EPoW: Ethic Proof of Work

November 2018

Authors:

Giovanni Antino

Francesco Pasetto

AiliA SA
Weinberghöhe 27
6300 Zug - Svizzera
mail: info@takamaka.io

AiliA SA
Weinberghöhe 27
6300 Zug - Svizzera
mail: info@takamaka.io

Abstract

The huge distributed computing structure set up by those who believed in the ethereum proof could be reused, always with the same principle, as a useful proof. In this document we aim to provide a technical solution able to make the PoW useful and ethically acceptable without disrupting the mechanisms underlying the PoW.
The document will present a solution to exploit the immense parallelism used by the miners for the resolution of the block applied to the search for solutions on a very large category of real problems. In this test the hash of the block will be the solution to the problem.
The idea is to be able to maintain this ecosystem:

- miner
- mining pool
- structures e rig

AiliA SA
Weinberghöhe 27
6300 Zug - Svizzera
mail: info@takamaka.io

Problem Statement

One of the main problems that needs to be faced in the construction of a blockchain is the selection of the voting mechanism for the choice of the next block.
At the moment the main alternatives are the following:

- Proof-of-Work (PoW): successfully used by Bitcoin[1] since 2009, currently used in the Ethereum blockchain[2]. Based on calculation power. It has a high expenditure of energy. Depending on the implementation, it tends to create concentrations of power in the larger mining farms. This process is accentuated if, as in Bitcoin, the required hardware is specific (ASIC).
- Proof-of-Stake (PoS): Currently used by Cardano, based on the possession and "risk" (Stake) of the cryptocurrency.it is Functional even though has been tested for less than the PoW.
- Proof-of-Burn (PoB): experimental proof that provides, as price, the "burn" (destroy) of part of the currency with non-redeemable transactions in order to be chosen as miners. Experimental.

The PoW has proved to be safe but presents some questionable ethical aspects. The idea behind this proof is neutral: you take a difficult problem, a problem for which you can not directly calculate a solution, and try to solve it so that it generates a certain result.
In the case of Bitcoin, it includes a predetermined number of zero (difficulties) as the beginning of hash.
The choice of the challenge is totally arbitrary, in this case based on the simplicity of implementation and the easy verification principle. This made the Bitcoin proof prone to be calculated from specific hardware within specialized farms. The machines of these farms are able to calculate only and exclusively this type of problem. All the energy used goes in the search for a sequence of zero and the infrastructure would become useless if it were to stop undermining Bitcoin.
To avoid the repetition of the concentration of calculation, and therefore of power, in the hands of a few owners of low-cost energy and hardware, Ethereum opted for a test that required generic hardware (can also be used for other purposes and not specialized) and a "memory hard" challenge. This has prevented the birth of ASIC farm by allowing all owners of a recent GPU to participate in the challenge. One of the advantages of this type of mining is that, if you stop mining ethereum, the hardware would still be valid. But the problem of the challenge remains. It is always the calculation of a "modified sequence of zero". This is why Ethereum is trying to replace the PoW with a new PoS[3].
An alternative could be to work on real and existing problems. But how?

Solution

To proceed on this path, you do not need a problem but a category of difficult problems. These problems must have the following characteristics:

[1] https://bitcoin.org/bitcoin.pdf capitolo 4
[2] http://ethdocs.org/en/latest/mining.html?highlight=proof nella sezione relativa al mining [http://ethdocs.org/en/latest/mining.html?highlight=proof#what-is-mining]
[3] passaggio alla proof of stake http://ethdocs.org/en/latest/frequently-asked-questions/frequently-asked-questions.html?highlight=proof#what-s-the-future-of-ethereum

- Difficulty (**NP-HARD**):

in order to exploit the computing structure and not work on problems whose solution can be easily found with a greedy solution method in reasonable time

- **Solution in compact standard format**:
  - the solution to the problem must be compact (reduced footprint for archiving)
  - easily distributable on a p2p network
  - linked to the block so that its alteration leads to block corruption and vice versa

- **Standard problem form**:
  - the problem must be modelled according to fixed rules
  - the solution algorithm must remain unchanged for each problem
  - the optimizations during the verification or calculation process must not compromise the safety of the structure
- **P-Time verification**

Core Idea

In the computability theory there is a non-deterministic Turing machine[4] model which is exactly what a mining pool approximates. Viewed in its entirety, a mining pool is a machine that can verify a huge number of solutions with an extreme degree of parallelism.
How do we tackle the resolution of the problem? How is it possible that all problems have the same solver?
Let's face the question step by step. First of all, the format of the problem.

Boolean satisfiability

One of the most common forms of problem modelling in computational complexity theory is SAT.
In this case 3-SAT [5]CNF.
Why 3-SAT?

