limited
Nonoutsourceable Puzzles
Large mining pools are a threat

- Bitcoin’s core value is decentralization

- If power is consolidated in a few large pools, the operators are targets for coercion/hacking

- Position: large pools should be discouraged!
  Analogy to voting: It’s illegal (in US) to sell your vote
June 12, 2014
GHash.IO large mining pool crisis
It's Time For a Hard Bitcoin Fork

Ittay Eyal, and Emin Gün Sirer

Friday June 13, 2014 at 02:05 PM

A Bitcoin mining pool, called GHash and operated by an anonymous entity called CEX.io, just reached 51% of total network mining power today. Bitcoin is no longer decentralized. GHash can control Bitcoin transactions.

Is This Really Armageddon?

Yes, it is. GHash is in a position to exercise complete control over which
Observation:

Pool participants don’t trust each other

Pools only work because the “shares” protocol lets members *prove* cooperation
Standard Bitcoin mining pool

Solution found!

“shares”: proof that a member is “toeing the line”

Payout dividing among members
The Vigilante Attack

Suppose a Vigilante is angry with a large pool

He submits “shares” like normal....

... but if he finds a real solution, discards it

Pool output is reduced, Vigilante loses a little
The Vigilante Attack

Solution discarded

Payout dividing among members

“shares”: proof that a member is “toeing the line”
Encouraging the Vigilante

Whoever *FINDS* a solution spends the reward

Approach:
- searching for a solution requires *SIGNING*, not just hashing. (Knowledge of a private key)
- Private key can be used to spend the reward
Encouraging the Vigilante

Solution found!

Take the money and run!
Nonoutsourcable puzzle

Solution:

\((\text{prev}, \text{mrkl}\_\text{root}, \text{nonce}, \text{PK}, \ s1, \ s2)\)

such that:

\[ H(\text{prev} \ || \ \text{PK} \ || \ \text{nonce} \ || \ s1) < \text{TARGET} \]
\[ \text{VerifySig} (\text{PK}, \ s1, \ \text{prev} \ || \ \text{nonce}) \]
\[ \text{VerifySig} (\text{PK}, \ s2, \ \text{prev} \ || \ \text{mrkl}\_\text{root}) \]
Nonoutsourceable puzzle

Solution:

\[(\text{prev, mrkl\_root, nonce, PK, s1, s2})\]

such that:

\[H(\text{prev || PK || nonce || s1}) < \text{TARGET}\]

\[\text{VerifySig(PK, s1, prev || nonce)}\]

\[\text{VerifySig(PK, s2, prev || mrkl\_root)}\]
Nonoutsourcable puzzle

Solution:

\[(\text{prev}, \text{mrkl\_root}, \text{nonce}, \text{PK}, s1, s2)\]

such that:

\[H(\text{prev} || \text{PK} || \text{nonce} || s1) < \text{TARGET}\]

\[\text{VerifySig(PK, s1, prev || nonce)}\]

\[\text{VerifySig(PK, s2, prev || mrkl\_root)}\]
Nonoutsourceable puzzle

Solution:

\[(\text{prev}, \text{mrkl\_root}, \text{nonce}, \text{PK}, s_1, s_2)\]

such that:

\[
\begin{align*}
H(\text{prev} \ || \ \text{PK} \ || \ \text{nonce} \ || \ s_1) &< \text{TARGET} \\
\text{VerifySig}(&\text{PK}, s_1, \text{prev} \ || \ \text{nonce}) \\
\text{VerifySig}(&\text{PK}, s_2, \text{prev} \ || \ \text{mrkl\_root})
\end{align*}
\]
Proof-of-Stake

“Virtual Mining”
Bitcoin Mining has an unnecessary step

Proof-of-Work Mining:

- **Miner**
  - Spend money on power and equipment
  - Find puzzle solutions
  - Earn mining rewards
Bitcoin Mining has an unnecessary step

Proof of Stake:

○ Creator of next block chosen at random based on current stake in the system

○ Assuming all the money owned/used by miners is in the system, this mechanism cuts the middle man (equipment manufacturer)
Potential benefits

● Lower overall costs
  - No harm to the environment
  - Savings distributed to all coin holders
● Stakeholder incentives - good stewards?
● No ASIC advantage
● 51% attack is even harder
51% attack prevention argument
The Bitcoin economy is smaller than the world
Wealth outside Bitcoin has to move inside
51% attack prevention argument
The Bitcoin economy is smaller than the world Wealth outside Bitcoin has to move inside
51% attack prevention argument
The Bitcoin economy is smaller than the world wealth outside Bitcoin has to move inside.
Variations of Virtual Mining

- Proof-of-Stake: “Stake” of a coin grows over time as long as the coin is unused (but potentially some upper limit)
- Proof-of-Burn: mining with a coin destroys it
- Proof-of-Deposit: can reclaim a coin after some time
- Proof-of-Activity: any coin might be win (if online)
Questions with Virtual Mining

Is there any security that can only be gained by consuming “real” resources?

- If so, then “waste” is the cost of security
- If not, then PoW mining may go extinct
Examples of PoS based Cryptocurrencies

- Peercoin
- Blackcoin
- Nxt
- Neucoin
- ...
Examples of secure PoS systems

- Algorand [Full version: Chen-Micali’17]
- Ourboros [Kiayias-Russel-David-Oliynykov’17]
- Snow white [Daian-Pass-Shi’17]
**Conclusion**

- Many possible design goals
  - Prevent ASIC miners from dominating
  - Prevent large pools from dominating
  - Intrinsic usefulness
  - Eliminate the need for mining hardware at all
- Further research required to understand the best tradeoffs
- Many competing systems already co-exist