A Prunable Blockchain Consensus Protocol Based on Non-Interactive Proofs of Past States Retrievability

Alexander Chepurnoy
IOHK Research
Sestroretsk, Russia
Email: alex.chepurnoy@iohk.io

Mario Larangeira
IOHK Research
Tokyo, Japan
Email: mario.larangeira@iohk.io

Alexander Ojiganov
Saint-Petersburg National Research University of Information Technologies, Mechanics and Optics
Saint-Petersburg, Russia

Abstract

Bitcoin [1] is the first successful decentralized global digital cash system. Usefulness of the mining process requiring a lot of computational resources to be wasted, though, remains disputable. One of possible alternatives for useful Proof-of-Work schemes, Permacoin [2], is using non-interactive proofs of a static dataset retrievability thus providing a mechanism to store a huge dataset being spread across miners collectively.

In this paper we present a new consensus protocol for Bitcoin-like peer-to-peer systems, where the right to generate a block is given to the party which provides non-interactive proofs of retrievability for the past state snapshots. This Proof-of-Work scheme has better resistance to specialized hardware than Bitcoin and Permacoin. Unlike blockchain systems being used today, a network using the protocol of ours is safe if majority of nodes are rational in terms of storing full blocks. By using that we also show that one useful application of our protocol is the construction of a prunable blockchain system without a security loss.

1. Introduction

Bitcoin whitepaper [1] defines a way a common ledger could be maintained within a trustless peer-to-peer network by using moderately hard computational puzzles [3] and the blockchain datastructure. Since then a lot of research has been done about Bitcoin and alternative systems. Nevertheless, there are still many open problems in the field, and performance is one of the most crucial [4]. Another threat to blockchain-based systems of today is the lack of reward for any activity supporting network other than for block generator self-election via finding a partal hash collision. In particular, there is no reason for nodes in network to store all the blocks since genesis until few newer ones.

In this paper we present a consensus protocol alternative to the Proof-of-Work used in Bitcoin. In our protocol a participant is rewarded for archiving only a few state snapshots, hence the term "prunable", amongst n states a network aims to store collectively. That is, if a miner is storing a state snapshot for height h, then on a new block appears she will need to replace it with a snapshot for height $h + 1$. Thus a miner, as the well as whole network, needs to store some number of blocks as well. We show how a scalable and safe trustless blockchain based ledger system could be built from our protocol.

1.1. The Blockchain Model

A blockchain could be seen as a linked list where an element (a *block*) is a tuple $(\Delta S_i, \Delta C_i)$, where ΔS_i is a *transactional state modifier* and ΔC_i is a *consensus state modifier*. The tuple of state modifiers could be applied to a state which results another state whenever the modifiers are valid. We denote a state modifier application by $\diamond$, then $(S_i, C_i) = (S_{i-1}, C_{i-1}) \diamond (\Delta S_i, \Delta C_i)$. Every network participant knows the pre-defined transactional and consensus state (S_1, C_1) resulted from the *genesis block*. Then each participant knows exactly the same state $(S_i, C_i) = (S_1, C_1) \diamond (\Delta S_2, \Delta C_2) \diamond \cdots \diamond$

$(\Delta S_i, \Delta C_i)$ if all the blocks are the same. The order of modifiers is defined via immutable link from each modifier to the previous one, where the first modifier in the history must be linked to the genesis state. For a block $(\Delta S_i, \Delta C_i)$ we refer to i as to *height* of the block.

Consensus state modifier changes rules on block validation not related to transactions stored in it (for example, it contains *difficulty* explained in the next subsection). A transactional state modifier being atomic in terms of its application itself contains a sequence of transactions. We will ignore this fact further. We will refer to *state snapshot* or just *state* as to transactional state after block application.

1.2. The Consensus and the Mining Lottery

The Bitcoin blockchain is generated in a worldwide peer-to-peer network without any central authority. In such an environment, the next block determination process requires protection against Sybil attacks.

The Proof-of-Work mining process [1] eliminates Sybil attacks and also enforces a rational miner to choose a single version of the history out of possible options.

The Proof-of-Work mining process could be seen as a lottery as described in [2]. A mining software iterates over changeable blockheader field values until it finds a solution to $hash(blockheader) < difficulty$[1], where *difficulty* is a publicly known value. Iterations could not be precomputed because of the epoch-dependent unique *puzzle ID* which is known to all the participants and stored into a block header. Each iteration result could be interpreted as a *ticket*, and a ticket could be the *winning* one for an epoch if the predicate above holds, giving its owner the right to generate the new block.

