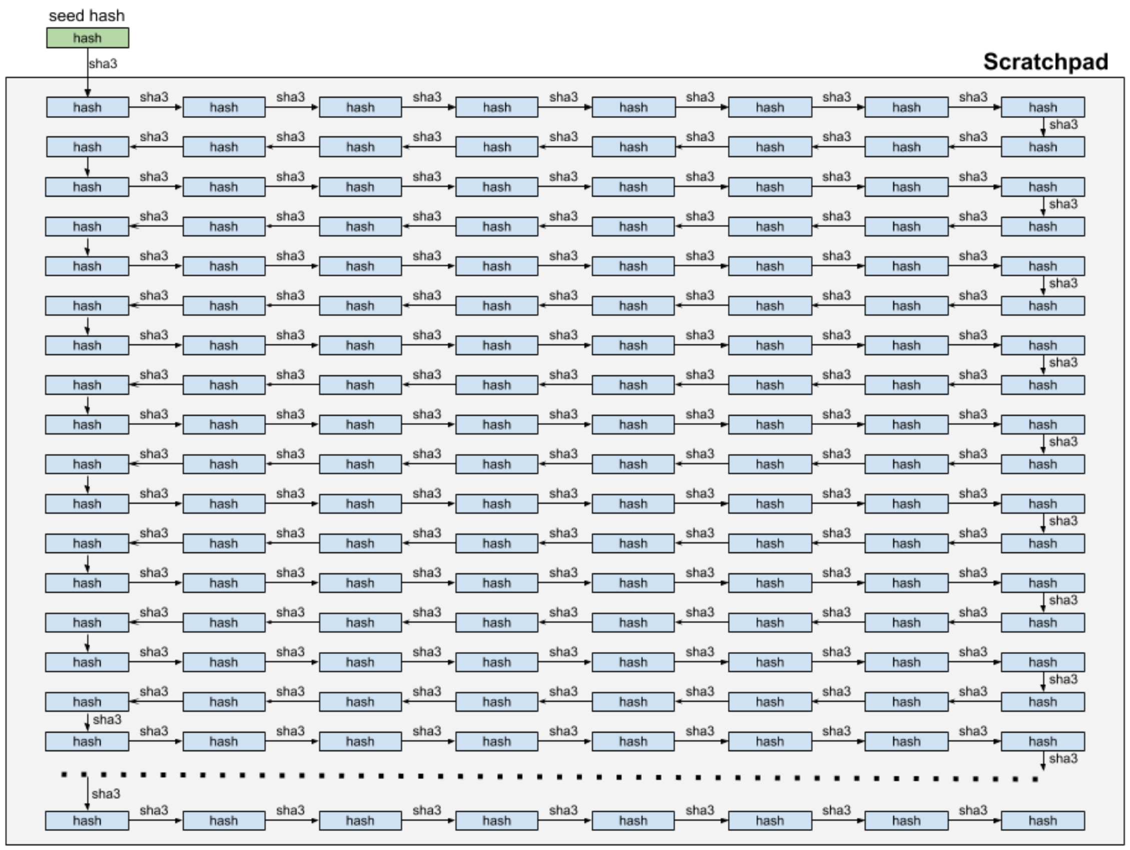
WildKeccak2 Scratchpad Generation Analysis

