OUROBOROS AND OUROBOROS PRAOS: PROBVABLE, SECURE, PROOF OF STAKE BLOCKCHAIN PROTOCOLS

CIS5371 – Paper Summary

Nikhilesh Reddy Tummala

(8350-1593)

CISE University of Florida, 2020

***Abstract* – This paper gives a comprehensive summary of the blockchain protocol OUROBOROS which is based on Proof-of-Stake mechanism that guarantees high security. Firstly, the paper gives a brief overview of bitcoin and its Proof-of-Work protocol – which is expensive – and that motivates us to look into Proof-of-Stake which later leads to the discussion of the workings of Ouroboros with a model that describes the operations of this protocol and prove security guarantees. Later, we look into Ouroboros Praos by differentiating it from Ouroboros and how it enables us to improve Ouroboros.**

I. INTRODUCTION

The energy required for the execution of blockchain protocols [1] are based on Proof of Work [2]. In the beginning, Huge number of hashing operations were needed to create a single block on the blockchain and to achieve such results required tremendous amount of computational power that many couldn't afford. Due to this state of affairs, new blockchain protocols came into existence that would prevent the need for Proof of Work by replacing it with a much efficient energy saving mechanism that could provide equivalent results, called Proof of Stake (which is explained in the bitcoin forum[3]). In bitcoin's Proof of Work mechanism, a miner is selected based on the computational power of each miner and then the selected miner issues the new block on blockchain, whereas in Proof of Stake mechanism, a miner is selected based on the amount of stake he invests into the currency which makes the blockchain depend on stakeholders to maintain itself and provide financial punishments and rewards to the stakeholders based on their behavior in the system.

This paper presents a brief summary and a top-level overview of Ouroboros [4] and Ouroboros Praos protocol models. Here are some of the important aspects of the protocol:

We show a model that formalizes the problem of realizing a Proof-of-Stake based blockchain protocol, focusing on persistence and liveness. Persistence specifies that when a party in the system declares a particular transaction as 'Stable', then the remaining parties in the system will report the same, if they are honest. Stability is established by parametrizing a security parameter 'k' which has an impact on the property. Liveness guarantees that when an honest transaction is made available to all the parties for a particularly sufficient amount of time, then it will be stable. The conjunction of these two properties, provide robust transaction ledger in ways, that honestly generated transactions are adopted and become immutable. We present a novel blockchain protocol based on Proof-of-Stake which assumes that parties can freely create accounts, receive payments, make payments and shifting of stakes over time by employing a secure multi-party computation [5] of a coin-flipping protocol to produce the randomness for the leader election process, making our approach unique from the previous solutions which either define such values deterministically based on the current state of the blockchain or use collective coin flipping as a way to introduce entropy.

A set of formal assumptions are provided which states that: No adversary can break persistence and liveness. The network is synchronous in a manner that an upper bound can be determined during which any honest stakeholder is able to communicate with any other stakeholder. A number of stakeholders pulled from the honest majority is available, to participate in each epoch. The stakeholders do not remain offline for long periods of time. The adaptivity of corruptions is not immediate, which is measured in rounds linear to the security parameter. At the core of our security assumptions is a probabilistic argument regarding a combinatorial notion of "forkable strings" which is formulated, proved and also investigated experimentally in the paper. Coming to the incentive structure of the protocol, the paper presented a novel reward mechanism for incentivizing the parties in the system which proved to be an (approximate) Nash equilibrium, thereby causing to alleviate the design model by attacks like block withholding [14] and selfish mining [15]. The paper introduced a stake delegation mechanism that can be seamlessly added to the blockchain protocol. Delegation is particularly useful in this context to enable the protocol to scale even in a setting where the set of stakeholders is highly fragmented.

Later, we look into another proof-of-stake blockchain protocol which is an improved version of Ouroboros called Ouroboros Praos [6], a provably secure proof-of-strake protocol which is the first to be secure against adaptive adversaries and scalable in practical sense. As Ouroboros Praos is based on Ouroboros, its analysis is developed on some of the core combinatorial assumptions that were built to analyze that scheme. However, the construction of this protocol involves novel elements that requires significant recasting and generalization of the previous combinatorial analysis. Ouroboros Praos is better when compared to Ouroboros due to its Semi-Synchronization and Adaptive Security. In this paper, we will look into these characteristics and the tools required to achieve such properties.