1.3. The Permacoin Cryptocurrency

The key idea behind Permacoin [2] is to make the mining process dependent upon the storage resources rather than the computational capabilities. In order to achieve the condition that every miner stores a random subset of a known static dataset, Permacoin requires that the dataset is generated by a trusted dealer.

The trusted dealer also announces the root hash value of a Merkle tree data structure [5] which is built on the top of the dataset. Then, instead of iterating over changeable blockheader parameters, a miner iterates over some random nonce value to find chunks whose indexes are dependent on the value. A ticket contains the nonce value and the chunks with Merkle paths for them being attached.

The winning ticket gives the right to generate the next block and is to be included into the block.

1. In general "hash" means a cryptographic hash function, whereas "blockheader" and "difficulty" are defined analogously as in the Bitcoin protocol jargon.

1.4. The Blockchain Storing Rationale

In Bitcoin Core, the reference implementation of the Bitcoin protocol [6], a node is storing all the blocks with all the transactions since the genesis block. In addition, the current state snapshot in the most compact form, which is *unspent transaction outputs list*, is stored as well.

The main benefit for the network for storing the blockchain is the ability for new nodes to download all the blocks since the genesis and process them revalidating all the transactions ever included to the blocks. Being useful for the network, storing all the full blocks is the altruistic activity for a blockchain holder.

A *rational* participant can have all the full node merits with all the blocks and state snapshots being deleted, except of last few ones needed to perform possible rollbacks. A node could become rational by switching to an implementation of a Bitcoin protocol supporting that. Then if the network has no altruistic participants the only way to get into it is to download the current state snapshot and trust it.

2. Related Work and Preliminaries

This section is divided in three parts: (1) We start by reviewing a few related works. (2) Next we review the motivation for a new protocol. (3) Here, we recall the Permacoin protocol which our construction is based.

2.1. Related Work

2.1.1. Ethereum. Ethereum [7] has state representation as an authenticated data structure being fixed by the protocol with a root hash to be included into a block. But as there is no any incentive to store past state snapshots, the state proof in a block header is useful only to help light nodes to get state elements from full nodes along with Merkle paths and does not solve the storing rationale problem stated in the section 1.4.

Ethereum is contending with specialized hardware by using Ethash Proof-of-Work algorithm, which is proposing to use pseudo-random dataset generated

from blockheaders in the past in a mining process. Usefulness of EthHash is the same as of Bitcoin. The disadvantage of the algorithm is heavy validation.

2.1.2. Cryptonite. Cryptonite [8] utilizes mini-blockchain scheme [9] to prune full blocks before a constant numbers of last ones, thus having proof chain of length h_{pc} (the chain of block headers only), a state for a height h_{pc} and full blocks for heights from $h_{pc}+1$ till current. It is not clear from the paper how the protocol enforces this scheme.

2.2. The Motivation For a New Protocol

Next, we detail the four main points which our protocol addresses.

2.2.1. Incentives to Keep a Full Node. In Bitcoin, the only rewarded activity is the iteration over values for certain fields in the block header. However, if a minin software includes transactions into the block, it needs to validate them, which can be carried by presenting the current state snapshot. No storing blocks is needed in order to mine. Our particular goal is to develop a protocol providing incentives to run a node which stores sufficiently enough amount of blocks for network viability, thus making the network safer in the long run. Furthermore, we denote such a node a *full node*.

2.2.2. Solving The Blockchain Storing Rationale Problem. A full node is a node which stores a state snapshot for current moment of time and *sufficiently* enough blocks in the past. For the blockchains of today "sufficiently" means *all*, it is enforced by an implementation as stated in the section 1.4.

As the growth of the blockchain is not bounded, in the long run such a system can survive only if the storage and processing power of an ordinary computer grows not slower than the storage and processing requirements of a blockchain system. As a practical consequence, it is hard to say whether a new full node will be able to process new blocks few years after, or whether it will be possible to rebuild the state by processing a blockchain since its genesis block in case of need.

As storing enough blocks is the activity with no reward, eventually most of full nodes will purge blocks from their disks except last ones having non-neglibible probability to be rolled back. This means a system where blocks are needed but not stored due to a practical impossibility. We aim to develop protocol rewarding for storing collectively some number of blocks and past states.

2.2.3. A Prunable Blockchain. Considering the storing rationale problem has been solved means some number of blocks and states are archived by the network in order to mine new blocks. This number n could be sufficiently large to assume any rollback caused by a fork could be of negligible depth in comparison with it. For example, $n = 100,000$ blocks gives approximately 694 days of history to be stored if a block is generated every 10 minutes on average. Any rollback imaginable is about negligible depth in comparison with that.

