



CS1699: Blockchain Technology and Cryptocurrency

15. Variations on a Theme: Non-SHA256 Puzzles

Bill Laboon

Recall: Hash Puzzle Properties

1. Difficult to compute
2. Parameterizable (i.e. adjustable) cost
3. Trivial to verify
4. Progress-free (memoryless process)

Bitcoin's Puzzle

- ❖ *Partial hash-preimage puzzle*
- ❖ Goal is to find block / nonce where:
$$H(\text{block} || \text{nonce}) < \text{target}$$
- ❖ Could we modify this in any way while maintaining the three properties of a good puzzle: Difficult to compute, parameterizable cost, trivial to verify?

Slight Variations on Bitcoin's Puzzle

- ❖ Number of 0's or 1's in binary representation of hash
- ❖ $H(\text{block} \parallel \text{nonce}) > \text{target}$
- ❖ $H(\text{block} \parallel \text{nonce}) \text{ XOR value} < \text{target}$

What About More Extreme Variations?

- ❖ Of course!
- ❖ But why would we want to do it?

Mining

- ❖ Most Bitcoin users do not mine, as you need specialized hardware
- ❖ Is this a good or a bad thing?

ASIC Resistance

- ❖ “Disincentivize the use of custom-built hardware for mining”
- ❖ This will never be perfect - some hardware will always be better at mining than others (e.g. different instructions on different chipsets)
- ❖ Thus, there is always some incentive (assuming a currency with value) to make our own specialized hardware
- ❖ But can we *narrow the gap*?

Memory-Hard vs Memory-Bound

- ❖ A puzzle that requires lots of memory instead of (or in addition to) lots of CPU power is *memory-hard*
- ❖ If time to access memory is limiting factor of computation, the puzzle is *memory-bound*
- ❖ *Puzzles can be either, neither or both!*

ASIC Resistance And Memory

- ❖ For ASIC resistance, we want puzzles to be both memory-hard and memory-bound... why?
- ❖ Time bound by memory storage, not processor design
- ❖ Variation in memory access relatively small across systems and harder to speed up
- ❖ Harder to parallelize without just buying more memory
- ❖ SHA-256 is neither memory-hard nor memory-bound

scrypt

- ❖ Pronounced “ess-crypt”, not “script”
- ❖ Originally designed for hashing passwords
- ❖ Similar to Bitcoin’s partial hash-preimage puzzle but has reliance on

“Simple scrypt” Pseudocode

```
def scrypt(size, seed) {  
    buf = [0] * size  
    buf[0] = seed  
    for j = (1..size) {  
        buf[j] = SHA256(buf[j - 1])  
    }  
    x = SHA256(buf[size - 1])  
    for j = (1..size) {  
        k = x % size  
        x = SHA256(x ^ buf[k])  
    }  
    return x  
}
```

Simple script

Work out on board

Why Is This Memory-Hard?

- ❖ Re-computing all of the values in buffer is possible
- ❖ But value of k changes pseudorandomly, thus would need to calculate (on average) $size / 2$ SHA-256 hashes
- ❖ If buffer used: $2 * size$ computations ($O(n)$)
- ❖ No buffer: $size * (size / 2)$ computations ($O(n^2)$)

Time/Memory Trade-Offs of scrypt

- ❖ Assume buffer of size $size / 2$ (half of original)
- ❖ Store values of $buf[k]$ iff k is even, recalculate later if odd
- ❖ Can generalize, can store every k th row of buffer
- ❖ Uses $size / k$ memory, computing $((k + 3) * size) / 2$ iterations of SHA256

“Tipping the Scales” Towards Memory

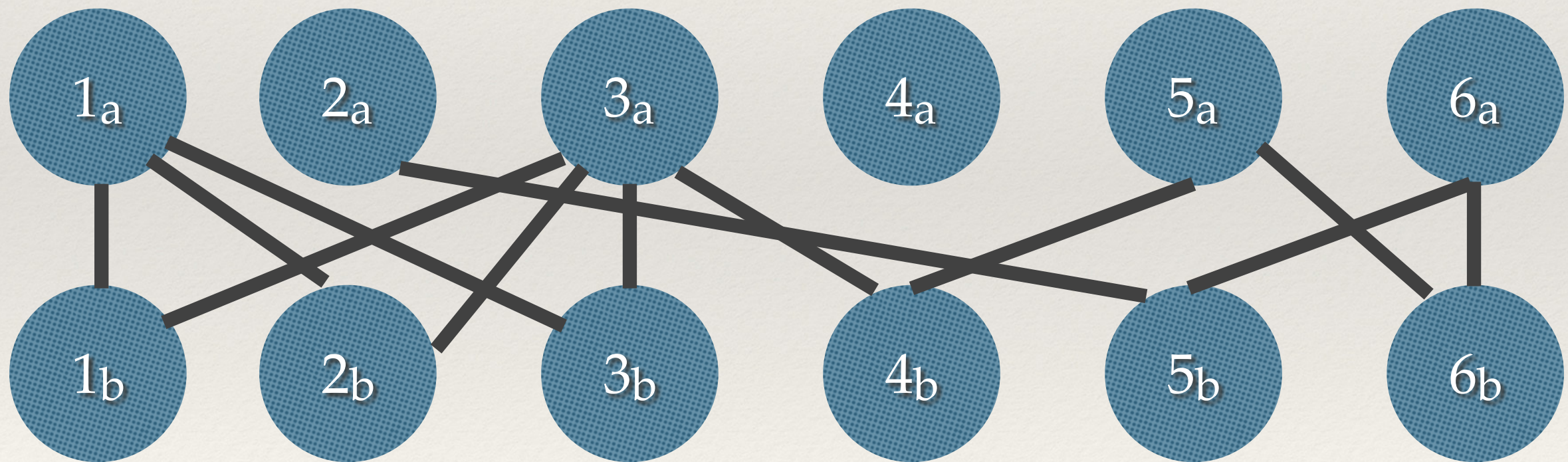
- ❖ What if the buffer itself were updated by each iteration?
- ❖ Would make time / memory trade-off even more in favor of memory as it would involve lots of recalculation