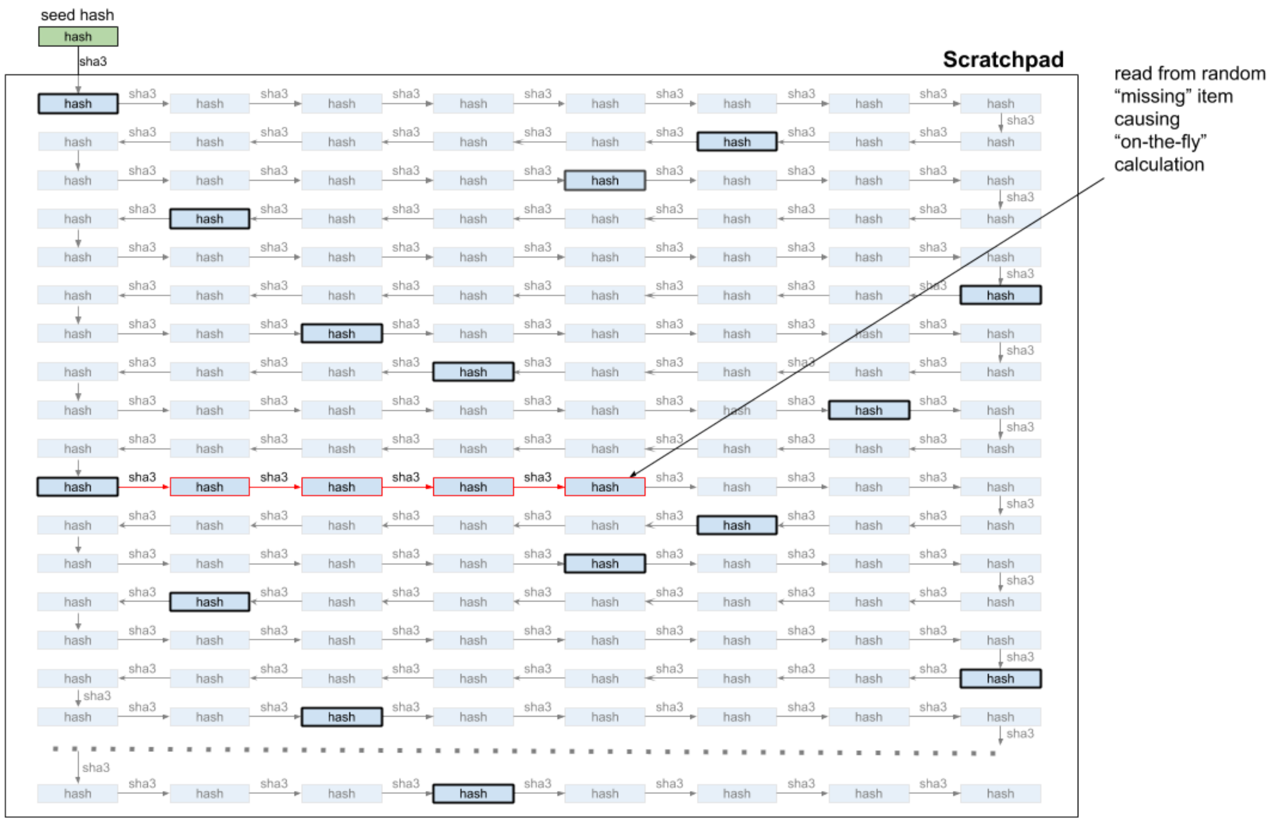
By calculating a scratchpad's items "on the fly", memory-hard algorithms may be optimized from resultant memory access operations. As such, the main advantages of a memory-hard PoW hash are eliminated. In this draft proposal, we present a theoretical analysis supporting the question of whether such operations would be reasonable for WildKeccak2 while using our current scratchpad generation scheme.

At this time of writing, our scratchpad calculation looks like the following:



Initially derived from seed, each next group of 32 bytes (shown as a hash for easier interpretation, let’s call it “item”) is generated from the previous group by calculating a sha3 hash. After all, we are getting a relatively big (starting from 500MB) solid memory block which is used as a scratchpad data for hashing in WildKeccak2. Each WildKeccak2 hash funtion call performs 7200 random read operations on this scratchpad.

With this approach, it is possible to have a light version of the scratchpad (the same idea as used in EthHash) which would be a trade-off between performance and memory consumption. For example, we can keep every 10-nth item and call them “bearing items”. Then, during the hash calculation, every time we need an arbitrary item, we have to calculate “on the fly” the missing items by calculating the SHA3 from the closest “bearing item”. This will then slow down the hash calculation but, unless this is *not* a mining process, this is still a reasonable amount of time.