In such a system we can assume overwhelming majority of nodes (except of tracking tools and other special applications) to be rational so to remove blocks not needed for mining having a freedom of choice given. A rational node has predictable storage resources consumption. A network of rational nodes is also sending less blocks, and bandwidth being saved could be repurposed for other tasks improving system performance.

2.2.4. Fast Trustless Bootstrap. A new full node in Bitcoin needs to download and process all the blocks since the genesis block. This results into an unreasonably long and resource-consumptive initial processing phase. In order to reduce the burn of this phase, a trusted Bitcoin state snapshot generated by a notable community member could be downloaded [10]. This approach reinstates trust-related issues solved by Bitcoin.

In Ethereum the state proof is constructed via of a Merkle-Patricia tree root hash value stored into the block header. Then a party joining the network could download a state from n_e blocks earlier and then apply n_e blocks to it in order to verify the resulting state against the proof stored in the blockchain.

There is a problem with the Ethereum approach as well: as there is no reward for storing previous states, it is unlikely that the new node could find a state from a week or two months ago amongst its peers. Therefore having a protocol which provides an incentive to store sufficiently enough states within the network is also solving the bootstrapping issues in a trustless manner.

2.3. Permacoin Issues To Fix

Our protocol is derived from Permacoin [2]. During careful analysis we have found few possible threats and open questions in the protocol.

2.3.1. Trusted Dealer. The Permacoin consensus requires a trusted dealer to encode a huge dataset and to build a Merkle tree on top of that. We would like to eliminate trusted dealer notion at all.

2.3.2. Vulnerability to Specialized Hardware. As a subset of a static dataset a miner stores is static as well, it is possible to put in into dynamic or static RAM and then connect specialized hardware to the memory. With such a scheme companies could get the same advantage over individual miners as in Bitcoin with the same degree of centralization we would like to avoid.

2.3.3. Storage Optimization Strategy. It is possible to iterate over miner's public key aiming to store less data segments (see "Setup" formula in "A simple POR lottery" of the Permacoin paper [2]). This could give even more advantage to a company with vast computational resources over individual miners.

3. The Protocol

Our protocol is designed to create an incentive for the miners to store collectively the last n states and blocks, where each miner stores at least k states and also $\frac{k \cdot n}{k+1}$ blocks on average in order to generate a block. The protocol reminds the "Local-POR lottery" algorithm in the Permacoin paper [2].

Before detailing our construction, it is convenient to introduce the notation that will be used from this point.

3.1. The Notation

Here, we introduce the notation which will be used further in this study.

We denote by *hash* a regular cryptographic hash function with a uniformly distributed output. Furthermore, given two strings z and w we denote by $z \cup w$ the string which results from the concatenation of z and w.

An arbitrary array of k values is represented by $u[i]$ where $1 \leq i \leq k$, whose the minimum value, among all possible k, is denote by $min(u[i])$.

Given a state S, we denote an i-th element of it as $S[i]$ and a Merkle path for $S[i]$ as $\pi(S, S[i])$.

We assume all the miners know value puz specific for a current blockchain state, and that value could not be predicted in prior by a computationally bounded party.

We assume that mining rewards could be given to an owner of a public key k_{pub}. We do not specify a signature scheme for k_{pub}.

3.2. The Fixed State Representation

In Bitcoin, the state representation is not fixed by the protocol in any way. A full node implementation usually stores a list of unspent outputs and also some node-specific additional information. By applying valid transactions from a new block, a node software takes unspent outputs out of the list and puts there outputs from the transactions [1].

Abstracting the Bitcoin-like model, a state could be represented as a list of *closed boxes* of size n_S. Each box has a value associated with it. A transaction contains keys to open n_k boxes and create n_b new closed boxes. The resulting set has size $n_S - n_k + n_b$ after applying the transaction to it.

We assume that a closed box has some standard binary representation. Consider we put boxes in a deterministic way into a list of size $L = 2^h$, filling positions not taken by boxes with special value $\perp$ also having some standard binary representation. Then we can compute hashes of list elements to build then a Merkle tree of height h upon the hashes. A root hash value of the tree and its height h is to be included into a corresponding block.

Note that our construction of a state representation is fixed and is a part of the protocol, unlike Bitcoin.

3.3. The Setup

Consider a blockchain whose length is h_c and miner Alice having a public key k_{pub}. She is willing to start mining at that moment in time. For every $i \in [1 \dots k]$ she calculates random heights as follows:

$$u[i] = \begin{cases} (\mathit{hash}(k_{pub} \cup i) \bmod n) + (h_c - n), & \text{if } n \leq h_c; \\ \mathit{hash}(k_{pub} \cup i) \bmod h_c, & \text{if } n > h_c, \end{cases}$$

where n is the of state snapshots to be stored collectively in a sufficiantly large network.

