BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY
Excel@FIT 2021
http://excel.fit.vutbr.cz

Fast, Scalable and DoS-Resistant Proof-of-Stake Consensus Protocol Based on an Anonymization Layer

Bc. Marek Tamaškovič*

Abstract
In this work, we summarized research in the state-of-the-art Proof-of-Stake protocols like Algorand, Tendermint, and LaKSA. We analyzed and summarized their features and issues. Based on the included research we implement a new PoS protocol that mitigates issues with throughput, scalability, and security.

Keywords: Blockchain — Proof-of-Stake — Anonymity — Verifiable Random Function

Supplementary Material: *N/A*

*xtamas01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

There are several interesting Proof-of-Stake protocols in the wild. However, they contain design problems that we want to resolve:

- It is possible to DoS the leader of the round[1] since he is known beforehand. An adversary might increase his chance of being elected as a leader.
- Relatively small throughput. Tendermint and Algorand uses some BFT ideas. Substituting them could improve overall throughput.
- Linkability of peers with their IP addresses. By removing this connection we can create an network anonymity of the participants.

One of the most mature PoS blockchains is Tendermint. It uses a committee that uses a byzantine fault-tolerant algorithm in each of three phases [1]. The committee is fixed and well known in the network, and that inhibits the scalability. Due to known committee, the adversary can DoS each member and change the output of the consensus.

Another mature PoS blockchain is Algorand. It aims to solve issues with Tendermint. Algorand uses a verifiable random function (VRF) to select in one phase but in the second phase, it uses BFT like Tendermint [2]. Due to the first improved phase, the throughput is significantly better. The VRF is a small leap to achieve anonymity inside the consensus layer but the adversary can still overcome it with low effort.

To enhance the current solutions, we introduce three main ideas in our design:

1. Probabilistic selection of leaders for ensuring high throughput with protection against sabotage.
2. Native anonymization of protocol transactions (inspired by onion routing).
3. Force selection function to not follow the order.

Our hypothesis is, that by the implementation of mentioned features, we can gain protection against DoS attacks, high throughput, and anonymity of all participants in the protocol. All ideas are verified by experimental implementation and partially presented in this paper.

2. Related Work

In this section we will describe algorithms that inspired this work, such as Algorand and Tendermint. We compared the included PoS protocols in Table 1. Finally, we will describe Tor as well because it inspired us to use onion routing in this work.

---

[1] Producer of the block.

2.1 Algorand

Algorand is a pure Proof-of-Stake protocol and it is commercially used as a cryptocurrency, but it can be extended for other purposes. Its advantages are very short time to finality, high throughput, and hard to corrupt by an adversary [2]. It uses very small computational power, no matter how many users are connected to the network. The consensus protocol introduced Byzantine Agreement (BA) that works as described below.

Algorand uses verifiable random functions (VRF) [3] to select new leaders. VRF is a public-key version of a keyed cryptographic hash. Only the holder of the private key can compute the hash, but anyone with a public-key can verify the correctness of the hash. Algorand uses VRF to select N members of the committee by letting the peers compute VRF of round randomness, selecting those with results lesser than certain value. Sometimes happens that the VRF will not produce any member of the committee in that case it will delay the block creation. Each new leader must be confirmed by the messages from all members of the committee (similar like BFT) which generates N messages. Each round has two steps:

1. Each member of the committee multicasts candidate for the next block
2. Each member of the committee sends a message with a signature of the winning block.

BA is not pur BFT but is a hybrid since BFT is used on a small group of nodes in the protocol and only one our of three stages of BFT are executed. However even one stage of BFT can cause a significant overhead limiting the throughput of the protocol. If there were any alternatives, it would significantly increase the throughput of the protocol.

2.2 Tendermint

Tendermint is a pure BFT proof-of-stake protocol [1]. Unlike Algorand, it has fixed committee members. A block is selected in a round-robin fashion. This implies that the leader is known in advance to all the nodes. An adversary can use this information to perform DoS attacks against the current leader. This will prevent the leader from publishing a new block. Because of BFT, Tendermint has relatively low throughput. Each round consists of three steps:

1. Propose - a proposed block is broadcasted
2. Prevote - peer validates proposed block and broadcasts its willingness to commit it
3. Precommit - After the peer receives at least 2/3 of the prevote messages, the peer signs the block and commits it in a special commit step.

The last two steps significantly slow this protocol and substitution of these steps would significantly increase throughput. Another aspect of BFT-based protocols like Tendermint is that when more than 1/3 of the network is unavailable, the protocol halts itself and it will wait untill 2/3 of the network can establish consensus.