Cuckoo Cycle

- ❖ Graph-theoretic hashing algorithm
- ❖ Used for æternity blockchain proof of work
- ❖ A variant (Cuckatoo cycle) used in Grin
- ❖ <https://github.com/tromp/cuckoo>
- ❖ <https://github.com/tromp/cuckoo/blob/master/doc/cuckoo.pdf>

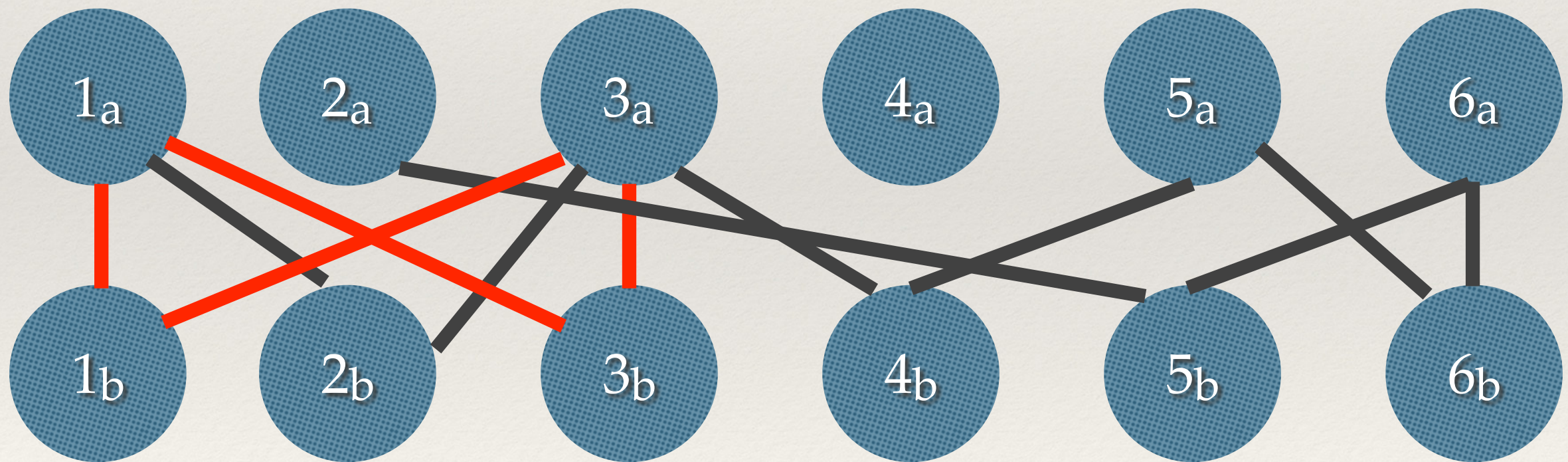
Cuckoo Cycle In A Nutshell

Given a bipartite graph, find a cycle



Cuckoo Cycle In A Nutshell

Given a bipartite graph, find a cycle:
 $1a - 1b - 3a - 3b - 1a$



Using Cuckoo Cycles for PoW

- ❖ Given some data, send it through a hash function (Grin and æternity use SIPHASH) multiple times
- ❖ Interpret output from hash function as edges between nodes
- ❖ Look for cycle of a specific length (for Grin, 42)

Cuckoo Cycle

Work out on board

Is The Cuckoo Cycle A Good Puzzle?

1. *Difficult to compute?* **yes, especially as size number of nodes increase**
2. *Parameterizable (i.e. adjustable) cost?* **yes, increase number of nodes, size of cycle, or add restrictions to which edges are “valid”**
3. *Trivial to verify?* **yes, easy to check if given cycle path**
4. *Progress-free (memoryless process)?* **yes, need to just try different possibilities**