She stores a subset of states $S_{k_{pub}} = S_{u[i]}$. All the $u[i]$ values must be unique, otherwise no valid block could be generated. We enforce such a requirement to prevent malicious iteration over public key space to find as much repeating $u[i]$ values as possible. When a new block at height $h_c + 1$ arrives, Alice needs to recalculate at least k states. Thus Alice must store blocks since $\min(u[i])$ as well.

3.4. The Scratch-off

For a scratch-off attempt seeded by a chosen seed s a miner iterates over the states $S_1, \dots, S_k$ to get random boxes for each of them.

$$e_1 \quad := \quad S_1[\mathit{hash}(s \cup k_{pub} \cup puz) \bmod L_1].$$

For *i = 2,3,...,k*:

$$e_i \quad := \quad S_i[\mathit{hash}(e_{i-1} \cup k_{pub} \cup puz) \bmod L_i],$$

where states $S_1, \ldots, S_k$ are represented as lists of boxes having sizes $L_1, \ldots, L_k$ accordingly, as described in Section 3.2.

A ticket is tuple $(k_{pub}, s, (\forall i \in 1 \ldots k, \{e_i, \pi(S_i, e_i)\}))$.

3.5. The Winning Ticket

Like Bitcoin, we introduce epoch-dependent *difficulty* parameter D, a positive number known to every participant. Then we distinguish winning tickets from others by using the following formula:

$$\mathit{hash}(s \cup e_k) < D.$$

3.6. The Ticket Validation

In order to validate a ticket, one should replay the scratch-off algorithm (from Section 3.4) with the value s provided in the ticket. The validation procedure requires $k+k\cdot(h_1+\cdots+h_k)+1 = k\cdot(h_1+\cdots+h_k+1)+1$ hash calculations, where $h_1 \ldots h_k$ are Merkle tree heights for the states the ticket is proving retrievability for.

3.7. The Formula for the *puz* Value

The puz value in the protocol algorithm must not be predictable before a beginning of a new epoch, so before a winning ticket generation, and also must be resistant to iterations.

Based on these requirements, the best option is to have it dependent on the s value included into the last block, as s value is costly to replay. Therefore, by denoting puz and s values for last epoch (block) with $lpuz$ and ls accordingly, the formula for puz value for current epoch is as follows:

$$puz := \mathit{hash}(lpuz \cup ls).$$

4. Discussion of The Protocol

In this section we analyze the property of our protocol regarding a well known model and also further relate our constructions with potential issues.

4.1. The Protocol Properties

In order to study the protocol properties we refer to the model in [11]. The formula for the winning ticket in our protocol (Section 3.5) precisely corresponds to the **validblock** predicate in [11, Section 3].

Our proposal does not add anything to break the compatibility with all the stated algorithms and their analysis, therefore our protocol has the same properties as the Bitcoin protocol in that model.

4.2. The Possible Threats and Attacks

4.2.1. Specialized Hardware. The best strategy for a miner to earn more in our protocol (as well as for Permacoin) is to put a subset, required for ticket generation, into the random-access memory. Later the miner connects as much as possible computational units with specialized memory architecture in order to parallelize iterations over the s values.

We argue that our proposal has better protection against such a miner because the dataset is not static, therefore a general-purpose CPU is needed to be connected to the RAM in order to update the states.

For a better protection of our scheme we offer to adopt techniques from other Proof-of-Work schemes, such as asymmetric Proof-of-Work schemes [12] and non-outsourceable puzzles [13].

4.2.2. The State Set Optimization. Due to the uniqueness requirement for the $u[i]$ values, it is not possible for a miner to iterate over the public key values with the goal to minimize the set of states to store. However, a miner could iterate over the public keys to find states closer to the current moment in time in order to store less blocks. We consider this as a minor issue.

4.2.3. The ε-consensus Attacks. Paper [14] describes the ε-consensus attacks where a miner includes heavy transactions into the block generated by herself (for free) in order to enforce other miners to spend some time to verify a block or to skip validation. For the protocol of ours such attacks could be more harmful as a miner needs to apply at least k blocks, not one. However, a mining software could apply blocks in the past in advance, in this case ε-consensus attack is no more harmful than in other systems.