2.3 LaKSA

LaKSA is derived from Algorand and it adopted ideas from DFINITY and Randhound [4, 5]. It is a proper Proof-of-Stake protocol with some BFT ideas [6]. It is not yet commercially used as the protocols above. It was developed to reduce drawbacks as high reward variance and long confirmation times. It enhances Algorand properties such as lightweight committee voting, it should be more robust and easily scalable than other protocols. In LaKSA, committee members are randomly and periodically sampled to vote for their preferred main chain views [6].

The LaKSA introduced a so-called cryptographic sampling in its consensus protocol that works as follows. Everyone obtains the beacon from the previous block. Based on this block there will be elected leaders and voters. Every node will afterward obtain a number of the stake it can use in that round. If the node has some stake to use in voting, it is called a voter. Otherwise, it is called a verifier. The voters assemble votes and broadcast them to the network. Every verifier will verify the votes and put them on the pending list of votes directly supporting a so-called virtual block. Next, every node will check if it was selected as a round leader. If yes, it will create a block based on the virtual block, and it will broadcast it to the network. Every node that received the newly created block will verify it. If the verification process was successful, then it will be included in the chain.

This protocol has increased fairness than the Algorand. However, the Algorand's leader and committee vary from round to round. In LaKSA it is fixed. Due to this detail, the Algorand may be less secure than the LaKSA. Besides that, the LaKSA is resilient to nothing at stake attack because the committee must accept the new block and it is very hard to create a fork.

The overall limitations of current protocols are summarized in the Table 1.

2.4 Tor

Tor project is an anonymization network project [7] that implements and extends the Onion routing [8]. The main idea behind the Onion routing is to use N Onion routers (OR) to route a message that we want

	Liveness	Throughput	Finality	Scalability
Tendermint	Eventualy every tx will be processed.	peaking performance arround 10000 txs/sec	Blocks are almost instantly finalized	Very hard, there is still the same set of verifying nodes
Algorand	Eventualy every tx will be processed.	3000 tx/sec	Finalized blocks are only those which are located before checkpoint	Simple scalability based on stake transfer
LaKSA	Similar as Algorand	450-1300 tx/sec	Similar as Algorand	Simple scalability based on stake transfer
Casper	depends on the chosen proposal mechanism.	Could not evaluate	Finalized blocks are until checkpoint	Could not evaluate

Table 1. Side by side comparison of PoS protocols and their properties.

to send through the internet. After selecting N ORs the sender will exchange a cryptographic key with them. Afterward, when the sender want's to send the message it will incrementally encrypt the message with each key. Next, the sender will send the message to the first OR. The OR will decrypt the message by the exchanged key and route this message to the next OR and vice versa. When the last OR receives the message, it will send the message to the location the original sender intended. The main advantage of this principle is that the receiver does not know who the sender is. And in the opposite, the sender does not communicate directly with the receiver.

3. Protocol Proposal

This section will propose a new Proof-of-Stake protocol that will have just one leader in a round. This leader will be elected from the randomness of the previous round using a verifiable random function (VRF). Another property that we had in mind during the design is that the protocol must be DoS resistant. We achieved this resilience by implementing an anonymization layer into the protocol itself. At the beginning of the blockchain must be created a genesis block that will hold initial stake distribution among nodes that will start the process of block creation. We assume that every node that is in the genesis block has every crypto-tokens invested in the stake because they want to participate in the protocol. When they get online, the first thing for them is to connect to at least *N* nodes using the anonymization layer (e.g., Dandelion [9]). Afterward, they determine who is the first leader by using the VRF. The VRF is based on the probabilistic selection of a leader based on the stake involved. This principle is iteratively used every round to elect a leader that will publish a block. We assume that this genesis block is hard-coded in every full node and thus they can retrieve and verify the full blockchain retrospectively.

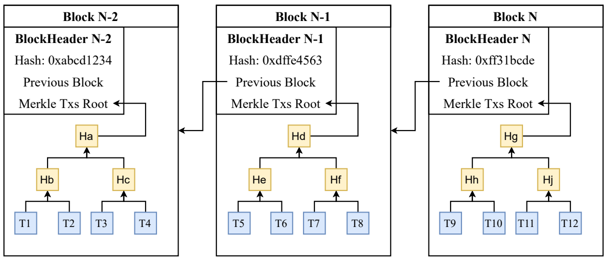

Figure 1. Ilustration of the blockchain distributed data structure. Our proposed protocol has extended header with public-key of the leader that signed the block, alternative leader count, and id of the block

3.1 Data structures

The proposed protocol creates and extends its blockchain, an append-only structure consisting of linked blocks. The block consists of aggregated transactions and a block header created by the leader of the current round. The block header consists of a hash of the previous block header, block id (counter of the blocks), the root hash of Merkle tree consisting of all transactions in block body, a public key of the leader (called coinbase), index of an alternative leader, the randomness of the current block, and the signature made by the leader.