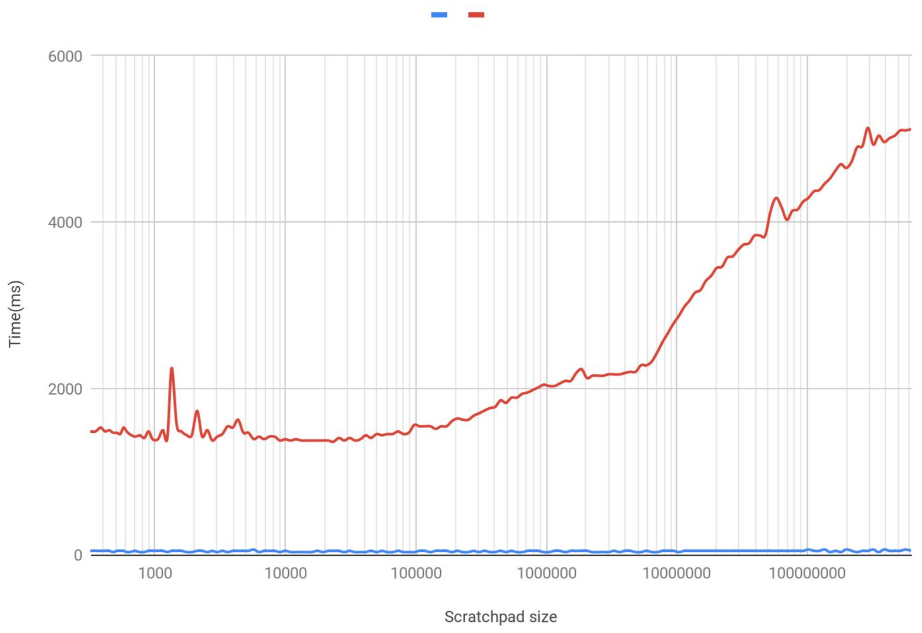


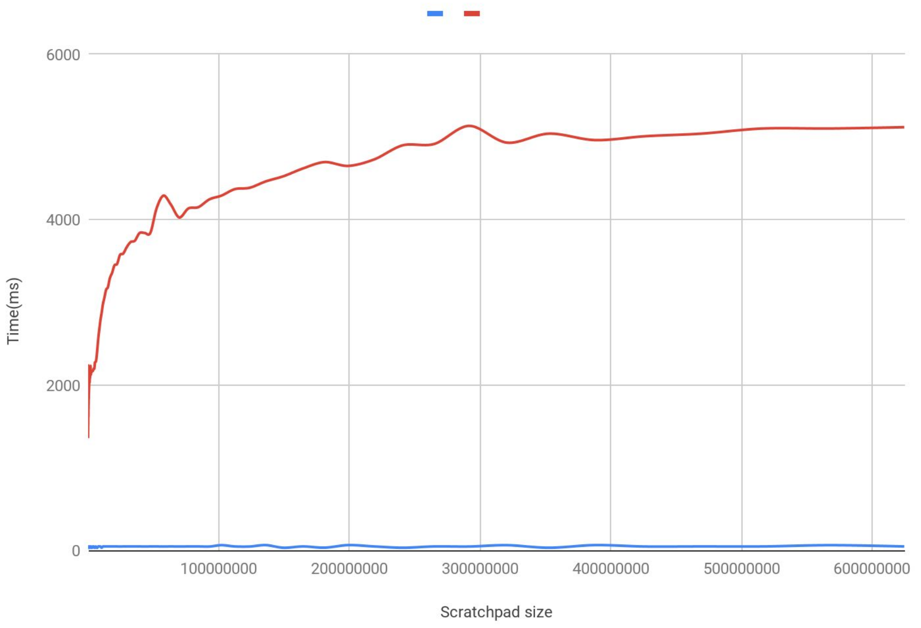
seed hash
hash
sha3
Scratchpad
read from random “missing” item causing “on-the-fly” calculation
hash
sha3

Now, as soon as we want to develop a memory-hard hash function, we would need to analyze whether it would be cheaper/reasonable for theoretical ASICs to recalculate the missing items instead of accessing global memory (which is slow and has a limited throughput).

Our question: how do we measure the time needed for reading one memory line of 8 bytes (uint64_t) relative to the SHA3 calculation time.

Let’s look at the graph with results of the measuring time (ms) needed to run 10000 WK2 (WildKeccak2, red line) hash operations on different sized scratchpads. X axis is the size of the scratchpad and Y axis is time(ms):





Both graphics show the same sequence: the left side displayed in logarithmic scale, the right side displayed in linear. This made for an easier reading of difference from the very beginning of the graph (with a very small size of scratchpad, where all operations mostly fit in CPU's 1st level cache memory) compared to very last part, where the length of time appears to stop growing because nearly all memory access operations are addressed to global memory (DRAM).

Here, we can see in the red line that WK2 running on very fast 1st level cache memory takes about 1400ms and WK2 running on global memory is about 5000ms (all numbers measured against 10000 hash operations). So, the latency loss on accessing global memory is 5000-1400=3600 ms for groupings of 7200 random read operations, with approximately 1/2 ms for each operation (still talking in terms of x10000 basis). And, the blue line, on x10000 basis, a SHA3 operation takes an average of 40 ms.

Following this logic, 1 random memory read cost (in a very rough approximation) is 1/80th of a SHA3 calculation coast. This leads us to believe that generating a WK2 scratchpad (that derives each subsequent item from the previous by only taking 1 hash) is a sensible solution because the cost to calculate at least one item is roughly x80 more then if read from global memory(normally need to calculate from 1 until 9 items).