4.3. The Bootstrapping Scenarios

Consider that Alice is joining a network. In the Bitcoin network she would have two options: (1) download and process all the blocks since genesis or (2) download the trusted state snapshot for a predefined moment in time and then apply blocks to it since that point. With our protocol she can choose one out of many possible bootstrapping strategies and none of them includes a central authority. Next, we provide some examples:

4.3.1. Fast Bootstrapping. Alice asks her peers to find the most recent state, then downloads it and applies a minimal number of blocks to it. However if only a single peer holds the state, this strategy is vulnerable to communication abort.

4.3.2. Reliable Download. Alice finds the most popular state snapshot and downloads it in parallel from many peers. Then she downloads and applies blocks to it.

4.3.3. Fast Paranoid. Alice preferably looks for popular states from long time ago. For example, if $n = 200,000$ and a new block is generated every 10 minutes on average, then a network collectively stores states snapshots for 3.8 years, and Alice can download a popular state older than 6 months ago. It is hard to imagine a fork lasting for such a long time.

4.3.4. Classical Paranoid. As in Bitcoin, Alice can download all the blocks since genesis and process them. This strategy fails if all her peers are rational.

In addition to the strategies above, many others are possible. Note that Alice needs to download block headers in prior to check chain validity by checking tickets and also to get state proofs.

4.4. Archiving Guarantees

Consider Alice is joining a network at height h_c and sees p peers, all of them are rational miners. How many full blocks could Alice find asking from her peers?

For simplicity, we ignore the fact that $u[i]$ values are uniqueness for a peer. Then p peers have $p \cdot k$ (possibly duplicate) values over n integer values. By using order statistics the expected minimum value is $h_{min} = (h_c - n) + \frac{n}{p \cdot k + 1} = h_c - \frac{p \cdot k \cdot n}{p \cdot k + 1}$. Alice can download state snapshots and blocks since h_{min} on average from at least one peer.

4.5. A State Exchange Protocol

Consider Alice joining a network. She chooses a state snapshot at height h_d to download from her peers. The problem is, if a new block arrives during downloading the snapshot, her peers are to replace the snapshot with another one of height $h_d + 1$. To avoid this problem Alice needs to ask her peers to store the snapshot even if it is not needed for mining. Alice can propose a reward to her peers for doing this, thus some fair protocol is needed. We are not going to specify it in this paper.

5. Conclusion

We have presented a modification to Bitcoin and Permacoin that is intended to recycle computational resources in order to maintain decentralized database of ledger state snapshots. The protocol of ours has the same properties as Bitcoin protocol. It is better protected against specialized hardware usage that Bitcoin and Permacoin. The state snapshots archiving providing safety against rational participants removing full blocks not needed after processing. This feature of our protocol addresses the concerns around the viability of the system when regarding the increase in the amount of transactions within the network.

Acknowledgments

The authors would like to thank Andrew Miller for a discussion on Permacoin and Roman Oliynykov for proofreading.

References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” pp. 1–9, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, “Permacoin: Repurposing bitcoin work for data preservation,” in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 475–490.

[3] A. Miller and J. J. LaViola Jr, “Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin.” [Online]. Available: https://socrates1024.s3.amazonaws.com/consensus.pdf

[4] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. Gün, “On scaling decentralized blockchains.” [Online]. Available: http://fc16.ifca.ai/bitcoin/papers/CDE+16.pdf

[5] R. C. Merkle, “A digital signature based on a conventional encryption function,” in *Advances in CryptologyCRYPTO87*. Springer, 1987, pp. 369–378.

[6] “Bitcoin Core Code Repository.” [Online]. Available: https://github.com/bitcoin/bitcoin/

[7] “Ethereum: A Secure Decentralized Generalized Transaction Ledger.” [Online]. Available: http://gavwood.com/Paper.pdf

[8] “The Cryptonite Project Homepage.” [Online]. Available: http://cryptonite.info

[9] “The Mini-Blockchain Scheme.” [Online]. Available: http://cryptonite.info/files/mbc-scheme-rev2.pdf

[10] “Bitcoin Blockchain Data Torrent.” [Online]. Available: https://bitcointalk.org/index.php?topic=145386.0

[11] J. Garay, A. Kiayias, and N. Leonardos, “The bitcoin backbone protocol: Analysis and applications,” in *Advances in Cryptology-EUROCRYPT 2015*. Springer, 2015, pp. 281–310.

[12] A. Biryukov and D. Khovratovich, “Asymmetric proof-of-work based on the generalized birthday problem,” *Proceedings of NDSS 2016*, p. 13, 2016.

[13] A. Miller, A. Kosba, J. Katz, and E. Shi, “Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 680–691.

[14] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, “Demystifying incentives in the consensus computer,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 706–719.