The second part of the block is so-called block body. Inside is a list of all transactions that are in this block. The transactions are composed of destination, the value that the sender wants to send, fee, and signature. One may think that the transaction misses the sender. However, the sender's address will be computed from the transaction itself and the signature.

To achieve this behavior the signing process must be done with a cryptographic function that can include the signer's public key into the signature; for example recoverable ECDSA signatures [10] To store the transactions in the block, we use the list that is aggregated by the Merkle tree. Besides, we use in-memory Merkle-Patricia trie to store the balances and stakes of all accounts, referred to as the global state.

3.2 Normal Operation

The normal operation of the consensus protocol is described bellow. When the round starts, the node resets the round counter of timeout expiration. Then a node checks if it is the leader of the current round based on randomness from the previous round. If the node is a leader for the current round, it will create a new block and broadcasts it afterward.

If the node is not a leader in this round, it will set a timeout timer in which the node expects to receive a block from the leader. When the node receives the block, the node will check the validity of the block, i.e., the block is correctly signed, it was signed by a leader of that round, the block round corresponds to the current round index. After the validation process, the node will execute the transactions above the node's global state and reward the leader. In the end, it will cancel the timeout and save the block. If the node will not receive the block in the specified time, that means the leader is offline and this situation will be solved with increasing round index and setting up the timer again until we get the correct block.

The node can receive transactions as well. In that situation, the node will check the signature of the received transaction and it will check if the source exists in the node's node-list and whether it has enough balance. In the positive case, the transactions will be added to the mempool and gossiped to the other nodes [9]

3.3 Rewarding Scheme

The leaders that create a new valid block will be rewarded. First, we must say that we designed the protocol to have separated stake and balance. This solution is used to have a higher and lower liquidity part of the node's value. To transfer the balance to the stake the node must create a specific transaction. The transfer will freeze the assets for some time to lower the liquidity and to behave like an investment that yields the 'interest rate'. The freeze time must be long enough to penalize the liquidity of the node's crypto assets. The leader will earn a reward value, that will be set to a value that will be experimentally verified. The leader gets the transaction fee from transactions that are included in the block. These fees will be transferred directly to the leader's stake.

3.4 Joining the Protocol

To join the protocol, the node has to buy the balance from any of the existing nodes and then convert it into the stake, which constitutes the semi-permissionless design [11].

3.5 Forks

When receiving blocks there can occur a situation where the received block is valid but it is at a lower height in blockchain than we currently are. If we put in detail the specific height, there may be a situation where the main leader was temporarily offline and the alternative leader published the block. However, the main leader returned back and produced a block and thus created a paralel chain. This behavior is called forking, and it is undesirable since up to some period, the blockchain might be reverted, and thus the finality is achieved only after this time. Therefore, the last few blocks are not stable immediately and can be changed during this time. To overcome this issue, we implemented check-pointing in the blockchain. It means that if the overturning chain is too long, we will not overturn the chain, and instead preserve the current one. The parameter of the maximum chain length allowed to overturn will be the subject of the experiments.

4. Anonymization Layer

The anonymization is realized at the network layer and works as described in this chapter. Details of joining, sending, and relying messages are shown in Algorithm 1. This communication on network level won't be anonymized. It is not an issue, because it will not contain any peer identifier, which makes it impossible to link peer identity (public key) to the node identity (IP address).

4.1 Joining the network

When node N want's to join a network, it must do following steps:

1. New node N gets a list of IP addresses of all nodes from a directory (its trustworthiness may be ensured by multiple ways, e.g. [12])
2. N selects n sets of m peers
3. For each route: Build a circuit consisting of m selected nodes (in chosen order $n_1, n_2, \ldots n_m$). To build a circuit, perform a key exchange with each of the selected nodes, for example by the approach proposed in [7].