II. BACKGROUND

Bitcoin is a cryptocurrency which was first proposed in 2008 by using the concept of decentralized ledger mechanism[7], maintained in a permission-less setting, which relies on a novel security assumption that an adversary can take over the whole bitcoin if his computing power can dominate those of the honest participants. After bitcoin, many other cryptocurrencies came to existence. Some of those aim to provide better security guarantees and some of these cryptocurrencies aim to work with a different consensus mechanism (which Ouroboros talks about). One of the major problems that bitcoin solves, would be using the concept of "Ledger" where distributed collection of parties wish to agree on a dynamic common sequence of transaction. All the transactions present in the Ledger is immutable and the new transactions will be added without any undue delay. Users have a choice to participate in the system or remain offline. It is wise to expect some adversaries who are among the users, trying to disrupt these protocols. This bitcoin ledger is stored in a data structure called Blockchain [8] that starts with a genesis block which gets created upon receiving consensus from every user and we keep on adding new blocks on top of this genesis block, containing information about the transactions and the hash of the previous block.

One of the Major challenges bitcoin encounters (or any other cryptocurrencies in general), is achieving consensus on extending the blockchain. Interestingly, this is outside the classical distributed computing models as it does not make sense to discuss about majority of players. This is due to sybil attacks [9]. To overcome this issue, Bitcoin deploys a solution of combining Proof-of-Work with the concept of Longest Chain Rule. This rule states that if a party is looking at a potential candidate for the state of the ledger, then he will choose the chain which has the longest length. In simpler terms the bitcoin protocol follows the next mentioned steps in a repeated fashion. First, it collects any pending or broadcasted transactions, adopt longest valid blockchain as current state and then attempt to mine a new block on the selected blockchain with the collected transactions. If successful, it will broadcast and repeat this process forever, achieving eventual consensus.

If we have an adversary that tries to delete any transaction from the ledger, he would have to fork the longest blockchain and that would be highly unlikely if he controls only a minority of the cryptocurrency. That is where the Proof-of-Work protocol is utilized to select a miner so that he can add another block on top of the valid blockchain. In this way, bitcoin solves the challenge of receiving consensus with a fluid population of participants. But as discussed before, Proof-of-Work protocol causes an issue with computational power by consuming loads of electrical energy. Replacing Proof-of-Work protocol with an alternative mechanism like Proof-of-Stake protocol would be an improvement.

Challenge for Proof-of-Stake is to elect a participant which requires unbiased randomness proportional to stake among the parties involved, but if the adversary can bias the randomness then he can bias the election process and try to hijack the whole blockchain. An obvious approach to handle this issue is to just hash the current blockchain which identifies a particular coin that later leads to identifying the owner, who is allowed to put the next block on the blockchain by using a simple fact that blockchain already contains effectively random data (i.e., block hashes). The solution seems plausible but it would be an incorrect approach because of rejection sampling where the adversary can locally test the outcome of process and if the resultant leader is not himself after creating a block, then he discards the block and tries again, until he is elected as the leader for the next block. Once that happens, the adversary can do this repeatedly and hijack the chain. This is also called 'Grinding Attack'.

III. RELATED WORK

Below are some of the constructions that deals with the above discussed problem and deliver rigorous guarantees in terms of security for Proof-of-Stake protocol. Ouroboros is one such solution that will be elucidated further in this paper. But to give a simple overview of Ouroboros, it implements a secure multi-party computation to generate clean randomness, unbiased by the adversary (As long as some assumptions satisfy, which will be explained later). There are different classes of protocols like, Snow White[10] and Ouroboros Praos. They choose slightly different approach but they implement hashing in a careful manner.

In addition, they do deliver a rigorous guarantee that contains grinding attack and the randomness can be used for staking without jeopardizing the security properties of the blockchain. Algorand [11] is another protocol (predates all the proposals listed above) that uses a procedure, contrary to the above constructions (that aim for foreign eventual consensus like the longest chain rule). Algorand tries to get complete consensus from every block. The reason Algorand is mentioned is because it is still randomness for its running and it has to address similar issues by implementing procedures similar to the above discussed procedures where it implements some hashing and does analysis, proving that the effects of grinding attack can be contained.

There are several other Proof-of-Stake solutions in the wild, namely: NXT cryptocurrency; Peercoin; DPoS which underlines Bitshares, Steam and EOS; Casper (Ethereum) etc.,

IV. OUROBOROS PROTOCOL

A. Insights

In this section, we discuss about the workings of Ouroboros model, where we analyse and prove its security. On a top level view, Ouroboros is a protocol that assumes synchronous time and communication. So, we look at it in a synchronous network model that is alleviated in Ouroboros Praos, it provably achieves persistence and liveless properties under some assumptions, such as: The adversary controls a minority of the stake throughout the execution of the protocol. We assume that corruption from the adversary is not immediate, i.e., he might corrupt some parties but some delay is expected. This is also an assumption that can be alleviated in Ouroboros Praos. Lastly, the stake shifts are happening at a bounded rate, which makes sense while transferring stake from one account to another in a bounded rate rather than transferring all at once.

