Ouroboros Clepsydra: Ouroboros Praos in Universally Composable Relative Time Model

Handan Kılınç Alper

Web 3.0 Technologies Foundation

Abstract

Ouroboros Praos is a proof of stake based blockchain protocol. One of its security assumptions is parties are synchronized i.e., all of them knows when the protocol passes a new state. However, it is not easy to have such a protocol in real life, especially in a decentralized network. Therefore, we construct a new version of Ouroboros Praos by composing a new protocol called *Relative Time* protocol. We call the new version Ouroboros Clepsydra. At the end of the relative time protocol, a party learns the approximate state of the protocol based on the median of arrival times of messages sent by the other parties and adjusts its local clock based on it. The relative time protocol does not add any new computation to the other parties. They even do not realize that they are part of the relative time protocol. In order to prove Ouroboros Clepsydrain the Universally Composable (UC) model, we define a general UC model to capture the notion of relative time. We remove the synchronization assumption in Ouroboros Clepsydra and show that Ouroboros Clepsydra is a secure proof of stake blockchain protocol in the UC model.

1 Introduction

The first popular decentralized cryptocurrency Bitcoin maintains the public distributed ledger with a proof of work (PoW) consensus mechanism. PoW is an important solution about consensus problem studied in many decades. However, it is a very known fact that PoW consumes a vast amount of energy. Therefore, the blockchain society has explored new solutions that replace PoW with a more energy-friendly mechanism while preserving the decentralized features. *Proof of stake* (PoS) is the most promising replacement among the other solutions appeared [12, 2, 13, 1]. It has the same nature of PoW which selects the block producers randomly proportional to their stakes while in PoW, block producers are selected randomly proportional to their computational power.

One of the critical issues in PoS is constructing a selection mechanism which cannot be biased by an adversary. This issue is carefully considered by current PoS protocols. Some important results for an unbiased random selection based on random oracle [18, 9], a publicly verifiable secret sharing (PVSS) scheme [17], verifiable random function (VRF) [10, 8, 15, 16], and threshold cryptography [16]. All parties in all solutions decide if they are eligible to produce a block in a certain time interval with the unbiased randomness. However, the question is how this party know whether (s)he is in the right time interval. This is another issue of PoS which has not been formally treated before. Indeed, it is very critical for an honest party to release his/her block in the right time for security. If (s)he is late, then the

next producers continue to build the chain without seeing the honest party's block. As a result, the honest party may not have chance to contribute building on the chain. If (s)he is early, similar problem arises.

The timeline of parties in Ouroboros [17] and Ouroboros Praos [10] is divided into slots where each slot may or may not have a block producer. The security is based on the fact that all parties know when the slot starts and ends. Ouroboros Genesis [3] protocol which is similar to Ouroboros Praos except with the chain selection rule mentions that the violation of this assumption breaks the security. The timeline of parties in the Dfinity consensus protocol [16] progresses based on certain events but it is not clear how everyone sees a certain event occurred at the same time in a partially synchronous network. Algorand [8, 15] is a different consensus algorithm which runs a Byzantine agreement protocol for each block. Parties can trust all blocks and adjust their local clock according to the round inside the blocks. We note that this is not possible in Ouroboros [17], Ouroboros Praos [10] or Ouroboros Genesis [3] because a party cannot know if the block is from an honest party or sent during the correct slot. Snow White [9, 18] is the only protocol considering this timing issue in their analysis by adding up the maximum time difference between parties into the network delay. However, they do not propose any protocol to obtain clocks with this maximum difference.

One solution to overcome the synchronization problem is a central clock which provides the same time to all parties. However, the corruption of this clock breaks the whole mechanism. In addition to this, it is not very logical to assume a central trusted clock while trying to build a fully decentralized system. The timestamp is another solution but it requires an analysis of how to trust the timestamp in a system where it is not clear who is honest or malicious.

In this paper, we focus on the synchronization problem of Ouroboros Praos [10]. We replace the assumption about knowing the current slot with another assumption by being Θ-slot behind the real slot number. We call the new version of Ouroboros Praos with our new security model Ouroboros Clepsydra. We model this assumption in the general universally composable (GUC) model for relative time. We also propose an algorithm that realizes our slot number assumption in the real world. In more details, our contribution in this paper is as follows:

Our Contribution:

- We construct a general universally composable (GUC) model that captures the notion relative time. Our model consisting of a global functionality which defines the clock rate globally and functionality for local clocks which does not necessarily follow the same rate to capture the notion of a drifted clock. Our other global functionality keeps track of the actual slot number according to the clock rate. The parties can contact with this global functionality to learn the slot number but may not get the correct one. To the best of our knowledge, our GUC model is the first model which models the relative time. The closest GUC model to ours is by Canetti et al. [7] but their relative time notion and ours differs because their model is constructed for server/client protocols.

- We modify Ouroboros Praos [10] such that newly joining parties obtain the slot number from the global functionality we define. Since the obtained slot number may not be correct, some of the parties may not release their blocks in the right slot. Considering these type of parties, we formally analyze the security of the new version of Ouroboros Praos:Ouroboros Clepsydra.

- We propose an algorithm called *Relative Time* algorithm that realizes the global functionality providing the slot number. Our algorithm does not add any extra computation to the algorithms of block producers. The algorithm outputs a slot number by using the arrival times of the blocks. Our algorithm outputs a slot number which is $\Delta + 1$ less than the actual slot number. If the clock is not drifted then the output is Δ less than the actual one. Here, Δ is the maximum network delay in the partial-synchronous network. Our security analysis in Relative time does not depend on any specific functionality that Ouroboros Clepsydra has. Therefore, our algorithm can be adopted by similar types of PoS protocols [9, 3, 16].

Structure of the Paper: In Section 2, we give some preliminaries related to UC model, some functionalities defined in Ouroboros [17] and Ouroboros Praos [10] and the blockchain structure. We define our new GUC model for relative time in Section 3 and give the UC model defined in [10] for partial-synchronous networks with minor changes. In Section 4, we describe Ouroboros Clepsydra in the hybrid model and prove its security. Finally, we give a relative time protocol and prove its security in our new GUC model in Section 5.

2 Preliminaries

Notations: We denote by B a blockchain block and by C a blockchain. Given two blockchains C_1 and C_2, $C_1 \preceq C_2$ means that C_1 is a prefix of C_2. $C^{\lceil k}$ is a blockchain after cutting the last k blocks.

We use $\mathcal{D}$ to define a distribution. $x \leftarrow \mathcal{D}$ shows that x is selected with respect to the distribution $\mathcal{D}$.

Two ensembles $X = \{X_1, X_2, ..., X_n\}$ and $Y = \{Y_1, Y_2, ..., Y_n\}$ are computationally indistinguishable if for all probabilistic polynomial time (PPT) algorithms D and for all $c > 0$, there exits an integer N such that for all $n \geq N$

$$|\mathsf{Pr}[\mathsf{D}(X_i) = 1] - \mathsf{Pr}[\mathsf{D}(Y_i) = 1]| < \frac{1}{n^c}.$$

The notation $\approx$ is used to show that two ensembles are computationally indistinguishable (i.e. $X \approx Y$).

2.1 Universally Composable (UC) Model:

UC model consists of an ideal functionality that defines the execution of a protocol in an ideal world where there is a trusted entity. The real-world execution of protocol (without a trusted entity) is called UC-secure if running the protocol with the ideal functionality $\mathcal{F}$ and in the real-world is indistinguishable by any external environment $\mathcal{Z}$.

A protocol π is defined with distributed interactive Turing machines (ITM). Each ITM has an inbox collecting messages from other ITMs, adversary $\mathcal{A}$ or environment $\mathcal{Z}$. Whenever an ITM is activated by $\mathcal{Z}$, the ITM instance (ITI) is created. We identify ITI's with their identifier consisting of a session identifier *sid* and the party identifier P.