- It is easy to model problems in this form
- It lends itself easily to formal demonstrations
- The input form is standard
- The verification of a solution is in P-Time[6]
- Protein Folding[7], Circuit Minimization[8], 3-Clique they all have reductions in SAT

[4] Non Deterministic Turing Machine
[https://en.wikipedia.org/wiki/Non-deterministic_Turing_machine#Non-Deterministic_Turing_Machine]
[5] 3-satisfiability
[https://en.wikipedia.org/wiki/Boolean_satisfiability_problem#3-satisfiability]
[6] Volendo rinunciare ad ogni ottimizzazione in fase di verifica il costo è lineare nel numero di clausole.
[7] Lynx: A Programmatic SAT Solver for the RNA-Folding Problem
[https://link.springer.com/chapter/10.1007/978-3-642-31612-8_12]
[8] SAT-Based Algorithms for Logic Minimization
[https://pdfs.semanticscholar.org/67fd/d3e15a500ebd2b10c6175ee344274d4e1d6a.pdf]

- There is an automatic scheme that reduces SAT to 3-SAT[9]
- There is an implementation scheme that uses even partial solutions and then improve them → there are many studies on the possibility of solving SAT incrementally[10]
- It is possible to efficiently model SAT on GPU [11]
- In this case it is possible to model a verifier on GPU

For the reasons described above, 3-SAT is ideal for modelling the problem because, from an input in standard form and an always standard output, the solutions are usable even if partial and become suitable for GPU modelling.

Valid modelling problems

As reported in the Boolean Satisfiability paragraph there are a practically infinite number of problems:

- difficult (NP-HARD)
- useful:
  - *protein folding* for the research of new drugs / chemical compounds
  - *circuit minimization* for hardware optimization

that can be used as an input set.
The problems will be part of an archive maintained in p2p, synchronized by the clients. A problem can be proposed, verified in polynomial time as 3-SAT validity and accepted by the miner.
Who keeps the mining client (eg the ethereum community) could sign a set of valid problems to be accepted and be used as a basis on which to calculate.

It is always possible to propose new problems as long as they are sufficiently complex (with a high number of clauses).

Calculation of the Difficulty

The calculation of the difficulty is correlated to the problem but does not require substantial changes to the current regulation algorithms. How to proceed?
First we define the difficulty of a block in the relationship:

$$Difficulty = \frac{Clause_{risolved}}{Clause_{total}}$$

This will result in a number between 0 and 1where:

- 0 represents the unsolved problem
- 1 complete resolution of the problem

At this point it will be enough to request that the number of clauses resolved is greater, to increase the difficulty, or less, to reduce it.

[9] SAT to 3-SAT
[https://www.ida.liu.se/opendsa/OpenDSA/Books/Everything/html/SAT_to_threeSAT.html]
[10] Incremental SAT and applications
[http://www.cril.univ-artois.fr/~audemard/slidesMontpellier2014.pdf]
[11] CUD@SAT: SAT Solving on GPUs
[https://users.dimi.uniud.it/~agostino.dovier/PAPERS/CUDAatSAT_JETAI_DRAFT.pdf]

AiliA SA
Weinberghöhe 27
6300 Zug - Svizzera
mail: info@takamaka.io

Modelling the input

Given formalization of the problem in 3-SAT CNF:

$$P = (X_1 \vee \neg X_2 \vee X_3) \wedge (X_4 \vee \neg X_1 \vee X_2) \wedge (X_5 \vee \neg X_6 \vee X_7) \wedge \ldots \wedge (\ldots)$$

For a total of n variables and a sequence of bit $B = b_1 b_2 \ldots b_n$

$$\forall\, X_i \in P \;:\; X_i = b_i$$

I assign to the first variable X and to all its successive instances the first truth value of the sequence B. When X instances are exhausted, I look for the first unassigned X_{+1} instance and continue in the same way with the value b_{+1}. I will stop when all the variables have been assigned.

The creation of the B-sequence

To generate the B-sequence I take the root node N_{Root} of the merkle tree of the current block and apply::

$$B = shake(N_{root},\ n)$$

where n = number of P variables

SHAKE256(M,d) is a special hash function belonging to the SHA-3 category. This function takes in input an arbitrary length M-sequence of bits and produces a sequence of length d of output bits[12]. It is a hash function with very high resistance even to attacks on quantum computers[13] and an excellent pseudo-random function.

Notes on the block hash and the minimum length of SHAKE output

For maximum strength it is necessary for d to be at least 768.
SHAKE (M, 768) is also recommended for creating the merkle tree and all the hashes necessary for securing the blocks.
If the input is less than 768, I proceed by taking the first p bits of the hash.