B. Communication Model

Detailing into this communication model, as mentioned before, it is a synchronous model where we assume that participants have synchronized watches and we split the time into slots (For e.g., In some implementation of Ouroboros, 1 slot is 20 seconds). We assume that all messages that are being sent by honest players are received by all other honest players within the same slot. This is the synchronicity assumption of the model. The property of the protocol is that, all the parties communicate only by broadcast which in practice, is implemented by peer-to-peer gossiping. We give the power to send arbitrary messages to arbitrary subsets of players and arriving at arbitrary times by those parties which are effected by the adversary.

C. Protocol Overview

This protocol works by organizing into epochs where each epoch is a sequence of R slots. So, it is a particular time interval. Multiple blocks are generated on the blockchain in every epoch, as shown in the image below, where all the blue blocks are generated in one epoch, the yellow blocks are created in the next epoch and the red ones in the next one. Within these epochs, we need Stake Distribution which will be used to sample the winners of the election process and the way we obtain this stake distribution for a particular epoch (Let's say, for all the yellow epoch) is by taking a snapshot of how the stakes are distributed in a particular moment in the previous network.

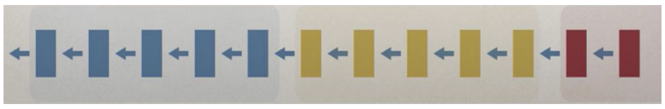


Fig: 1

Except for stake distribution, we also need randomness and it is produced in the previous epoch by a multi-party computation. During the Blue epoch, there is an NPC running, which leads to an unbiased randomness that is then used in the yellow epoch to sample the winners of the election. These are called Slot Leaders in Ouroboros. So, in every slot we elect a unique player that is allowed to create a block in that slot and the sequence of these slot leaders is called a Leader Schedule which is then sampled using the stake distribution and randomness. This circle of steps is how Ouroboros looks from a top level perspective.

In other words, a snapshot of the current set of stakeholders are taken in every epoch in which, a secure multiparty computation is implemented by employing the blockchain as the broadcast channel. A set of randomly selected stakeholders form a committee in each epoch, which is accountable for executing the coin-flipping protocol. The outcome of the protocol determines the set of elected stakeholders that will execute the protocol in the subsequent epoch, as well as the outcomes of all leader elections for the epoch.

If we want to analyse the security of Ouroboros, then we do it in 2 steps where the first step is to look at one single epoch and assume that the stake is static. This is because of the fixed stake distribution being used for sampling the slot leaders and the randomness is clean and given at the beginning. This is called static setting and later we will see how to bootstrap to analyse the whole protocol and the sequence of these epochs. So, let's begin with the static case.

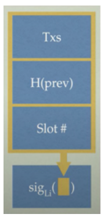
1. Static Case

We assume that there is a genesis block that contains the public keys of the players, together with their proportional stakes which are given at the beginning and shared by all the players. We also assume that there is a sufficiently long, clean random string for the leader election. Now, we look at one particular epoch of length R and try to show that during this epoch, persistence and liveless is achieved. In static case leader election, leaders are determined by ρ: Fix a function L so that $L(\rho) = (L_1, \dots, L_R)$ where L_i is a participant, independently chosen, according to relative stake.

$$\Pr[L_i = U] = \frac{\text{Stake of U}}{\text{Total Stake}}$$

Since, L is public and random is public, the leader schedule is public as well. Now, let's look at the verification of the validity of the blockchain in the static case. Firstly, it has to begin with the genesis block which contains a sequence of blocks that are associated with increasing slot numbers and each head contains the address of the precious one along with no conflicting transactions. Most importantly, there doesn't have to be a block for each of the slots because there might be a scenario where, to put the unique slot leader that was chosen for that particular slot, did not produce a block or maybe he was malicious or he was offline for a while or any other reason. We consider a chain valid even if there are slots without blocks but each of these blocks have to be signed by the leader of the slot to which this block claims to belong. That is why the structure of Ouroboros is as follows: it contains the transactions, the head of the previous block, slot number and signature from the secret key that corresponds to the public key that is present in the stake distribution.

Fig: 2 :-



The protocol in the static case is as follows: The party collects all the transactions that he sees on the network, collects all the broadcasted blockchains and adopts the longest valid chain as its current state. Later, if the party figures out that it is the leader for the current happening slot, then it creates an additional block on top of the adopted chain, signs it and broadcasts it to everyone. We can see a lot of similarity between this protocol and the working of bitcoin even though we got rid of the Proof-of-Work aspect and different election process, but the longest valid chain rule remains the same. However, the analysis is different in several crucial aspects from bitcoin. For e.g., the adversary in contrasts to the bitcoin adversary has various additional powers at his disposal. Here, the adversarial parties determined at the outset (before selection of ρ & leaders); stake = (1 - ε) / 2. Unlike bitcoin adversary, he knows the entire sequence of leaders ahead of time and can generate multiple blocks per slot without any cost. But, this raises the question: How much does this increase the power of the adversary? Can we prove that it doesn't allow him to disrupt the security of the blockchain? To answer the question, let's look at the analysis.