- π in Real World: $\mathcal{Z}$ initiates all or some ITM's of π and the adversary $\mathcal{A}$ to execute an instance of π with the input $z \in \{0,1\}^*$ and the security parameter κ. The output of a

protocol execution in the real world is denoted by $\mathsf{EXEC}(\kappa, z)_{\pi,\mathcal{A},\mathcal{Z}} \in \{0,1\}$. Let $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathsf{EXEC}(\kappa, z)_{\pi,\mathcal{A},\mathcal{Z}}\}_{z\in\{0,1\}^*}$.

- π in Ideal World: Ideal world consists of an incorruptible ITM $\mathcal{F}$ which executes π in an idealistic way. The adversary $\mathcal{S}$ (called simulator) in the ideal world has ITMs which forwards all messages provided by $\mathcal{Z}$ to $\mathcal{F}$. These ITMs can be considered as corrupted parties and they are known by $\mathcal{F}$. The output of π in the ideal world is denoted by $\mathsf{EXEC}(\kappa, z)_{\mathcal{F},\mathcal{S},\mathcal{Z}} \in \{0,1\}$. Let $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\mathsf{EXEC}(\kappa, z)_{\mathcal{F},\mathcal{S},\mathcal{Z}}\}_{z\in\{0,1\}^*}$.

$\mathcal{Z}$ outputs whatever the protocol in the real world or ideal world outputs. We refer [4, 5] for further details about the UC-model.

Definition 1. *(UC-security of π) Let π be the real-world protocol and $\mathcal{F}$ be the ideal-world functionality of π. We call π UC-realizes $\mathcal{F}$ (π is UC-secure) if for all PPT adversaries $\mathcal{A}$ in the real world, there exists a PPT simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$,*

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$$

π in Hybrid World: In the hybrid world, the parties in the real world interact with some ideal functionalities. We say that a protocol π in hybrid world UC-realizes $\mathcal{F}$ when π consists of some ideal functionalities.

We define relative time model of Ouroboros Clepsydra in *generalized UC model* [6]. GUC model formalizes the global setup in UC-model. In GUC model, $\mathcal{Z}$ can interact with arbitrary protocols and ideal functionalities $\mathcal{F}$ can interact with GUC functionalities $\mathcal{G}$.

2.2 Primitives

In this section, we give the functionalities used in Ouroboros Clepsydra in the hybrid model. We do not give the details of them because the details are not required to understand Ouroboros Clepsydra.

Ideal Signature Scheme $\mathcal{F}_{\mathsf{DSIG}}$: $\mathcal{F}_{\mathsf{DSIG}}$ is defined in [17] that defines an ideal signature scheme. They show that any EUF-CMA secure signature scheme realizes $\mathcal{F}_{\mathsf{DSIG}}$. $\mathcal{F}_{\mathsf{DSIG}}$ provides a public key pk_i^s to a party P_i after receiving the message $(\mathsf{KeyGen}, sid, P_i)$. After the key generation, when $\mathcal{F}_{\mathsf{DSIG}}$ receives a message $(\mathsf{Sign}, sid, P_i, m)$ from P_i for signing, it provides the signature by sending $(\mathsf{Signature}, sid, P_i, m, \sigma)$. $\mathcal{F}_{\mathsf{DSIG}}$ verifies a signature when it receives a message $(\mathsf{Verify}, sid, P_i, \sigma, \mathsf{pk}^s)$. The details of $\mathcal{F}_{\mathsf{DSIG}}$ is in [17].

Ideal Forward Secure Signature $\mathcal{F}_{\mathsf{KES}}$: David et al. [10] define $\mathcal{F}_{\mathsf{KES}}$ for signature schemes with evolved keys. In $\mathcal{F}_{\mathsf{KES}}$, the signatures generated in the past with an old key are not valid signatures anymore. The key generation of $\mathcal{F}_{\mathsf{KES}}$ is same as $\mathcal{F}_{\mathsf{DSIG}}$. Differently, a party sends a message $(\mathsf{USign}, sid, P_i, m, j)$ for signing with the parameter j which indicates the total number of signatures so far. $\mathcal{F}_{\mathsf{KES}}$ verifies the signature of a party if its index is equal to or greater than j. The details of $\mathcal{F}_{\mathsf{KES}}$ is in [10].

Ideal Verifiable Random Function $\mathcal{F}_{\mathsf{VRF}}$: A verifiable random function (VRF) [11] generates an unpredictable random number with an input and a secret key. The construction of this random number should be provable with corresponding secret key i.e., the random number is generated with the public input and the secret key. David et al. define a functionality for a VRF ($\mathcal{F}_{\mathsf{VRF}}$) which generates an unpredictable random number under malicious key generation. It briefly works as follows:

A party P_i sends a message $(\mathsf{KeyGen}, sid, P_i)$ to $\mathcal{F}_{\mathsf{VRF}}$ to obtain a verification key. Then, $\mathcal{F}_{\mathsf{VRF}}$ gives the verification key pk_i^v with the message $(\mathsf{VerificationKey}, sid, P_i, \mathsf{pk}_i^v)$ and creates a table to keep the tuples $(\mathsf{pk}_i^v, m_i, d_i)$ (i.e., the unpredictable random number d_i corresponding to the input m_i and the key pk_i^v). After the key generation, a party P_i can contact with $\mathcal{F}_{\mathsf{VRF}}$ to obtain a random number and a proof that corresponds the input provided by P_i. For this, P_i sends a message $(\mathsf{EvalProve}, sid, P_i, m, \mathsf{pk}_i^v)$. If m is not given by P_i before, $\mathcal{F}_{\mathsf{VRF}}$ chooses a random number d and asks for a unique proof π from the adversary $\mathcal{S}$. Then, $\mathcal{F}_{\mathsf{VRF}}$ gives $(\mathsf{Evaluated}, sid, P_i, m, d, \pi)$ to P_i. $\mathcal{F}_{\mathsf{VRF}}$ also verifies the proof when it receives a message $(\mathsf{Verfiy}, sid, P_i, m, d, \pi, \mathsf{pk}_i^v)$ from any party. It verifies if the table includes a tuple (pk_i^v, m, d). Even if the adversary $\mathcal{S}$ asks for an evaluation of an input m, $\mathcal{F}_{\mathsf{VRF}}$ makes sure that the random number corresponding m and the public key of $\mathcal{S}$ is unique. We refer to [10] for further details of this functionality.

2.3 Blockchain

Definition 2 (Block, Blockchain)**.** *The first block of a blockchain is called genesis block. Given a* genesis block B_0*, a block B_i is collection of data having the information of another block that is connected to. All blocks is connected to* only one *block. We call that B_i is connected to a block B_{i-1} if the data related to parent of B_i is $H(B_{i-1})$ where H is a hash function. A* blockchain $C = B_0||B_1||...||B_{i-1}||B_i$ *is a data structure consisting of connected blocks.*

A protocol that defines the construction of a blockchain is called a blockchain protocol. Garay et al. [14] define the properties defined below in order to obtain a secure blockchain protocol. If a blockchain protocol satisfies these properties, it is called secure. In all definitions below, slot represents time. The formal definition of the notion of slot is in Section 3.1.

Definition 3 (Common Prefix (CP) Property [14])**.** *The common prefix property with parameters $k \in \mathbb{N}$ ensures that any blockchains C_1, C_2 owned by two honest parties at the onset of the slots $sl_1 < sl_2$ satisfy that $C_1^{\lceil k} \preceq C_2$.*

In other words, the CP property ensures that the blocks which are k-blocks before the last block of an honest party's blockchain cannot be changed. We call all unchangeable blocks *finalized* blocks.

