Proof-of-stake Protocols for Privacy-Aware Blockchains

Research Seminar in Cryptography

Author: Hiie Vill
Supervisor: Karim Baghery

1 Introduction

This report gives an overview on the paper ”Proof-of-stake Protocols for Privacy-Aware Blockchains”, published in 2019 by Chaya Ganesh, Claudio Orlandi and Daniel Tschudi. Despite the popularity of proof-of-work protocols in cryptocurrencies, there is a necessity for alternatives due to the wasteful nature of PoW protocols, causing huge energy loss due to the need to have a lot of computational power. Proof-of-stake protocols have risen as a contender for PoW protocols to tackle this problem. However, PoS protocols do not offer the same security guarantees as PoW protocols do – namely, such protocols leak information about the identity and wealth of the stakeholders, based on the frequency of winning the stakes lottery. The article this report is based on offers a framework for private PoS protocols, and an implementation of it for one PoS protocol named Ouroboros Praos [1].

2 Preliminaries and background information

This sections provides background information about the terms, definitions and topics necessary to understand private proof-of-stake protocols.

2.1 Blockchains

A blockchain is a list of cryptographically linked records. Each record is called a block and contains a cryptographical hash of the previous block (thus providing the link), a timestamp and data in the form of a Merkle tree. The data can contain any kind of information, for instance in the case of cryptocurrencies, the data consists of any coin transaction data. Since each block contains a hash of the previous one, modifying even a single bit in one of the previous blocks changes the hash value, making it impossible to change anything. This immutability property makes the blockchains very appealing for usage in cryptocurrencies [5].

As new transactions are made, they need to be stored in new blocks. There are economic incentives for people to add a new block - e.g. whoever gets to add a new block will be entitled to collect any transaction fees. In order to determine the next party who gets to add a new block (that party is also known as the block leader), a consensus must be reached. This paper only investigates one type of consensus protocols – lottery-based consensus protocols.

2.2 Lottery-based consensus protocols

Each lottery-based consensus protocol has a publicly verifiable probabilistic algorithm for determining the winner, who is responsible for adding the next block to the blockchain. The more resource the user owns, the greater the probability of winning the lottery. The integrity of this protocol is guaranteed by the following:

1. the resource used is scarce, meaning that nobody can manipulate their chances of winning

2. it is assumed that the majority of the resource is controlled by honest users.

The most prevalent type of protocols used in cryptocurrencies is proof-of-work (PoW), which uses a party's computational power as the resource, but due to its wasteful nature, proof-of-stake has gained popularity as a less wasteful alternative. For PoS, the resource is the user's own wealth. Other types of consensus protocols include proof-of-elapsed-time, proof-of-space, proof-of-retrievability etc [2].

2.3 Proof-of-work (PoW)

For Proof-of-work, each user performs a computationally difficult calculation to solve a puzzle – something that can only be solved by using brute force. Whoever solves the puzzle first has to publicly publish the solution and add a new block to the blockchain. All other users can verify that the block is valid (e.g. it contains the correct solution to the puzzle) and if so, use that block. Anonymity is guaranteed by the fact that all users are communicating to each other via an anonymous communication channel, such as Tor. This means it is very difficult to determine whether two blocks originate from the same user or different users.
Examples of proof-of-work protocols are Bitcoin, Litecoin and Zcash [2].

2.4 Proof-of-stake (PoS)

In the case of proof-of-stake, the blockchain has to contain information about each user's wealth. Instead of a centralized lottery, each user computes locally, based on a randomizer, whether they won the lottery or not. The winner may add a new block and publish proof of winning the lottery. Based on this information, other people can verify whether he indeed won the lottery. However, in order for this verification to work, all of the existing PoS protocols have the newly added block linked to the winner's account, revealing the identity and wealth of the winner to everybody.

The rising popularity of privacy-based cryptocurrencies raises the question of whether it is possible to make proof-of-stake protocols private, concealing the identity and wealth of the lottery winner, without losing in efficiency.

Some examples of PoS protocols are Algorand [4] and Ouroboros Praos [3].

Proof-of-stake protocols can in turn be divided into slot-based and committee-based protocols. Slot-based PoS protocols (e.g. Ouroboros Praos) use time divided into slots. A new block is generated for each slot, each unit of time. In the stakes lottery, stakeholders compete for those slots. In committee-based PoS protocols (e.g. Algorand), the next block is chosen by a committee and people in the lottery compete for the committee membership rights.

2.5 Ouroboros Praos

Ouroboros Praos is a proof-of-stake blockchain protocol, constructed by B. David, P. Gai, A. Kiayias and A. Russell in 2017 as an extension of an existing protocol called Ouroboros. It is the first PoS protocol that guarantees security against adaptive attackers. That means the protocol is secure even when an adversary corrupts any stakeholders at any point, assuming that the majority of the stakes still belong to honest parties [3][5].