1.1 Analysis

The paper introduces a framework to describe the combinatorial behaviour that is happening when we are extending the blockchain in the above discussed manner where some slots belonging to honest parties and some being controlled by the adversary. To understand this, we use the notion of characteristic string which is essentially a binary string of length R, that displays how the slots are distributed among honest parties and adversaries in the form of 0's and 1's, where 0's slot leader for the particular slot at the position is an honest party and 1's indicate that they belong to adversarial party. We notice that this string, in the execution of the protocol, is just binomially distributed with parameter (1 - ε) / 2. The adversary controls a minority of the stake and also controls minority of the slots, where less than one half of the bits in the bit string will be shown 1 We hope that this is sufficient to observe the dynamics of the protocol as it is executing, leads to achieving persistence and liveness.

1.2 Proof

The goal here, is to show persistence and liveness but it is well known that in blockchain based protocols, we show 3 main properties:

Common Prefix (k): Any 2 chains (C_1 , C_2) possessed by any 2 honest parties potentially at different times, have the property that if we cut the final k blocks from one of the chains, it will become a prefix to the other one.

Chain Quality (k): It says that any chain possessed by an honest party, has the property that at least one of the blocks within the last k blocks was honestly generated by an honest party.

Chain Growth (k,τ): It means that there is a lower bound to the growth of the chain at every sequence of k slots.

Before moving further, let's understand the dynamics of the protocol.

1.3 Dynamics

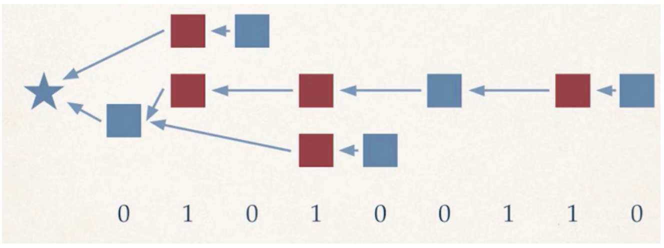


Fig: 3

Let's consider an epoch with 9 slots having an agreed-upon genesis block along with a characteristic string. Now, the first slot leader is honest and so he follows the protocol where he simply creates the block that extends the longest valid chain, which is the genesis block. At this point, the adversary can create a block that does not extend along the genesis blockchain but creates an alternate chain. If there are multiple chains of same length then we let the adversary decide which of the chains will be extended by this honest party. For e.g., if the adversary makes the honest party extend the upper chain then it will be the adversary's turn and he would create another chain of same length and so on. As we can see, the honest parties get confused if there are numerous chains of same highest length and this violates the persistence property. It is imperative to note that it is highly unlikely for an adversary to create multiple chains of same lengths in real-time, to confuse the honest parties, because of the computational power required to achieve this task would be unimaginably high and quite expensive.

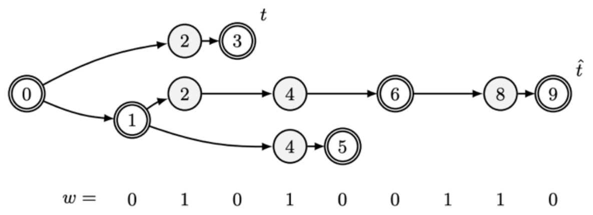


Fig: 4

To show this formula, there is a forks calculus introduced in the paper where this notion is formalized. We talk about fork which is the name for such a tree as shown above. It is a graph that indicates these chains. Nodes corresponds to blocks and edges corresponds to the predecessor relation. We have a genesis block along with honest blocks (represented by double circles) and adversarial blocks (represented by single circle). All nodes are labelled with their slot number and each node has a unique edge to its previous node with a smaller label.

According to the model, all players here are honest, broadcast all honest blocks and all these players speak at most once in any slot. From these two observations, we understand about the graph that any honest slot if associated with exactly one honest node and the depth of any honest block exceeds the depth of all previous honest blocks. Because, if an honest party broadcasted a block then later on, this honest party will not be willing to extend a chain that will be shorter than the one that this honest party has already broadcasted (due to the synchronization assumption).