Definition 4 (Chain Growth (CG) Property [14])**.** *The chain growth property with parameters $\tau \in (0, 1]$ and $s \in \mathbb{N}$ ensures that if a blockchain owned by an honest party at the onset of some slot sl_u is C_u, and a blockchain owned by another honest party at the onset of slot $sl_v \geq sl_s v + s$ is C_v, then the difference between the length of C_v and C_u is greater than or equal to τs.*

In other words, the CG property guarantees a minimum growth after s slots later. τ defines how fast the blockchain grows.

Definition 5 (Chain Quality (CQ) Property [14])**.** *The chain quality property with parameters $\mu \in (0, 1]$ and $k \in \mathbb{N}$ ensures that the ratio of honest blocks in any k length portion of an honest blockchain is μ.*

The CQ property ensures sufficient honest block contribution to any blockchain owned by an honest party.

3 UC-Model for Ouroboros Clepsydra

In this section, we describe the GUC-model for relative time and the partial synchronous network for Ouroboros Clepsydra.

The ITMs of Ouroboros Clepsydra in the hybrid world are called stakeholder and each ITM P_i is denoted by P_i. $\mathcal{Z}$ activates all stakeholders in order and the adversary and creates ITIs. $\mathcal{A}$ is also activated whenever the ideal functionalities invoke.

We first introduce our new GUC-model for the relative time notion and then give the UC-model for partial synchronous network similar to the model in [10, 3].

3.1 GUC-Model of Relative Time

GUC network time model is introduced by Canetti et al. [7] which models the clock adjustment of a client with servers of which can be corrupted. Now, we consider a similar model in a network where relative time is critical. We do not have a client or a server in this model instead we have parties who send messages (e.g., blocks) regularly according to their clock rate. We use the notion of a slot that is defined for a specific time interval. A slot is a metric time in the metric system in the real world. Therefore, we consider it as the same notion of time in the network time model [7].

In a nutshell, we want to define a GUC-model where each party immediately accesses to its local clock which may not progress as the same rate of other parties' local clock. The rate of all local clocks is defined in a global setup but the local clocks may not follow the same rate while they progress. In our model, the only absolute value is slot number. The rest is based on relative time.

Remark that the main difference in our model and the GUC-network model [7] is the definition of relative time. If we consider slot as the same as clock output in [7], the relative time means how many slots passed. However, in our model, the relative time means how much a local clock is progressed.

GUC-Model of relative time consists of the following functionalities:

Reference Rate ($\mathcal{G}_{\mathsf{refRate}}$) : This functionality defines a global clock rate. The parties who wants to be notified in every progress of time register to $\mathcal{G}_{\mathsf{refRate}}$. Whenever $\mathcal{Z}$ contacts with $\mathcal{G}_{\mathsf{refRate}}$, $\mathcal{G}_{\mathsf{refRate}}$ signals it all registered parties as a sign of time progress. The difference between $\mathcal{G}_{\mathsf{refClock}}$ [7] and $\mathcal{G}_{\mathsf{refRate}}$ is that $\mathcal{G}_{\mathsf{refRate}}$ does not have any notion of clock or time. It just lets the all functionalities increase their own clock. The details are in Figure 1.

Local clock ($\mathcal{F}_{local}^{\Sigma,P}$): The local clock functionality $\mathcal{F}_{local}^{\Sigma,P}$ represents a local clock of a party. $\Sigma \in \mathbb{N}$ represents how much the time is drifted. This clock is accessible by the party P without any delay. $\mathcal{F}_{local}^{\Sigma,P}$ stores two local clocks: $\mathsf{localclock}_{\mathcal{Z}}$ and $\mathsf{localclock}$. It increments $\mathsf{localclock}_{\mathcal{Z}}$ whenever it receives a message from $\mathcal{Z}$ and increments $\mathsf{localclock}$ whenever it receives $\mathcal{G}_{\mathsf{refRate}}$.

Functionality $\mathcal{G}_{\mathsf{refRate}}$

$\mathcal{G}_{\mathsf{refRate}}$ has a list of registered parties ListRate which is initially empty.

- (Registration) If a party P sends a message $(\mathsf{Register}, sid, P)$, $\mathcal{G}_{\mathsf{refRate}}$ adds P to ListRate and sends $(\mathsf{Registered}, sid, P)$ to P.
- (Progress in Time) When $\mathcal{Z}$ sends a command Continue, $\mathcal{G}_{\mathsf{refRate}}$ sends $(\mathsf{Increase}, sid', P)$ to all $P \in$ ListRate

Figure 1: The global functionality $\mathcal{G}_{\mathsf{refRate}}$

Functionality $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$

$\mathcal{Z}$ initiates $\mathcal{F}_{local}^{\Sigma,P}$ with a value counter. Then, $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ creates two parameters $\mathsf{localclock}_{\mathcal{Z}} = \mathsf{counter}$ and $\mathsf{localclock} = \mathsf{counter}$.

- (Registration) $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ sends $(\mathsf{Register}, sid, \mathcal{G}_{\mathsf{Slot}}^{T,\Theta})$ and receives back $(\mathsf{Registered}, sid, P)$. This is done once after initialization of the functionality.
- (Increasing $\mathsf{localclock}_{\mathcal{Z}}$) When $\mathcal{Z}$ sends $(\mathsf{Increase}, sid, P)$, $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ increases $\mathsf{localclock}_{\mathcal{Z}}$ and lets $\Sigma = \mathsf{localclock} - \mathsf{localclock}_{\mathcal{Z}}$.
- (Increasing localclock) When $\mathcal{G}_{\mathsf{refRate}}$ sends $(\mathsf{Continue}, sid, \mathcal{F}_{\mathsf{local}}^{\Sigma,P})$, $\mathcal{F}_{local}^{\Sigma,P}$ increases $\mathsf{localclock}_{\mathcal{Z}}$ and lets $\Sigma = \mathsf{localclock} - \mathsf{localclock}_{\mathcal{Z}}$.
- If P sends $(\mathsf{get\_clock}, \mathsf{sid}, P)$, $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ sends $(clock, sid, \mathsf{localclock}_{\mathcal{Z}}, P)$.

Figure 2: The global functionality $\mathcal{G}_{\mathsf{refRate}}$

However, it never shares localclock with the party. The reason of having localclock is to keep the amount of total drift (Σ) in any time. We define it in Figure 2.

One can see the functionality $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ as a very powerful functionality than the real-world local clocks since it has the information localclock. However, in the realization of $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ in the real world, we do not need to simulate the parameter localclock or Σ because it is never shared with the parties and its execution of $\mathsf{localclock}_{\mathcal{Z}}$ does not depend on it. So, a usual local clock in the real world can be proven as a realization of $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$.

Slot Provider ($\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$): $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ has a similar role as $\mathcal{G}_{\mathsf{Clock}}$ in the network time model [7]. $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ provides a slot number to a requester party. When a party asks about the current slot number, $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ asks to $\mathcal{A}$ what to send to the party. If $\mathcal{A}$ provides a slot number which is less than $sl_{curr} - \Theta$ or greater than sl_{curr} where sl_{curr} is the current slot number, $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ sends sl_{curr} instead of the slot number given by $\mathcal{A}$. Otherwise, it sends the adversarial slot number. $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ updates the current slot number every T message from $\mathcal{G}_{\mathsf{refRate}}$.

We note $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ is less restrictive functionality than $\mathcal{G}_{\mathsf{Clock}}$ because we model the functionality in the partially-synchronized network (Section 3.2). It means that it always responds to a party with a slot number. Differently than $\mathcal{G}_{\mathsf{Clock}}$, $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ does not access a global reference clock $\mathcal{G}_{\mathsf{refClock}}$ instead it access to $\mathcal{G}_{\mathsf{refRate}}$ which actually define the rate of slot number change. The description of $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ in Figure 3.

