Ethash computation question

David Sanda, 26.4.2017

Intro: My understanding of how ethash works is based on <https://github.com/ethereum/wiki/wiki/Ethash>. I will describe my understanding of the relevant principles and what I see as potential flaw (or result of careless reading). Answering it is more of a research interest not a business case, but it might be of interest to you.

Ethash calculation:

Once the block is put together and trie is constructed and uncles are assembled, actual ethash calculation starts. **Block construction** is nontrivial in a sense, that blocks reference each other and maintain in some sense a complete state for each involved account, that is not just the current transaction, but also actual balance of the account (address) and the same holds for contracts. I did not delve into detail in this, but did not see any issue there. Main this is that transactions can be added easily. For example, there is nothing like Sha^10000(block) concatenated to it.

Anyway, once this is done, it is necessary to compute the ethash itself, ie hash that, when taken as value, is less than set target.

Now, bitcoin has a specific field in the block used for varying input to the difficulty hash.

Ethash also has this, but I tis realized slightly differently. For every 30 000 blocks, there is a constant set of 32 bytes sequences, this dataset is called DAG. This specific block that bitcoin has for hash calculation is mapped to subset of the DAG in Ethereum case and the input into the hashing function. Given index j, it is impractical to the to compute sequence j from DAG, so the only way to get the input into the hash is to read it from the DAG file, that is precomputed and here lies the key to this method. DAG is huge and does not fit any specialized FPGAs or ASICs for massive parallelization, so the bottleneck is effective memory access. In any case, once the hash is found, it is easy to verify, given access to DAG.

Attack:

Instead of varying over DAG subsets, we instead hold it fixed and create a bogus transaction, where some account under miner’s control sends some minimal amount to a random account R(random number) and use R instead as the source of randomness for generating the hash. It should provide enough entropy and if not, it is always possible to extend this entropy by adding another transaction. Presumably, this allows miner to return to the bitcoin style mining and be much more effective, as he is no longer limited by DAG memory access bottleneck.