Formally a Fork 'F' for a characteristic string $w = w_1 \ldots w_n \in \{0,1\}^n$ is a labelled rooted tree. Each node is labelled with an element $\{0,1,\ldots,n\}$ such that: Root is labelled with 0; edges are directed away from the root; labels increase along the paths; honest slots (i so that $w_i = 0$) label a unique vertex. And the depths of the honest vertices increase.

Any tree that satisfies these four conditions is called a fork for that particular string. So, for a string, there is a large set of forks that can potentially occur as the dynamics of the protocol evolve. The goal for proving common prefix property for the protocol is to show that, if we start by using this characteristic string from this binomial distribution that we discussed, then the probability that common prefix will be violated is negligible. But the way we try to accomplish this is by showing that if common prefix is violated for the particular string, it would mean that a very obscure fork has occurred. So, the graph would have some unexpected properties, where there would be two long paths that were diverged from each other a long time ago. This would be necessary for violation of common prefix. And so, we can try to show in a combinatorial way that, over the probability of sampling this characteristic string with overwhelming probability, we end up with a string that doesn't allow a fork which would have this undesirable property.

We call a string 'w' forkable, if there are two paths of maximum length and they diverge at the beginning, having no overlap except for the genesis block. An example for this is shown below where we have 9 slots.

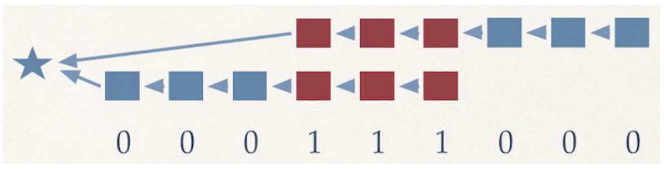


Fig: 5

From the image, it is clear that adversary controls one-third of the slots but what it can do is, it will create an alternative path of length 3 after the honest parties have done their job. Later, the final honest parties extend the adversarial chain and then create an additional 3 blocks in the lower chain. This creates 2 paths that are both of length 6, having the genesis block as the only intersection block, which shows that this particular string of length 9 is forkable. One can observe that no string of density less than one-third is forkable. This is like a corner case because the adversary needs at least one-third of 1's to be able to create such a fork. If we were aiming for one-third, then security would be done quite easily.

Ouroboros is aiming for security up to adversaries that contain any arbitrarily large minority of the stake. Of course, it is noticeable that if the adversary controls more than one-half of the slots, then the string is clearly forkable. Now comes the question: what is the probability that a particular string that is sampled from this distribution will be forkable? This question is answered in the Ouroboros paper but a stronger bound is achieved by another author in sense of the following theorem:

- Theorem[12] : Draw $w_1 \ldots w_n$ from the binomial distribution with parameter $(1 - \varepsilon) / 2$. Then Pr[w forkable] $\leq e^{-\Omega(n)}$.

- Theorem: If w permits a k-CP violation, there must be a forkable substring of length k.

- Theorem: (via union bound):
  Pr[k-CP violation in string of length n] $\leq ne^{-\Omega(k)}$.

A brief description of the proof for this theorem is shown below. It is via martingale argument. The relevant feature that needs to be looked at, is the longest and the second longest chain so far.

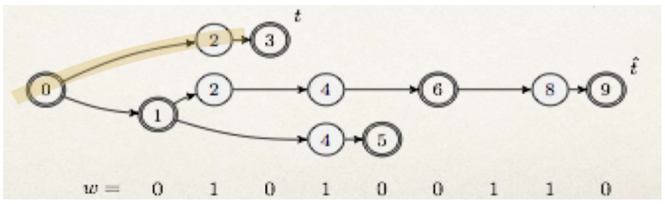


Fig: 6

From the above figure, if we look at the particular path denoted by 't', then:

- gap(t): Length difference with deepest honest node.
- reserve(t): Number of adversarial slots after end of t.
- reach(t) = reserve(t) – gap(t).

Based on the definitions, we can say that gap(t) = 4, reserve(t) = 3 and reach(t) = -1. And when we look at the fork F, then:

- reach(F) = max reach(t).
- margin(F) = second best disjoint reach(t).

If we move one level higher, then for a string w:

- $\rho(w) = \max_F \text{reach}(F)$
- $\mu(w) = \max_F \text{margin}(F)$

String w is forkable iff $\mu(w) \geq 0$. This is because, there exists a fork for a particular string in which the adversary if wishes, is capable to extend the second longest disjoint chain to the length of the longest chain. That indicates that he can produce two histories that are of the same length, overlapped at the beginning, there by puzzling all the honest parties and violating persistence property. We try to achieve probability of $\mu(w) \geq 0$ is negligible, by this distribution of the characteristic string. The idea – that is followed in the paper – is to analyse the pair $(\rho(w), \mu(w))$ as random variables for this distribution of string w.