Functionality $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$

$\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ is parameterized with T, Θ given by $\mathcal{Z}$. $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ first sends $(\mathsf{Register}, sid, \mathcal{G}_{\mathsf{Slot}}^{T,\Theta})$ and receives back $(\mathsf{Registered}, sid, P)$. Then, it sets $sl_{curr} = 0$ and $\mathsf{counter} = 0$.

- If $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ receives an input $(\mathsf{req\_slot}, sid, P)$, it relays the input to the adversary and waits for the adversary's response. $\mathcal{A}$ replies with $(\mathsf{new\_slot}, sid, sl', P')$. Given that the current slot number is sl_{curr}, if $sl_{curr} - sl' \geq \Theta$ or $sl_{curr} < sl'$, $sl = sl_{curr}$. Otherwise, $sl = sl'$. $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ sends $(\mathsf{current\_slot}, sid, sl)$ to P_j.
- When $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ receives $(\mathsf{Continue}, sid, \mathcal{G}_{\mathsf{Slot}}^{T,\Theta})$, it increases $\mathsf{counter}$. If $\mathsf{counter} \mod T = 0$, it increase s_{curr}.

Figure 3: The global functionality $\mathcal{G}_{\mathsf{Slot}}^{T}$

3.2 UC- Partially Synchronous Network Model

Our network model $\mathcal{F}_{\mathsf{DDiffuse}}$ is similar to the model of Ouroboros Praos [10, 17, 14]. Differently, it accesses to $\mathcal{G}_{\mathsf{refRate}}$ in order to have the notion of relative time.

$\mathcal{A}$ can corrupt stakeholders $P_1, P_2, ..., P_n$ or desynchronize them if $\mathcal{Z}$ permits it. When $\mathcal{Z}$ permits a corruption of a party P_i, it sends the message $(\mathsf{Corrupt}_i, P_i)$. If a party is corrupted, whenever it is activated, its current state is shared with $\mathcal{A}$. If $\mathcal{Z}$ permits desynchronization of P_i, it sends the message (Desync, P_i) to $\mathcal{A}$. If P_i is desynchronized, P_i contacts with $\mathcal{G}_{\mathsf{Slot}}$ to get the slot number[1]. This is the critical difference between the security model in Ouroboros Praos [10] and our security model.

Now, we define the functionality $\mathcal{F}_{\mathsf{DDiffuse}}$ which models a partially synchronous network which at most Δ-slots delay.

$\mathcal{F}_{\mathsf{DDiffuse}}^{T}$: The message handling functionality $\mathsf{Diffuse}$ first introduced by Garay et al. [14] in the synchronous network. Then, David et al. [10] define a new functionality "delayed diffuse" ($\mathsf{DDiffuse}$) that realizes a partially synchronous network where the messages of a stakeholder P_i read by others at most Δ-slots later. $\mathsf{DDiffuse}$ does not need the notion of relative time of Δ-slot while $\mathcal{F}_{\mathsf{DDiffuse}}^{T}$ needs because slot is not defined based on time in $\mathsf{DDiffuse}$. Therefore, we combine $\mathsf{DDiffuse}$ with the $\mathcal{G}_{\mathsf{refRate}}$ and obtain $\mathcal{F}_{\mathsf{DDiffuse}}^{T}$:

$\mathcal{F}_{\mathsf{DDiffuse}}^{T}$ first registers to $\mathcal{G}_{\mathsf{refRate}}$ and creates a local clock ℓ which is 0 at the beginning. Whenever $\mathcal{G}_{\mathsf{refRate}}$ sends a message with $\mathsf{Continue}$, it increments ℓ.

Each honest P_i can diffuse only one message in one slot. P_i can access its inbox anytime. $\mathcal{A}$ can read all messages diffused by stakeholders and decide their delivery order before they arrive to inboxes of uncorrupted stakeholders. For any message comes from an uncorrupted stakeholder, $\mathcal{A}$ can label it as $\mathsf{delayed}_i$. When $\mathcal{F}_{\mathsf{DDiffuse}}^{T}$ receives $\mathsf{delayed}_i$ for a message to P_i, it marks it with the current local time ℓ_i. A delayed message is not moved to the inbox of the corresponding stakeholder until $\mathcal{A}$ let $\mathcal{F}_{\mathsf{DDiffuse}}^{T}$ move it to the inbox or the local time reaches $\ell_i + \Delta T$. When $\mathcal{F}_{\mathsf{DDiffuse}}$ receives $(\mathsf{Create}, P, \mathcal{C})$ from $\mathcal{Z}$, it creates a new stakeholder P with the initial chain $\mathcal{C}$.

Before starting the next section, we give the q-bounded random oracle model [14].

[1] We note that desynchronization of a party is not related to network. It is related to notion of slot number.

$\mathcal{F}_{\mathsf{RO}}$**:** $\mathcal{F}_{\mathsf{RO}}$ works different than the random oracle model. It bounds the number of queries that an $\mathcal{A}$ can make in each slot. It is first introduce by Garay et al. [14] in q-bounded model. Given a slot where $\mathcal{A}$ corrupts t stakeholders, $\mathcal{F}_{\mathsf{RO}}$ sets the parameter that defines the bound as tq and sets the counter as 0. Whenever $\mathcal{F}_{\mathsf{RO}}$ receives an input ρ then it runs the following algorithm:

$\underline{\mathcal{G}_{\mathsf{RO}}(\rho)}$
counter $\leftarrow$ counter+1
if counter $\leq tq$:
  if $(., \rho) \in T$:
    return r where $(r, \rho) \in T$
  else:
    $r \leftarrow_r \mathcal{U}$
    add (r, ρ) to T
    return r
else if:
  return $\perp$

Here, $\mathcal{U}$ is a uniform distribution.

4 Ouroboros Clepsydra

Ouroboros Clepsydra is a new version of Ouroboros Praos with stakeholders who do not necessarily know what current slot number is. Ouroboros Clepsydra consists of sequential non-overlapping epochs $(e_1, e_2, ..., e_E)$, each of which consists of a number of sequential block production slots $(e_i = \{sl_1^i, sl_2^i, \ldots, sl_R^i\})$ up to some bound $R \in \mathbb{N}$.

We give the Ouroboros Clepsydra protocol in the hybrid model with the functionalities $\mathcal{F}_{\mathsf{VRF}}$, $\mathcal{F}_{\mathsf{KES}}$, $\mathcal{F}_{\mathsf{DSIG}}$, $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{local}}^{P,\Sigma}$ and $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$. An instance of this protocol is in Appendix A. We believe that this instance makes the protocol more clear for ones who are not familiar with the UC-model [4, 5].

Before giving the details of Ouroboros Clepsydra we define the probability of being selected as a slot leader:

$$p_i = \phi_c(\alpha_i^m) = 1 - (1 - c)^{\alpha_i^m}$$

where α_i^m is the relative stake of the stakeholder P_i in an epoch e_m and $c \in (0, 1)$ is a constant parameter of the protocol that is chosen according to how frequently we want to have slots with no slot leader. As it can be seen, each stakeholder is selected proportional to its stake. Remark that the function ϕ has the "independent aggregation" property as remarked in [10], which informally means the probability of being selected as a slot leader does not increase as a stakeholder splits his stakes across virtual stakeholders.