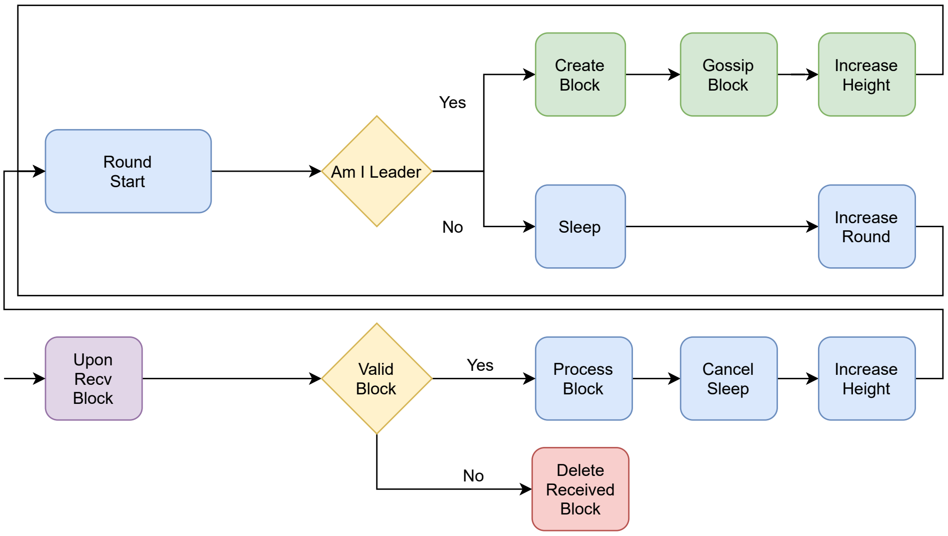

Figure 2. Ilustrated consensus

4. The circuit has been established. N now shares secret key K_i with n_i for each i. Every further communication will be anonymized.

4.2 Sending the messages

Any message P wants to send (broadcast) is sent in the onion routing manner, i.e.:

1. The message is encoded so it can be received by the intended receivers
2. The message M is encrypted with K_i: $K_i(M)$
3. The result of the previous step is encrypted with $K(m-x)$ for $x = m-1$ *downto* 1 and appended with IP of the $(m-x+1)$th peer in the circuit. For example (m=3): $K1(p2, K2(p3, K3(M)))$

4.3 Relaying the messages

- When a peer pn in a circuit receives a message, it decrypts it using the key shared with P. It discovers the identity of p(n+1) and sends the message (that is still encrypted by P using K(n+1)) to it (the message is encrypted by the transport layer).
- If there is no p(n+1), the peer is an exit peer. It decrypts the message and gossips it [7, 8].

Algorithm 1: Anonymization layer interface

```
▷ DECLARATION OF TYPES AND VARIABLES:
    route { $node_{n-1}, node_{n-2}, \ldots, node_0$ },
    node { addr, key },
    addr { IP, port },
    this: the current node,
    routes: list of all routes that will be used in
  anonymization layer,
    Message: constructor of selected messages,
function joinNetwork(n_routes, m_nodes)
    $allnodes \leftarrow getNodes()$;
    $routes \leftarrow pickRoutes(n\_routes, m\_nodes)$;
    for route: routes do
        for node: route do
            $exchangeKey(node)$;
    for route: routes do
        $verifyRouteInitialization(route)$;
function SendMessage(dst, msg)
    for route : routes do
        $relay\_msg \leftarrow Message.Relay(dst, msg)$;
        for node : route do
            $ct \leftarrow \Sigma_{node.key}.encrypt(msg)$;
            $em \leftarrow Message.Encrypted(this.addr, ct)$;
            $relay\_msg \leftarrow$
              $Message.Relay(node.addr, em)$;
        $gossip(route[-1], relay\_msg)$;
function RelayMessage(src, relay_msg)
    $msg\_key \leftarrow findKey(nodes, src)$;
    $msg \leftarrow \Sigma_{msg\_key}.decrypt(relay\_msg)$;
    $transport\_key \leftarrow findKey(nodes, msg.dst)$;
    $ct \leftarrow \Sigma_{transport\_key}.encrypt(msg)$;
    $em \leftarrow Message.Encrypted(this.addr, ct)$;
    $send(msg.dst, em)$;
```

5. Experiments

We concluded multiple experiments that consists of running blockchain with specific properties. We can divide these experiments to three parts:

- Without anonymization layer on localhost.
- With anonymization layer on localhost.
- With anonymization layer on separate virtual machines.

Each part consists of multiple runs with different settings that are described in Table 2.

First part aims to test the raw performance of the consensus layer. That can be seen as blue line in Figure 4 and Figure 3. The peak value of transaction per second is 28.4 when processing 10000 transaction in 1 block. The red line shows us results when the anonymization layer is turned on. This experiment shows us the difference in throughput when anonymization layer is turned on. The Figure 4 shows us that the anonymization layer has a small impact on throughput of the protocol.

The overall results are not that appealing and we found out that they are badly influenced by the implementation constrains of the used language (Python) and some other architectural flaws of the application itself (serialization and deserialization of Json, slow cryptographic library). We assume when using compiled language the result could be significantly better.