With some work, one can show that these two quantities inductively follow a particular recursive expansion that is shown below.

$(\lambda(\eta), \mu(\eta)) = (0,0)$ and for all strings $w \in \{0,1\}^*$,

$(\lambda(w1), \mu(w1)) = (\lambda(w1) + 1, \mu(w1) + 1)$ and

$$(\lambda(w0), \mu(w0)) = \begin{cases} (\lambda(w0) - 1, 0) & \text{if } \lambda(w) > \mu(w) = 0, \\ (0, \mu(w) - 1) & \text{if } \lambda(w) = 0, \\ (\lambda(w) - 1, \mu(w) - 1) & \text{otherwise} \end{cases}$$

A related martingale can be defined to these random variables λ & μ. This concludes the proof of common prefix. This establishes that the static part of the protocol which only describes one epoch, does achieve persistence and liveness.

2. Dynamic Protocol

Now we need to bootstrap and get to the dynamic protocol where one epoch follows another. As mentioned before, we need an updated randomness for the new epoch to be used for sampling the slot leader schedule. Ouroboros uses a secure multi-party computational protocol to generate this randomness by utilizing the blockchain itself during this epoch as the communication medium. The generated randomness is guaranteed to be clean as along as the majority of the stake holders or the slot leaders for that particular epoch are honest. The tool that we use for this protocol is Publicly Verifiable Secret Sharing [13] where there are dealer and family of players.

The dealer can choose a value and produce shares for each of the players and then players can check whether the broadcast it shared are valid, in the sense that together they reflect the consistent value. If it is valid, then majority of the players can reconstruct this value but minority of the players can learn nothing of the value. This is very useful for our goal because we can simply make the parties use PVSS to share a random string that they generate. For efficiency reasons, the parties commit to this string and after all these commitments are put into the blockchain, the players open the commitment and ExOR the values to get the new randomness.

But if someone fails to open their commitment, then the PVSS shares can be used to reconstruct their value. In this way, we achieve clean randomness for next epoch. When it comes to practicality of Ouroboros, it was implemented by IOHK. There are several ways to look at the performance, but the one presented in the paper is where one can compare how much time he needs in minutes to achieve 99.9% assurance against an adversary attempting a double spending attack, as a function of hashing power or stake.

Adversary	BTC	OB Covert	OB General
0.10	50	3	5
0.15	80	5	8
0.20	110	7	12
0.25	150	11	18
0.30	240	18	31
0.35	410	34	60
0.40	890	78	148
0.45	3400	317	663

Fig: 7

The above table shows the comparison of transaction confirmation time between bitcoin and Ouroboros, displaying the time a verifier has to wait in order to confirm the best possible double-spending attack, succeeds with probability less than 0.1%. In bitcoin, we take into account a double-spending attacker who controls particular portion of total hashing power and chooses to revert a transaction. The adversary tries to double-spend through a block-withholding attack (where the attacker privately generates a fork and broadcasts it when its length is as along as the longest valid chain). Whereas in Ouroboros, the double-spending adversary tries to brute-force the space of all forks for the current slot leader distribution in a particular segment of protocol and controls a portion of the total stake. We consider both settings for Ouroboros.

There are 2 main assumptions for Ouroboros. First is assuming an arbitrary adversary. And the second is assuming that an adversary does not want to give any indication to any parties about his adversarial behaviour. For e.g., the adversary is not willing to sign two blocks that would belong into the same slot because that would clearly identify him as a malicious party. This is interpreted as "Covertness". The covertly forkable strings are a subclass of the forkable strings having very little density which allows us to provide two distinct security arguments that achieve different trade-offs in terms of efficiency and security guarantees. Our forkable string technique is a natural and general tool that may have applications beyond analysis of our specific Proof-of-Stake protocol.

V. OUROBOROS PRAOS

Ouroboros Praos is an improved version of Ouroboros that achieves a semi-Synchronous communication and fully adversarial adaptive corruption. The tools used to achieve these goals are local & private leader selection, forward secure signatures and implementing hashing to obtain randomness.

A. Semi-Synchronous Communication

The first strengthening factor which enables us to achieve the above mentioned goals is the Semi-Synchronous Communication. Similar to Ouroboros, participants in Ouroboros Praos have synchronized watches and the time is divided into slots. Any messages sent by an honest player is delivered to all honest players but unlike Ouroboros, this happens within at most Δ slots. Adversary may sent arbitrary messages to arbitrary subsets, arriving at arbitrary times, but in Ouroboros Praos, adversary has full control over delays (within the Δ - bound). Δ is unknown to the participants. Security of Ouroboros Praos degrades gracefully with increasing Δ, i.e., with worse and worse network, it is difficult to maintain the same security and performance but degrades gracefully, which did not follow from the analysis of Ouroboros because that analysis only assumes synchronous communication.