ϕ is used to set a threshold τ_i for each stakeholder P_i.

$$\tau_i = 2^{\ell_{vrf}} \phi_c(\alpha_i^m) \tag{1}$$

where ℓ_{vrf} is the length of the unbiased random number outputted by VRF. The detailed description of Ouroboros Clepsydra in the hybrid model is given in Figure 4. The overview is as follows:

Overview of Ouroboros Clepsydra: It consists of three phases:

The first phase is the *initialization phase* which has two different modes. The first mode continues until $\mathcal{Z}$ provides the genesis block. $\mathcal{Z}$ creates n stakeholder instances with some amount of stakes. The parties who are activated before the genesis block contact with the functionalities $\mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}$ and obtain the public keys $\mathsf{pk}^v, \mathsf{pk}^e, \mathsf{pk}^s$, respectively. Then, they register to $\mathcal{F}_{\mathsf{init}}$ with the public keys and the stake. When $\mathcal{Z}$ commands $\mathcal{F}_{\mathsf{init}}$, $\mathcal{F}_{\mathsf{init}}$ creates the genesis block and gives it to the parties who has already registered. Similarly, $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ starts to functioning. When a party receives the genesis block, it asks the current local time t_1 from $\mathcal{F}_{\mathsf{local}}^{P,\Sigma}$ and sets $sl = sl_1^1$. It then stores $\Pi = (sl_1^1, t_1)$ as a reference point for its own slot counter. The second mode is for new parties who join after the genesis block. They first register their public keys as in the first mode before the genesis released. After the registration, they learn the current slot number. For this, they contact with $\mathcal{G}_{\mathsf{Slot}}^{T,\Sigma}$ and obtain the slot number sl_{ref}. Immediately, they contact with $\mathcal{F}_{\mathsf{local}}^{P,\Sigma}$ to obtain the current local time t_{ref}. As in the first mode, they store $\Pi = (sl_{ref}, t_{ref})$ as a reference point to compute the current slot number later.

The second phase is the *chain extension* phase where parties construct blocks if they are the slot leaders, collect all chains arrive to their inbox, choose the best chain. A party in this phase learns the current slot number sl_{curr} by running the algorithm $\mathsf{S\_Map}(\Pi, t_{cur})$ where t_{cur} is the current local time obtained from $\mathcal{F}_{\mathsf{local}}^{P,\Sigma}$.

Algorithm 1 $\mathsf{S\_Map}(\Pi, t_{cur})$

$\Pi = (sl_{ref}, t_{ref})$
$dif = \lfloor \frac{t_{curr} - t_{ref}}{T} \rfloor$
return $sl_{ref} + dif$

When stakeholders receive blockchains, they check the validity of these blockchains as described in Figure 4 and add all valid ones to the set $\mathbb{C}$. After collecting all chains, they choose the best chain among the all chains they received with following algorithm.

Algorithm 2 $\mathsf{maxvalid}(\mathbb{C}, C_{best})$

$C_{max} \leftarrow C_{best}$
for all $C_{cand} \in \mathbb{C}$ **do**
    if $C_{cand} > C_{max} \wedge C_{best}^{\lceil k} \preceq C_{cand}$ **then**
        $C_{max} \leftarrow C_{cand}$
return C_{max}

They also check if they are a slot leader by sending the input $(r_m || sl_{cur})$ to $\mathcal{F}_{\mathsf{VRF}}$. r_m is the randomness beacon of the current epoch m. We explain how the randomness beacon is generated in the next phase. $\mathcal{F}_{\mathsf{VRF}}$ provides a random number d and a proof π. If d is less than the threshold defined in Equation 5, then the party is selected as a slot leader of sl_{cur}. Slot leaders generate a block B as described in Figure 4 and append it to their C_{best}. Then, they give $C_{best} || B$ to $\mathcal{F}_{\mathsf{DDiffuse}}^{T}$ for the distribution.

The last phase is *epoch formation*. In this phase, stakeholders update the parameters the randomness beacon and the stake distribution of the next epochs. The genesis block includes the randomness beacon for the first two epochs e_1 and e_2 and the stake distribution for the first three epoch e_1, e_2 and e_3. The stake distribution of an epoch e_m where $m > 3$ can be

obtained from the blocks belong to epoch e_{m-3}. The randomness beacon of an epoch e_m where $m > 2$ is obtained from the VRF values of blocks in epoch e_{m-2}. Each party sends the VRF values $\rho_m = d'_1||d'_2||...|d'_\ell$ to $\mathcal{F}_{\mathsf{RO}}$. The output of $\mathcal{F}_{\mathsf{RO}}$ is the randomness beacon r_m of the epoch e_m.

4.1 Security Analysis

Our security analysis of Ouroboros Clepsydra in hybrid model follows the same structure as in Ouroboros Praos [10]. The critical difference of our security analysis is that we have two type of honest parties: honest and synchronized parties and honest but late parties. Assume that the current slot number is kept by $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ is sl_{curr} which starts at $\mathsf{counter} = t$ and ends when counter reaches $t + T$.

- Honest and Synchronized Parties: We say that a party is synchronized if there exists $t_x \in [t, t+T]$ such that the party outputs sl_{curr} when the counter of $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ is equal to t_x.
- Honest but Late Parties: These parties output sl where $sl_{curr} - \Theta \leq sl < sl_{curr}$ in a time interval between t and $t + T$.

Remark that honest and synchronized parties do not have to output sl_{curr} between whole interval $[t, t + T]$. Only one moment that share the same slot number is sufficient in our security analysis.

Before starting the security analysis, we remind that in Ouroboros Praos [10], $\Theta = 0$ and in Ouroboros [17], $\Delta = \Theta = 0$. Therefore, we need to modify some definitions given by [10, 17]. One of these definitions is the notion of characteristic string. In order the differentiate, we call our characteristic string unsynchronized characteristic string and the one in Ouroboros Praos [10] synchronized characteristic string. Note that synchronization is not related here with the network. It is related with the synchronization of slot numbers among honest parties.

Definition 6 (Unsynchronized Characteristic String)**.** *Assume that the set $\mathcal{L} = \{L_1, L_2, ..., L_n\}$ has elements of which each one L_i includes the slot leader(s) of each slot sl_i. A unsynchronized character of an element in $\mathcal{L}$ is defined over a function $f_{chr} : \mathcal{L} \to \{0_s, 0_\ell, 1, \perp\}$ as follows:*

$$f_{chr}(L_i) = \begin{cases} \perp & \text{if } L_i = \emptyset \\ 0_s & \text{if } |L_i| = 1 \text{ and the slot leader is in } \mathcal{H}_s \\ 0_\ell & \text{if } |L_i| = 1 \text{ and the slot leader is in } \mathcal{H}_\ell \\ 1 & \text{otherwise.} \end{cases}$$

where $\mathcal{H}_\ell$ is the set of honest but late parties and $\mathcal{H}_s$ is the set of honest and synchronized parties.

The unsynchronized characteristic string of $\mathcal{L}$ is $w = w_1 w_2 ... w_n$ where $w_i = f_{chr}(L_i)$.

A *fork* F is a directed graph which consists of vertexes ($\mathcal{V}$) and edges ($\mathcal{E}$). One of the vertexes is a special vertex called root. All edges are directed from the root (similar to the tree structure). We call each path from the root as *tine*. $\Delta\Theta$-fork is a specific fork which consructed based on the rules derived from a characteristic string $w \in \{0_s, 0_\ell, 1, \perp\}^n$. In the following definition, we give these rules. $\Delta\Theta$-fork is very similar to the Δ-fork definition in [10]. We have an extra property to capture 0_ℓ values in w.

Ouroboros Clepsydra in Hybrid Model

The protocol consists of sequential non-overlapping epochs $(e_1, e_2, ..., e_E)$, each of which consists of a number of sequential block production slots $(e_i = \{sl_1^i, sl_2^i, \ldots, sl_R^i\})$ up to some bound R. Stakeholders $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ send messages to each other with the functionality $\mathcal{F}_{\mathsf{DDiffuse}}^{T}$ and communicate with other functionalities $\mathcal{F}_{\mathsf{init}}, \mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{local}}^{P,\Sigma}$ and $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$.