Block structure and block enveloped

In this algorithm, the block structure is quite unrestricted. It must contain:

- the hash of the previously enveloped block
- a nonce to alter the hash
- A unique identifier of the problem that has been solved (eg the hash of the problem that has been solved)

Other fields like coinbase and timestamp will depend on the type of blockchain to which this EPoW will be applied.

Envelopment

An important note. An envelope containing the underlying couple will be attached to the block:

$$(Clause_{resolved},\ Clause_{total})$$

[12] SHAKE256 [https://en.wikipedia.org/wiki/SHA-3#Instances]
[13] Quantum Attack
[https://en.wikipedia.org/wiki/SHA-3#Security_against_quantum_attacks]

AiliA SA
Weinberghöhe 27
6300 Zug - Svizzera
mail: info@takamaka.io

This does not weaken the block, but only serves to speed up the indexing. In case of a menace (false envelope), when the client is checked, the block will be discarded.

Block and envelope complete structure

Envelope	E_p
Clause Risolved	INT
Clause Total	INT

Block	B_p
Hash of the previous block	SHAKE256((E_{p-1} , B_{p-1}), 768)
Hash of the problem	SHAKE256(P, 768)
Merkle Root	SHAKE256(<DATA>, 768)
Nonce	INT(256)

This is an extremely minimal structure. Depending on the type of blockchain, it may be necessary to introduce other fields such as timestamps, block numbers ... to speed up execution.

Do not add anything to the envelope! this is just a declaration that does not have to add anything that can not be calculated to the problem!
The envelope is then protected by the following block.

Problems solved

If a block were to solve 100% of a problem, this would be removed from the list of problems to be solved.

Conclusion

The aim of this structure is to employ the enormous parallelism of the mines to generate solutions with a high degree of quality to be passed to incremental algorithms.
In many studies, the algorithms (as is clear from the articles linked in the Boolean satisfiability section) are based on a high-quality random solution.
The need for the hash of the block to be the input on which to calculate the goodness of the solution is imposed to guarantee the immutability properties typical of a blockchain.
Instead of a search for zeros, the problem is perceived as a search for positive solutions on a set of clauses.
The fact that the effort is focused on the verification, it allows to keep the structure of the solver simple by applying, in fact, the definition of a non-deterministic polynomial Turing machine.
The SHAKE function is, in itself, an excellent pseudorandom that relives the client from having to keep track of the solutions already tried (it would mean colliding) by reducing the memory footprint of the solver.
All the optimizations that allow to reduce the number of clauses in the verification phase, recalling them from previous executions are welcome. They do not undermine the security of the problem and do nothing but contribute to raising the degree of difficulty.
As for the structure of the problem, the degree of difficulty represents the quality of the solutions. The higher the difficulty, the better the solution to be proposed.
The structure is immune to the trivialization of the problem since the effort is not towards the resolution but the verification.
The use of the SHAKE function, structured as mentioned above, allows to compress very complex inputs in the few KB needed to represent the root of the B_p block.

Ethical solution?

The mining of real problems puts an end to the waste of electricity but does not make the thing automatically ethical. What makes the thing interesting is that the result of the calculation is public. Obscuring the solutions to the proposed problems would compromise the security of the blockchain by making it impossible for the clients to verify the block.
The implementation we are going to propose has the following characteristics:

- Problems are in standard form, easily transferable to resolvers that require starting from an existing partial solution.
- The block, which represents the solution, is inherently public.
- The problem pool is multiple and rotates, in order to allow a wide search.
- In the event that a problem is 100% solved, it is removed from the pool so that no further work is done on something that is not needed.
- Any subject can propose a problem. This will allow people, universities and companies, that do not have access to large computing resources, to benefit from the power of the miners and, at the same time, secure the network with new problems.
- The main criteria in selecting problems is their complexity. Only objects that are really difficult to solve must be inserted (this requirement is to not trivialize the

effort. Even if a problem is trivial, security is not compromised as long as the input is above 768 bits).

- The system lends itself to the implementation of a web portal (block explorer) that allows to sort the problems by category and quality of the solution.
- It is possible to access the description of the problem and download its formalization as well as the solution since each miner must have both in order to work and validate the blocks (everything is public).
- The problem and its solution are part of the distributed architecture: they do not rely on a central portal but there is a copy in every miner. This guarantees a fair and indiscriminate access to the solutions without the possibility of censorship.
- It will be possible to incentivize the miner to work on a specific solution proposing a bounty to make a problem more inviting. This will be done via a smart contract. The presence of this incentive does not undermine the ethics of the structure, because the solution remains public, but helps to keep up the development of the calculation network by guaranteeing remuneration to those who find the solution required with the quality level indicated.
- The use of miner with prizes for:
    - mining
    - fee
    - bounty

will help in the distribution of the coin to subjects from all over the world without going through a single "reseller".