B. Fully Adaptive Corruptions

This is the second strengthening factor which helps in achieving the above mentioned goals. Here, the adversary can "immediately" corrupt any party (immediately – being the key word). Which means that, at any point in the execution of the protocol, the adversary can corrupt any party and immediately obtain all of its memory, including secret keys and act on his behalf. This is something that was not captured by original model of Ouroboros, but also would be disastrous for the protocol because once the slot leader schedule is public, then the adversary could corrupt the slot leader and could hijack the chain. So, the only restrictions that the adversary still needs to obey, is that he could corrupt only the minority of the stake throughout the experiment and we still assume that the stake shifts are happening at a bounded rate.

C. Tools

There are 3 tools that are employed to prove persistence and liveness in this doubly strengthened model.

1. Local & Private Leader Selection

If we recall, in Ouroboros we chose the leader schedule publicly, whereas here we replace it with a tool called verifiable random functions which is a cryptographic primitive that fundamentally consists of two procedures: 1. Evaluate. So, the first procedure 'Evaluate' needs a secret key and using this secret key, it takes an input and produces the corresponding output which is unique for the given input and key. It might look random to someone who does not know the secret key. The function also produces a proof that the party owning the secret key can convince anyone else that this is indeed the output for the corresponding input and the secret key.

There is a second procedure called 'Verify' that allows you to check for a particular input, the corresponding output and proof are correct only if the public key is controlled. Below is the VRF, we can use it in a natural way to get a leader selection for deciding slot leaders in a local and private way. So, every stake holder simply has his own secret key for VRF which he plugs into this VRF, the randomness for that epoch along with the slot number and checks the output. If the output is smaller than some threshold that is derived from their particular ratio, then he is the leader for that slot.

Interestingly, the function ∅ that creates the threshold from our frame of stake, needs to be sublinear with respect to the stake as opposed to the case in Ouroboros where the slot leader attribution was public because it helps to make it linearly proportional to our stake, but that's not the case in private leader selection case if we want to maintain independent aggregation. Which means that, if we have a particular pile of stake, it does not help for you to split the stake into several piles and pretend that you are several players to increase the chance of you becoming a leader. If we want to maintain this property, then the function needs to be sub linear. So, we have such a function in Ouroboros Praos.

Verifiable Random Functions:

- $\text{Evaluate}_{sk}\text{(input)} = \text{(output, proof)}$
- $\text{Verify}_{pk}\text{(input, output, proof)} = 0/1$

Leader Selection Lottery for Stakeholder U_i :

$$\text{Evaluate}_{sk}(\text{rnd, slot}) < \varnothing(\text{stake}_i)$$

A similar construction was previously used in the NXT cryptocurrencies and also in Algorand for implementing private leader selection. It is not enough to use an arbitrary VRF generic one, because we need some security even in case when the keys of the VRF are adversarially generated as every player will be generating keys for themselves. So, we don't want corrupted players to be able to win the leader selection lottery by improperly generating the VRF keys. In the paper, we can see a functionality of VRF that is sufficient for this task and we also give an efficient realization of it.

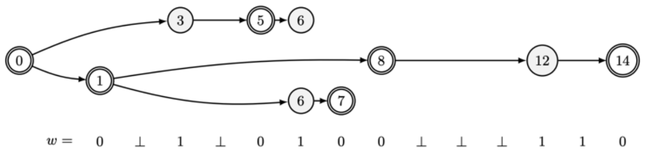


Fig: 8

The interesting fact about this local and private leader election is that it produces empty slots and multi leader since the leaders selection cannot be synchronized throughout all the parties. So, if every party decides locally whether he or she is a slot leader, it means that there will be slots that contain no leaders and there will be slots containing several leaders. To analyse this and show that persistence and liveness is still maintained, we need to make an extension of the forkable strings analysis which is mentioned in the Ouroboros Praos paper. Interestingly now, the ratio of empty slots (basically how many slots turns out to have no slot leader) is a protocol parameter which will be hidden in function ∅. This looks to be useful because such short leaderless periods help us in the analysis in the semi-synchronous setting, as they allow parties to synchronize over the short period of silence. It is important to note that this does not degrade the performance, as now we can aim for much shorter slots. Since our assumptions is not about messages getting delivered within this slot, as it was the case with Ouroboros.