Phase 1: Initialization Phase

- A stakeholder P_j before starting sl_1^1 (first slot of the first epoch) sends the message $(\mathsf{register}, sid, P_j, \mathsf{pk}_j^v, \mathsf{pk}_j^e, \mathsf{pk}_j^s, st_j)$ where $\mathsf{pk}_j^v, \mathsf{pk}_j^e, \mathsf{pk}_j^s$ are the verification keys obtained from respectively $\mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}$ and st_j is the stake of P_j. If the message of P_j is correct, $\mathcal{F}_{\mathsf{init}}$ adds it to the genesis list. After $\mathcal{Z}$ commands $\mathcal{F}_{init}$ to generate the genesis block, $\mathcal{F}_{\mathsf{init}}$ does not accept any registration request and creates the genesis block $B_0 = \{r_0, (P_1, \mathsf{pk}_1^v, \mathsf{pk}_1^e, \mathsf{pk}_1^s, st_1), (P_1, \mathsf{pk}_2^v, \mathsf{pk}_2^e, \mathsf{pk}_2^s, st_2), ..., (P_n, \mathsf{pk}_n^v, \mathsf{pk}_n^e, \mathsf{pk}_n^s, st_n)\}$ where r_0 is picked from a uniform distribution. When P_j is activated, it receives B_0 from $\mathcal{F}_{\mathsf{init}}$ and P_j set $sl = sl_1^1$. It also stores $\Pi = (sl_1^1, t_1)$ as a reference point where t_1 is obtained from $\mathcal{F}_{\mathsf{local}}^{P_i,\Sigma}$ when it receives the genesis block.
- If a new P_j is activated after the first slot, it first obtains $\mathsf{pk}_j^v, \mathsf{pk}_j^e, \mathsf{pk}_j^s$ from $\mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}$ by sending a message $(\mathsf{KeyGen}, sid, P_j)$, respectively. Then, it obtains the current slot number sl_{ref}^m from $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ by sending the message $(\mathsf{req\_slot}, sid, P_i)$. Then, it sets $\Pi = (sl_{ref}^m, t_{ref})$ as a reference point where t_{ref} is obtained from $\mathcal{F}_{\mathsf{local}}^{P_i,\Sigma}$. The initial chain C_{best} is $\emptyset$. P_j adds C_{best} to the set $\mathbb{C}_j$.

Phase 2: Chain Extension This phase is the same with Ouroboros Praos [10] except the first bullet.

- (Finding the current slot number) P_j runs $\mathsf{S\_Map}(\Pi, t_{curr})$ and obtains the slot number sl where Π is the reference point and t_{curr} is obtained from $\mathcal{F}_{\mathsf{local}}^{P_i,\Sigma}$.
- (Collecting other chains) P_j receives all chains sent to it from $\mathcal{F}_{\mathsf{DDiffuse}}$. For each chain C, it checks the validity of each block $B = (sl', H, (d, \pi), tx, \sigma)$ of C by following the following steps:
  1. P_j sends the message $(\mathsf{Verify}, sid, (sl', H, d, tx), sl, \sigma, sl', \sigma, \mathsf{pk}_i^v)$ to $\mathcal{F}_{\mathsf{KES}}$. If $\mathcal{F}_{\mathsf{KES}}$ replies with $(\mathsf{Verified}, sid, (sl', H, d, tx, sl', 1)$, then P_j follows the following step. Otherwise, ignores C
  2. P_j sends the message $(\mathsf{Verify}, sid, r_m||sl', d, \pi, \mathsf{pk}_i^v)$ to $\mathcal{F}_{\mathsf{VRF}}$. If $\mathcal{F}_{\mathsf{VRF}}$ replies with the message $(\mathsf{Verified}, sid, r_m||sl', d, \pi, \mathsf{pk}_i^v, 1)$, then P_j follows the next step. Otherwise, it ignores C.
  3. P_j runs the algorithm $\mathsf{Validate}(tx)$ (i.e., check the validity of the transactions). If $\mathsf{Validate}$ outputs valid, P_j validates the block. Otherwise, it ignores C.

  If all blocks of the chain C passes the above steps, P_j adds C to $\mathbb{C}$.
- (Updating the best chain) P_j runs the algorithm $\mathsf{maxvalid}(\mathbb{C}, C_{best}) \rightarrow C'$ and sets $C_{best} = C'$.
- (Checking being a slot leader) P_j sends message $(\mathsf{EvalProve}, sid, P_j, r_m||sl)$ to $\mathcal{F}_{\mathsf{VRF}}$ and receives back $(\mathsf{Evaluated}, sid, P_j, d, \pi)$. If $d < \tau_j$, P_j creates a block for the best chain C_{best} whose last block is B'. The block creation consists of the following steps:
  1. P_j sends (Hash, sid, B') to $\mathcal{F}_{\mathsf{RO}}$ and receives back $H_{B'}$.
  2. Let $m = sl||H_{B'}||d||\pi||tx$. P_j sends $(\mathsf{USign}, sid, P_i, m, sl)$ to $\mathcal{F}_{\mathsf{KES}}$ and receives $(\mathsf{Signature}, sid, m, sl, \sigma)$.
- P_j creates the new block $B = sl||H_{B'}||d||\pi||tx||\sigma$ and appends it to the best chain $C_{best}||B$ and lets $C_{best} = C_{best}||B$. Finally, it sends C_{best} to $\mathcal{F}_{\mathsf{DDiffuse}}$ to be added all other stakeholders' inboxes.

Phase 3: Epoch Formation: P_j runs this phase before starting a new epoch e_m just after sl_R^{m-1} or just after receiving the genesis block R is the epoch length.

If $m \in \{1, 2, 3\}$, then the stake distribution P_j remains the same with the stake distribution defined in the genesis block. Otherwise, P_j retrieves the stake distribution $\mathbb{S}_j$ from the subchain between slot number sl_1^{m-3} to sl_R^{m-3}.

If $m \in \{1, 2\}$, P_j sets $r_m = r_0$. Otherwise, P_j retrieves the VRF values of the blocks which belong to the epoch e_{m-2} in the current best chain. Then, it sends all these values $\rho_m = d_1'||d_2', ..., d_\ell'$ to $\mathcal{F}_{\mathsf{RO}}$ and receives r_m from $\mathcal{F}_{\mathsf{RO}}$.

Figure 4: Ouroboros Clepsydra the hybrid model

Definition 7 ($\Delta\Theta$-Fork)**.** *Let a characteristic string $w \in \{0_s, 0_\ell, 1, \perp\}^n$ and $\Delta, \Theta \in \mathbb{N}$. Δ-fork is a fork $F = (\mathcal{V}, \mathcal{E})$ defined with rules based on w. Let the set $\mathcal{L} = \{i : w_i \in w\} \cup \{0\}$ be a level set that defines levels of F. Each vertex has depth value defined with the function $d : \mathcal{V} \to \mathcal{L}$ where $d(v)$ is the number of edges that is directed from r to v. All levels are mapped to some vertexes with* map*. map has the following properties:*

1. *Level 0 is mapped to the root.*
2. *If $w_i = \perp$, the level i is not mapped to any vertex.*
3. *If $w_i = 0_\ell$ or $w_i = 0_s$, then the level i is mapped to only one vertex.*
4. *For all i, j where $w_i = 0_s$ and $w_j \in \{0_\ell, 0_s\}$, if $i + \Delta \leq j$ then $d(i) \leq d(j)$. This is called as Δ-monotonicity.*
5. *For all i, j where $w_i = 0_\ell, w_j \in \{0_\ell, 0_s\}$, if $i + \Delta + \Theta \leq j$ then $d(i) \leq d(j)$. This is called as $\Delta\Theta$-monotonicity.*

We define also the labelling function $\ell : \mathcal{V} \to \mathbb{L} \cup \{0\}$ where $\ell(v)$ is the level of the vertex i.e., $v \in \mathsf{map}(\ell(v))$. The notation $F \vdash_{\Delta\Theta} w$ represents a $\Delta\Theta$-fork F derived from a characteristic string w.