2.6 VRF

Verifiable random function (VRF) is a pseudorandom function, so that together with a verification key, it is possible to verify whether any given output of the function is in the preimage of that function, relative to the verification key. Essentially this enables proving that a VRF output has been computed correctly.

3 Functionalities

This section describes a set of functionalities that are going to be used later on. Functionalities are called via messages and they return messages as well. sid stands for a stake id in each of the functionality calls.

1. Stake Distribution: $F_{Init}^{Com,SIG}$. It is assumed that all stakeholders can access a list of stakeholder accounts, each containing information about the committed stake and the signature verification key. This functionality has two parameters: a commitment scheme Com and a signature scheme $SIG = (KeyGen, Sign, Ver)$. Given a list of stakeholder ID-s pid, and stakes for each ID α_{pid}, the stake commitment is computed as $Com(\alpha_{pid}; r_{pid})$ and the verification key as $vk_{pid} = KeyGen(sk_{pid})$, where sk_{pid} is chosen randomly. This functionality enables each user to query a list of static stake data ($(Com(\alpha_{pid}, vk_{pid})$ for each pid) and each user pid to query his own static data (α_{pid}, r_{pid} and sk_{pid}). This functionality supports two messages:
   (a) $(GETPRIVATEDATA, sid)$. This returns static stake information for a user pid. This includes a user's stake value α_{pid}, his signing key sk_{pid} and a randomness r_{pid} used for the stake commitment.
   (b) $(GETLIST, sid)$. This returns the list of all registered commitment and verification key pairs.

2. Common reference string: F^{D}_{crs}. The protocols described in this paper make use of zero-knowledge in order to prove a user's victory. Given a distribution D, the function F^{D}_{crs} returns a CRS usable in these ZK-proofs.

3. Verifiable pseudorandom function: F^{Com}_{VRF}. Verifiable random functions are used to introduce randomness into the lottery. This functionality is split three ways:

   (a) $(KEYGEN, sid)$. Firstly, there exists a key generation functionality to generate a key vid for any party pid.

   (b) $(EVAL, sid, vid, m)$. Secondly, this functionality enables to evaluate the VRF: for a user pid with a message m, the tuple $(y, Com(y; r), r)$ is returned, where $y, r \leftarrow_{\$} \{0,1\}^{l_{VRF}}$. The returned value is stored in a database and linked to that particular message m and the user pid. It should be noted that the message m usually represents the new block that is going to be added.

   (c) $(VERIFY, sid, m, c)$. Thirdly, this functionality enables other users to verify whether a given VRF output was computed correctly. For a message m and a commitment c, the function returns if a matching entry exists in the database, by checking each vid and m combination. The exact user who generated that particular VRF is kept secret.

4. Anonymous broadcast: F_{ABC}. This functionality enables users to send messages to other parties anonymously over an anonymous broadcast channel. This is required to conceal their identities.

4 Feasibility of Private Proof-of-Stake

When discussing the feasibility of a Private proof-of-stake (PPoS), it is not sufficient to only conceal a stakeholder's identity, since the PoS lottery reveals information about the wealth of the winner, based on how often that user wins the lottery. Hence both identity and wealth should be kept secret.
This can be achieved by a user proving that he knows the corresponding secret key to a public key in the list of all accounts. The stake corresponding to that account wins the lottery.
The lottery itself may take several entry parametres called ε, as well as the value of the stake α_{pid}. ε may vary from protocol to protocol, e.g.for slot-based PoS protocols ε is made up of slots.
An important thing to note is that this strategy only applies to protocols which use locally verifiable lottery functions - where each party can compute themselves whether they won or not.

4.1 Private Lottery Functionality $F^{LE,\varepsilon}_{Lottery}$

In addition to the four functionalities described in section 3), a fifth functionality will be described to act as the lottery process, called by all users wishing to participate in the block leader election. The private lottery functionality $F^{LE,\varepsilon}_{Lottery}$ is defined as follows:

1. Let ε be a set of entries to the lottery.

2. Let LE be a predicate function with inputs α_{pid} and r_{pid}, returning true if pid won the lottery and false otherwise.

3. Let P be a list of registered parties participating in the lottery and let T be a table storing LE results for each input pair (pid, e). Additionally, let there be a message buffer for each party.

4. The lottery functionality enables sending 5 types of messages, supporting the abilities to participate in the lottery, publish messages, the ability for the adversary to publish messages on the winner's behalf; read messages and retrieve one's stake value.