2. Key – Evolving Signatures

The second tool that we use in Ouroboros Praos is the Key-Evolving Signatures (KES). There are digital signature schemes with special property where the public keys are fixed but the secret keys can be evolved in such a way that you can update in every step and in our case it would be every slot. Even if we get hold of anyone's secret key for a particular slot, we won't be able to counterfeit signatures for previous slots. So, if any party deletes their older secret key as prescribed by the protocol, before broadcasting any messages (even when it acts as a slot leader in a certain slot), then even if the party gets immediately corrupted by the adversary, he can no longer create any blocks in the name of this party for past and current slots, because the keys are already deleted or evolved. This helps us achieve adaptive security to manage with adversary that is allowed to do immediate corruptions. Such KES is used for signing blocks in Ouroboros Praos. It also gives UC-functionality for KES that is sufficient for this role and gives a realization of it.

3. Hashing for Dirty Randomness

The final tool is that instead of Ouroboros Praos moving away from NPC to generate randomness for the next epoch for slot leader selection, it turn towards 'hashing', as it is much more efficient but if we recall, there was a basic complaint against hashing stating that adversary could sometimes do rejection sampling. To prevent this, we include an additional VRF value from the leader (which means that if someone turns out to be a leader producing a block, they also have to include a particular VRF value using their secret key for the VRF) into every block. These values are hashed together from the whole past epoch to create the randomness for the next epoch at one shot. This means that with clean randomness, the next epoch (thanks to the static analysis) there would be a negligible probability of violating common prefix or any other desired properties. Then an adversary who is able to perform rejection sampling polynomially many times, cannot increase this probability sufficiently to make it non-negligible. The actual bound in the paper is more nuanced but this argument is sufficient on the asymptotic level. Pr[k-CP violation in string of length n] $\leq ne^{-\Omega(k)}$. These are the three main differences of Ouroboros Praos compared to Ouroboros and the two goals that we achieve by these three changes. Additional details are mentioned in the papers listed in the references section.

VI. FUTURE SCOPE & CONCLUSION

Originally, Ouroboros protocol is the first of its kind that could guarantee rigorous security analysis, and from this came several other protocols that are improvement over Ouroboros (like Ouroboros Praos) which provide better security guarantees in the presence of much stronger adversaries that are capable of delaying the messages on the network to gain control over some portion of the network, deliver protocol messages among honest parties and obtain more control over corrupting honest parties. A recent version of this protocol is Ouroboros Genesis [17] which explains how these protocols, when composed with other protocols then it could provide more functionality on top of the blockchain, running on Proof-of-Stake system, along with addressing a very important issue of bootstrapping from genesis where a participant who wants to join the protocol later, can do it securely by having only a trusted copy of the genesis block without any need of particular guaranteed checkpoints. I conclude this paper by stating that, it is more convincing to employ Proof-of-Stake based blockchain protocols instead of others which require resources (like Proof-of-Work) due to of their obvious merits.

VII. REFERENCES

[1] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II, volume 9057 of Lecture Notes in Computer Science, pages 281–310. Springer, 2015.

[2] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstractly]. SIGMETRICS Performance Evaluation Review, 42(3):34–37, 2014.

[3] Noga Alon and Joel Spencer. The Probabilistic Method. Wiley, 3rd edition, 2008.

[4] Bernardo Machado David, Roman Oliynykov, Aggelos Kiayias, and Alexander Russell. Ouroboros: A provably secure proof-of-stake blockchain protocol. IACR Cryptology ePrint Archive, 2016:889, 2016.

[5] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. J. Cryptology, 23(2):281–343, 2010.

[6] Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively secure, semi-synchronous proof-of-stake protocol. IACR Cryptology ePrint Archive, 2017:573, 2017.

[7] W. Wang et al., "A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks," in IEEE Access, vol. 7, pp. 22328-22370, 2019.

[8] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 557-564.

[9] S. Zhang and J. Lee, "Double-Spending with a Sybil Attack in the Bitcoin Decentralized Network," in IEEE Transactions on Industrial Informatics, vol. 15, no. 10, pp. 5715-5722, Oct. 2019.

[10] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. IACR Cryptology ePrint Archive, 2016:919, 2016.

[11] Silvio Micali. ALGORAND: the efficient and democratic ledger. CoRR, abs/1607.01341, 2016.

[12] Alexander Russell, Cristopher Moore, Aggelos Kiayias, and Saad Quader. Forkable strings are rare. Cryptology ePrint Archive, Report 2017/241, 2017. http://eprint.iacr.org/2017/241.

[13] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In Michael J. Wiener, editor, Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, volume 1666 of Lecture Notes in Computer Science, pages 148–164. Springer, 1999.

[14] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Angelos D. Keromytis, editor, Financial Cryptography, volume 7397 of Lecture Notes in Computer Science. Springer, 2014.

[15] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. CoRR, abs/1507.06183, 2015.

[16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. CoRR, abs/1406.5694, 2014.

[17] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. ACM CCS 2018 ePrint Archive, 2018:378, 2018.