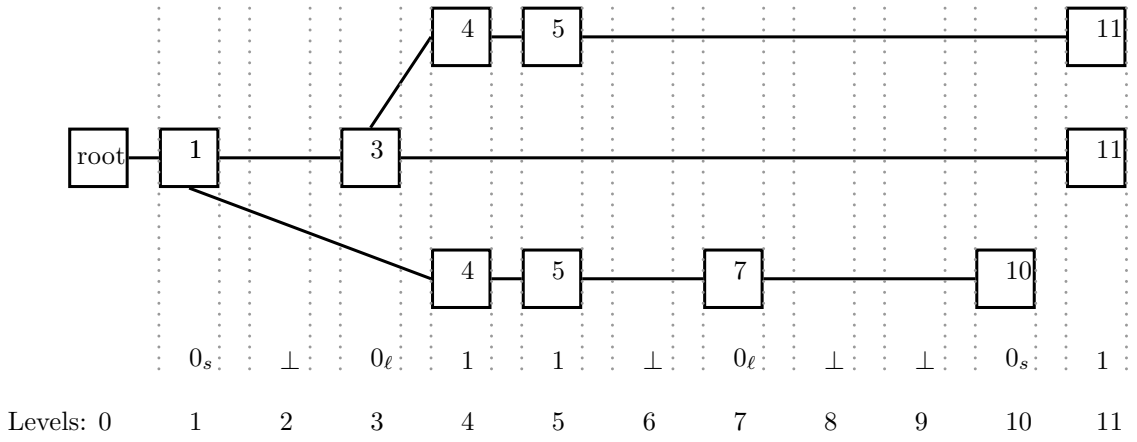

Figure 5: An example $\Delta\Theta$-fork. Each square is vertex and the values in vertexes are their labels. Each level is mapped to the vertexes between two dot-lines e.g, $\mathsf{map}(4)$ is the vertexes labelled with 4, $\mathsf{map}(2)$ is $\emptyset$.

$\Delta\Theta$-fork defines all possible forks during the protocol execution where honest parties follow the maxvalid (See Algorithm 2) chain-selection algorithm and construct only one block on top of one chain during their slots (indexes). For instance, the properties 4 and 5 listed above are always preserved during the execution of Ouroboros Clepsydra because honest parties always construct a block on top of the longest chain. Consider i, j as slots and tines as chains in Ouroboros ClepsydraİIf an honest and synchronized party constructs a block on top of chain C in slot i, then all honest parties receive it at latest at the beginning of slot $i + \Delta$. So, an honest block producer of the slot $j \geq i + \Delta$ already has C. It means that he constructs either on top of C or another chain longer than C. In either case, the depth of new block is greater than the chain of honest and synchronized slot leader at i. The similar argument is valid for

an honest but late slot leader since its chain is received by all honest parties at latest in slot $i + \Delta + \Theta$.

Now, we give the viability and divergence definition from [10] with $\Delta\Theta$-forks.

Definition 8 (Δ-Viability [10])**.** *Let the function* length *measure the length of a tine which is the number of edges in that tine. We call a tine* t *of a* $\Delta\Theta$*-fork* F *is* Δ*-viable if* $\mathsf{length}(t) > \max\{d(h) : h \leq \ell(t) - \Delta, w_i \in \{0_\ell, 0_s\}$ *st.* $\mathsf{map}(i) = h\}$ *where* $\ell(t)$ *is the label of the vertex on tip of* F*.*

Δ-viable tine is a tine which has a possibility to be selected according to $\mathsf{maxvalid}$ algorithm by honest parties.

Definition 9 (Divergence [17])**.** *Consider a* $\Delta\Theta$*-fork* F *retrieved from* $w \in \{0_s, 0_\ell, 1, \perp\}^*$*. The divergence value of* F *is defined as follows:*

$$\mathsf{div}_\Delta(F) = \max_{t_1,t_2}\{\min \mathsf{length}(t_1), \mathsf{length}(t_2) - \mathsf{length}(t_1 \cap t_2)\}$$

where t_1, t_2 *are* Δ*-viable tines of* F*. Here,* $t_1 \cap t_2$ *is the common prefix of* t_1 *and* t_2*. Given* $\mathsf{div}_\Delta(F)$*, we define divergence value of* w *which is equal to the maximum divergence value among all divergence values of a* $\Delta\Theta$*-fork derived from* w*. In other words,*

$$\mathsf{div}_\Delta(w) = \max_{F \vdash_{\Delta\Theta} w} \mathsf{div}_\Delta(F)$$

We would like to have $\mathsf{div}_{\Delta\Theta}(w)$ small enough so that the common prefix property is not violated.

In the following theorem, we restated a result from Ouroboros protocol [17] with $\Delta = \Theta = 0$.

Theorem 1 (Ouroboros Result [17])**.** *Given* $R, k \in \mathbb{N}$ *and* $\epsilon \in (0, 1)$*, let* $w \sim \mathsf{B}(R, \frac{1-\epsilon}{2})$ *(*B *is a binomial distribution). Then* $\mathsf{Pr}[\mathsf{div}_0 \leq k] \leq 1 - \exp(\ln R - \Omega(k))$*.*

Our aim next is to define a mapping from $\{0_s, 0_\ell, 1, \perp\}^n$ to $\{0, 1\}^{n'}$ so that we can reduce a fork F derived from $w \in \{0_s, 0_\ell, 1, \perp\}^n$ to another fork F' derived from $w' \in \{0, 1\}^{n'}$. This reduction helps us to use the result in Theorem 1.

The first map is from an unsynchronized characteristic string to a synchronized characteristic string defined in [10].

Definition 10 (Reduction Mapping $\rho_{\Delta,\Theta}$)**.** *For* $\Delta, \Theta \in \mathbb{N}$*, the function* $\rho_{\Delta,\Theta} : \{0_s, 0_\ell, 1, \perp\}^* \to \{0, 1, \perp\}^*$ *maps an unsynchronized characteristic string to a synchronized characteristic string as follows:*

$$\rho_{\Delta,\Theta}(w) = \begin{cases} 0||\rho_{\Delta,\Theta}(w') & \text{if } w = 0_\ell||w' \text{ and } w' \in \perp^{\Delta+\Theta} ||\{0_s, 0_\ell, 1, \perp\}^* \\ 1||\rho_{\Delta,\Theta}(w') & \text{if } w = 0_\ell||w' \text{ and } w' \notin \perp^{\Delta+\Theta} ||\{0_s, 0_\ell, 1, \perp\}^* \\ x||\rho_{\Delta,\Theta}(w') & \text{if } w = x||w' \text{where } x \in \{0_s, 1, \perp\}. \end{cases}$$

and $\rho_{\Delta,\Theta}(null) = null$ *where null is the end of any characteristic string.*

Remark that this map considers a uniquely honest but late slot sl as malicious if there are not enough number of empty slots (i.e., $\Delta + \Theta$) after sl. The intuition behind it is as follows: A block generated by an honest but late party in sl arrives to the other parties at

latest $\Delta + \Theta$ slot later because the late party sends its block Θ slots later (since it is late) and it may arrive Δ slot later to other honest parties (because of the network delay). So, the slot leaders between sl and $sl + \Delta + \Theta$ continue to produce blocks without seeing the block in sl. This causes that the block in sl may not be in the best chain. So, honest but late party does not contribute the block production as described in Ouroboros Clepsydra if there are not $\Delta + \Theta$ empty slots.

It is clear to see that $\mathsf{div}_\Delta(w) \leq \mathsf{div}_\Delta(\rho_{\Delta,\Theta}(w))$ because $\rho_{\Delta,\Theta}(w)$ has more 1's than w and the $\perp$ values are in the same indexes.