   (a) $(LOTTERY, sid, e)$. Given an entry input e by a stakehoder pid, check if $T(pid, e)$ is empty. If it is, generate $r_{pid} \leftarrow_{\$} \{0,1\}^l$ and $LE(\alpha_{pid}; r_{pid})$and save the result in $T(pid, e)$. Return $T(pid, e)$.

   (b) $(SEND, sid, e, m)$ Given an entry input e and a message m from pid, check $T(pid, e)$. If $T(pid, e) = 1$, the party pid won the lottery. In that case, output (e, m) to the message buffers of each party. If an adversary poses as pid, he additionally receives $(SEND, sid, e, m)$.

   (c) $(SEND, sid, e, m, P')$. This message duplicates the previous one with the exception that it is sent by an adversary posing as P'. In case $T(pid, e) = 1$, this also outputs (e, m) pairs to each message buffer and sends $(SEND, sid, e, m, P')$ back to the adversary.

   (d) $(FETCH - NEW, sid)$, received from a party P (or an adversary posing as P), returns P's message buffer contents and empties his message buffer.

   (e) $(GET{-}STAKE, sid)$, received from a party P, returns his stake value: $(GET{-}STAKE, sid, \alpha_{pid})$.

4.2 Private Lottery Protocol

The private lottery protocol makes use of the lottery functionality $F^{LE,\varepsilon}_{Lottery}$, as well as all of the functionalities defined in section 3. Each user turns to $F^{LE,\varepsilon}_{Lottery}$ to participate in the lottery and locally compute whether they won. The winning party is then allowed to publish a message m.
In order to guarantee privacy, the lottery winner uses zero-knowledge proofs to prove his victory and ownership of the winning stake. The winner pid will need to prove the conjunction of the following statements:

1. $LE(\alpha_{pid}; r_{pid}) = 1$ – this particular stake won the lottery.

2. $C_{\alpha_{pid}} = Com(\alpha_{pid})$ – the winning stake's commitment equals pid's commitment, meaning he owns the winning stake.

3. $Ver_{vk_{pid}}((e,m),\sigma) = 1$ – the pair (e,m) has been signed by pid, verifiable by his verification key vk_{pid}.

4. $vk_{pid} = KeyGen(sk_{pid})$ – the winner knows the secret key to the corresponding verification key.

5. $(C_{\alpha_{pid}}, vk_{pid}) \in L$ – the stake and verification key pair is valid and belongs to the maintained list containing the commitment and public key pairs of all stakeholders, published at the start of the protocol.

The lottery Protocol also assumes the existence of the algorithms $Setup$, $Prove$ and $Verify$, forming a NIZK system.
The $LotteryProtocol^{\varepsilon,LE}$ works as follows:

1. The protocol begins with each party P sending $(GETLIST, sid)$ to retrieve the list of all commitment and public key pairs of all stakeholders.

2. Each party then sends $(Setup, sid)$ to F_{crs}^{D} to generate himself a common reference string crs to be used later in the ZK-proofs.

3. All stakeholders additionally retrieve their own static stakes data: α_{pid}, r_{pid} and sk_{pid} by sending $(GETPRIVATEDATA, sid)$ to $F_{Init}^{Com,SIG}$; and a VRF evaluation key vid by sending $(KEYGEN, sid)$ to F_{VRF}^{Com}. They also maintain a table V.

4. Each stakeholder sends $(LOTTERY, sid, e)$ to $F_{Lottery}^{LE,\varepsilon}$. If there is no value in V corresponding to e, send $(EVAL, sid, vid, e)$ to F_{VRF}^{Com} to retrieve y, c and r, acting as the randomness in this protocol. Then they each compute $b = LE(\alpha_{pid}, y)$ to determine whether they won, and store (b, y, c, r) in $V(e)$. Then $(LOTTERY, sid, e, b)$ is called. This step represents each stakeholder taking part in the lottery.

5. Each stakeholder can then send the message $(SEND, sid, e, m)$. The winner's message m is added to everybody's message buffers, all other messages are ignored.

6. The winner creates a ZK-proof as follows:

   (a) Since he won, meaning $b = 1$, $V(e)$ is assigned the value $(1, y, c, r)$.

   (b) Using sk_{pid} (obtained in step 3), sign the pair (e, m). This is necessary to prevent any other messages apart from m being published with this proof.

   (c) He then calls the NIZK algorithm $Prove$ to generate a NIZK proof π_{zk} and returns the e, m, π_{zk} to the anonymous broadcast functionality F_{ABC} via $(SEND, sid, (e, m, c, \pi_{zk})$, thereby publishing his entry, message, commitment and ZK-proof of his victory.