Run	Block Size	τ
1.	10	120
2.	100	120
3.	1000	120
4.	10000	120

Table 2. Blockchain properties that are applied to a specific run of a blockchain.

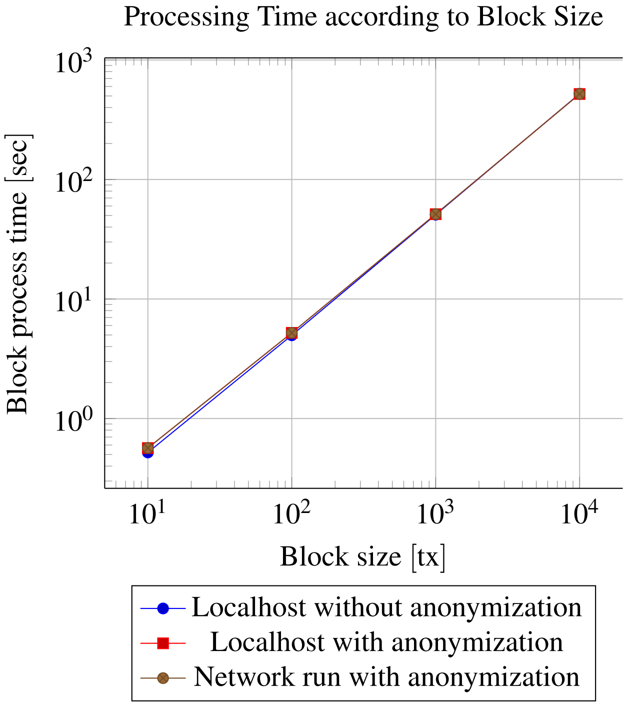

Figure 3. My first autogenerated plot.

6. Conclusions

This paper identifies the current problems of the current state-of-the-art Proof-of-Stake protocols such as throughput, anonymity in consensus layer. It implements proposed consensus protocol by the authors mentioned in Acknowledgment. To present the achieved results, the proposed protocol was implemented as proof-of-concept and tested in several scenarios.

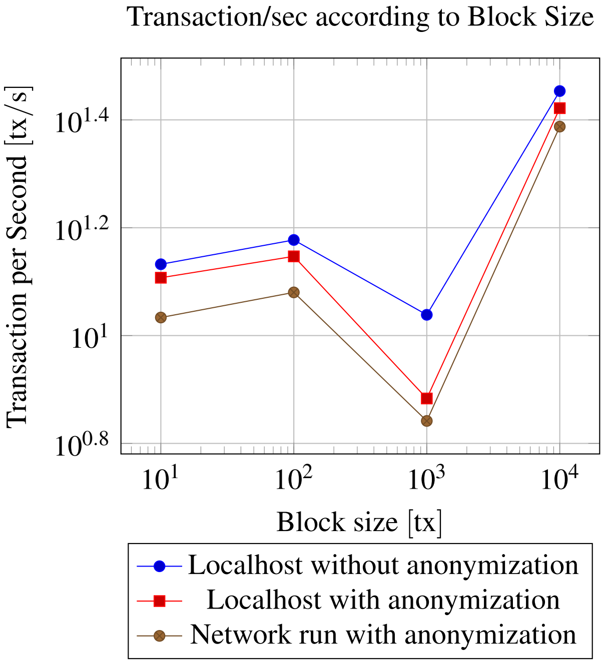

Figure 4. My first autogenerated plot.

Acknowledgements

We would like to thank my supervisor Ivan Homoliak for his supervision of this work. Further, we acknowledge that this work is a part of the ongoing research in the security@FIT group, and it will be fully published with all the contributors: Lukas Helebrant, Ivan Homoliak, Kamil Malinka, and Peter Hanacek. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

References

[1] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, School of Engineering, 2016.

[2] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[3] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[4] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[5] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.

[6] Daniël Reijsbergen, Pawel Szalachowski, Junming Ke, Zengpeng Li, and Jianying Zhou. Laksa: A probabilistic proof-of-stake protocol.

[7] Paul Syverson, Roger Dingledine, and Nick Mathewson. Tor: The secondgeneration onion router. In *Usenix Security*, pages 303–320, 2004.

[8] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[9] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):1–34, 2017.

[10] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[11] Ivan Homoliak, Sarad Venugopalan, Daniël Reijsbergen, Qingze Hum, Richard Schumi, and Pawel Szalachowski. The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses. *IEEE Communications Surveys & Tutorials*, 23(1):341–390, 2020.

[12] Lukas Hellebrandt, Ivan Homoliak, Kamil Malinka, and Petr Hanacek. Increasing trust in tor node list using blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 29–32. IEEE, 2019.