7. Each stakeholder can run $(FETCH - NEW, sid)$ to read their message buffer and see the winner's message (e, m, c, π_{zk}). They can then verify that:

   (a) e belongs to the entry set

   (b) the VRF output was computed correctly, by calling $(VERIFY, sid, e, c)$

(c) the ZK-proof is correct (by calling the NIZK algorithm $Verify$)

If all three verifications pass, other users can be certain of the winner, and can use his message accordingly.

As proven in the article, this protocol guarantees security against any probabilistic polynomial-time adversaries, assuming that the majority of the resource is controlled by honest parties.

5 Making Ouroboros Praos private

This section gives an overview of the Ouroboros Praos protocol and a constrcution for improving this protocol to make it private, using the private PoS framework described in the previous section.

5.1 Ouroboros Praos Leader Election

For each party pid, the probability of winning the lottery for a slot sl is $p = \phi_f(x) := 1 - (1-f)^{\frac{\alpha_{pid}}{Stake}}$, where f is the difficulty parameter, α_{pid} is the stakeholder's wealth and $Stakes$ is the total number of stakes.
Each stakeholder pid evaluates the VRF, using his private key k: $(y, \pi) = VRF(k, sl)$. For a VRF output length l, he also computes the threshold $T = 2^l \cdot p$. In order to check if he won the lottery or not, pid then compares y to the threshold. If $y < T$, then he has won, otherwise he has lost. This is calculated independently and locally by each stakeholder.

5.2 Ouroboros Praos VRF

For the VRF, Ouroboros Praos makes use of a 2-hash VRF based on the hardness of the Computational Diffie-Hellman (CDH) problem. For a group G of order q, a secret key $k \in Z_q$, a public key $v = g^k$ and two hash functions H_1 and H_2, $VRF(m)$ returns a pair y, π, which can be verified by others by using the public key.

5.3 Anonymous Verifiable Random Function

As a new term, this article defines an anonymous verifiable random function - a VRF with the property that the public key remains secret during the verification. This can be achieved by associating many public keys with one secret key, meaning that two different VRF evaluations $(y_1, \pi_1) = VRF(k, m_1)$ and $(y_2, \pi_2) = VRF(k, m_2)$ on two different messages, but using the same secret key, cannot be linked to a public key - it is impossible to distinguish whether they were generated by the same secret key or not. However, the verification still holds as there exists such a public key that passes the VRF evaluation verification.

5.4 Anonymous VRF Construction

An anonymous VRF contains of 4 algorithms Gen, $Update$, $VRFProve$ and $VRFVerify$:

1. Gen generates a key pair. Namely, it generates a group of a prime order q, generated by the generator g, chooses $k \leftarrow_\$ Z_q$ and returns $(pk, k) = ((g, g^k), k)$.
2. $Update(pk)$ is responsible for updating the key pair by choosing $r \leftarrow_\$ Z_q$ and, for a $pk = (g, v)$, returning $pk' = (g^r, v^r)$.
3. $VRFProve(pk', x)$ generates a proof for x, using the key pair $pk' = (g, v)$. The algorithm outputs (pk', y, π), where π is the proof.
4. $VRVerify_{pk'}(x, y, \pi)$ performs the proof verification and outputs 1 if the verification passes, 0 otherwise.

5.5 Private Ouroboros Praos

On a high level, the idea behind making the Ouroboros Praos protocol private is to prove $y < T$ in zero-knowledge, revealing neither y nor T. Additionally, it is necessary to prove that T was computed correctly. This can be done by splitting a party's stake between multiple virtual stakes. The probability of that party's stake winning is equal to the probability that one of those virtual parties wins, so it is sufficient to prove that one of the virtual parties won the lottery.

6 Conclusion

This report gave an overview on an article of the same name, which introduced a framework for generating private proof-of-stake protocols to conceal stakeholders' identities and wealth, and extended an existing Ouroboros Praos protocol by implementing this framework.

7 References

[1] Ganesh, C., Orlandi, C. and Tschudi, D., Proof-of-Stake Protocols for Privacy-Aware Blockchains. Cryptology ePrint Archive, Report 2018/1105, 2018. https://eprint. iacr. org/2018/1105.
[2] Wahab, A. and Mehmood, W., 2018. Survey of Consensus Protocols. arXiv preprint arXiv:1810.03357.
[3] David, B.M., Gazi, P., Kiayias, A. and Russell, A., 2017. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. IACR Cryptology ePrint Archive, 2017, p.573.
[4] Gilad, Y., Hemo, R., Micali, S., Vlachos, G. and Zeldovich, N., 2017, October. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles (pp. 51-68). ACM.

[5] Crosby, M., Pattanayak, P., Verma, S. and Kalyanaraman, V., 2016. Blockchain technology: Beyond bitcoin. Applied Innovation, 2(6-10), p.71.