Definition 11 (Reduction Mapping ρ_Δ [10])**.** *For $\Delta \in \mathbb{N}$, the function $\rho_\Delta : \{0,1,\perp\}^* \to \{0,1\}^*$ maps a synchronized characteristic string as follows:*

$$\rho_\Delta(w) = \begin{cases} 1||\rho_\Delta(w') & \text{if } w = 1||w' \\ 1||\rho_\Delta(w') & \text{if } w = 0||w' \text{ and } w' \notin \perp^\Delta ||\{0,1,\perp\}^* \\ 0||\rho_{\Delta,\Theta}(w') & \text{if } w = 0||w' \text{ and } w' \in \perp^\Delta ||\{0,1,\perp\}^* \\ \rho_{\Delta,\Theta}(w') & \text{if } w = \perp ||w'. \end{cases}$$

Lemma 1. *For all $w \in \{0_s, 0_\ell, 1, \perp\}$, $\mathsf{div}_\Delta(w) \leq \mathsf{div}_0(\rho_\Delta(\rho_{\Delta,\Theta}(w)))$.*

Proof. It has been shown in [10] that for all $w' \in \{0,1,\perp\}$, $\mathsf{div}_\Delta(w') \leq \mathsf{div}_0(\rho_\Delta(w'))$. We also know that for all $w \in \{0_s, 0_\ell, 1, \perp\}$, $\mathsf{div}_\Delta(w) \leq \mathsf{div}_\Delta(\rho_{\Delta,\Theta}(w))$. Let us assume that $w' = \rho_{\Delta,\Theta}(w)$, then $\mathsf{div}_\Delta(w) \leq \mathsf{div}_\Delta(\rho_{\Delta,\Theta}(w)) \leq \mathsf{div}_0(\rho_\Delta(w')) = \mathsf{div}_0(\rho_\Delta(\rho_{\Delta,\Theta}(w)))$ □

Definition 12 (Relative Stake)**.** *Given $st = \sum_{P_i \in \mathcal{P}} st_i$ (the total stake) where $\mathcal{P}$ is a set of all stakeholders, a relative stake of a party P_i is $\alpha_i = \frac{st_i}{st}$. The relative stake of honest and synchronized parties is $\alpha_s = \frac{\sum_{P_i \in \mathcal{H}_s} st_i}{st}$ and the relative stake of honest but late parties is $\alpha_\ell = \frac{\sum_{P_i \in \mathcal{H}_\ell} st_i}{st}$.*

Definition 13 (Characteristic String Distribution ($\mathcal{D}_{chr}$))**.** *Given parameters α_s, α_ℓ and c, we define the distribution $\mathcal{D}_{chr}$ on strings $w = w_1 w_2, ..., w_R \in \{0_s, 0_\ell, 1, \perp\}^R$*

$$p_\perp = \mathsf{Pr}[w_i = \perp] = \prod_{i \in \mathcal{P}} 1 - \phi(\alpha_i) = \prod_{i \in \mathcal{P}} (1-c)^{\alpha_i} = 1 - c$$

$$\begin{aligned} p_{0_\ell} = \mathsf{Pr}[w_i = 0_\ell] &= \sum_{i \in \mathcal{H}_\ell} \phi(\alpha_i)(1 - \phi(1-\alpha_i)) \\ &= \sum_{i \in \mathcal{H}_\ell} (1 - (1-c)^{\alpha_i})(1-c)^{1-\alpha_i} \\ &= (1-c) \sum_{i \in \mathcal{H}_\ell} \frac{\phi(\alpha_i)}{1 - \phi(\alpha_i)} \\ &\geq p_\perp \phi(\alpha_\ell) \end{aligned}$$

similarly,

$$p_{0_s} = \mathsf{Pr}[w_i = 0_s] \geq p_\perp \phi(\alpha_s)$$

and

$$p_1 = \mathsf{Pr}[w_i = 1] = 1 - p_\perp - p_{0_\ell} - p_{0_s}$$

Given a characteristic string $w \in \{0_s, 0_\ell, 1, \perp\}$, we want to find to find $\mathsf{Pr}[\mathsf{div}_\Delta(w) \leq k]$. This is equivalent to find $\mathsf{Pr}[\mathsf{div}_0(\rho_\Delta(\rho_{\Delta,\Theta}(w))) \leq k]$ thanks to Lemma 1. Therefore, we analyze the distribution of 0's in $\rho_\Delta(\rho_{\Delta,\Theta}(w))$ where $w \leftarrow \mathcal{D}_{chr}$ in the following lemma.

Lemma 2. *Let $w \leftarrow \mathcal{D}_{chr}$, $z = \rho_\Delta(\rho_{\Delta,\Theta}(w))^{\lceil \Delta+\Theta}$ and $z = ||_{\forall i: w_i \in w \wedge w_i \neq \perp} z_i$. Each $z_i \in z$ is selected from a binomial distribution with the probability*

$$\mathsf{Pr}[z_i = 0] \geq \alpha_s(1-c)^{\Delta+1} + \alpha_\ell(1-c)^{\Delta+\Theta+1}$$

.

Proof. We consider only the distribution of 0's in the trimmed $\rho_\Delta(\rho_{\Delta,\Theta}(w))$ because the last $\Delta + \Theta$ values in $\rho_\Delta(\rho_{\Delta,\Theta}(w))$ do not have the same probability of being 0 with the previous ones i.e., if the last $\Delta + \Theta$ values in w are not $\perp$, then the last $\Delta + \Theta$ values in z are 1.

$$\begin{aligned}
\mathsf{Pr}[z_i = 0] &= \mathsf{Pr}[w_i = 0_s | w_i \neq \perp]\mathsf{Pr}[w_{i+1}||...||w_{i+1+\Delta} = \perp^\Delta] \\
&\quad + \mathsf{Pr}[w_i = 0_\ell | w_i \neq \perp]\mathsf{Pr}[w_{i+1}||...||w_{i+1+\Delta+\Theta} = \perp^{\Delta+\Theta}] \\
&\geq \frac{\phi(\alpha_s)(1-c)^{\Delta+1} + \phi(\alpha_\ell)(1-c)^{\Delta+\Theta+1}}{c} \\
&\geq \alpha_s(1-c)^{\Delta+1} + \alpha_\ell(1-c)^{\Delta+\Theta+1}
\end{aligned}$$

The last inequality comes from the fact that $\phi(\alpha) \geq c\alpha$. □

Now, we prove that CP, CG, CQ properties are preserved in the first two epochs. Remark that the randomness and the stake distribution of the epoch e_1 and e_2 are defined in the genesis block.

Theorem 2 (CP Property)**.** *Let $k, \Delta, \Theta \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $\alpha(1-c)^{\Delta+1}(\gamma + (1-c)^\Theta\beta) \geq (1+\epsilon)/2$ where $\alpha = \alpha_s + \alpha_\ell = \gamma\alpha + \beta\alpha$ is the relative stake of honest parties. The probability of an adversary $\mathcal{A}$ whose relative stake is at most $1-\alpha$ violating the common prefix property with the parameter k in the first or second epoch is $p_{cp} \leq \exp(\ln R + \Omega(k - \Delta - \Theta))$.*

Proof. The proof is very similar to the proof of the CP property in Ouroboros Praos [10]. The proof based on the value of div_Δ because if $\mathsf{div}_\Delta(F) > k$, then the CP property is broken. Therefore, we first determine the probability of having $\mathsf{div}_\Delta(w) > k$ for all $w \leftarrow \mathcal{D}_{chr}$, $|w| = R$. We know the following:

$$\mathsf{div}_\Delta(w) \leq \mathsf{div}_0(\rho_\Delta(\rho_{\Delta,\Theta}(w))) \overset{*}{\leq} \mathsf{div}_0(\rho_\Delta(\rho_{\Delta,\Theta}(w))^{\lceil \Delta+\Theta}) + \Delta + \Theta$$

The inequality * comes from "Lipshitz property" stated in Theorem 4 of [10]. Given that $z = \rho_\Delta(\rho_{\Delta,\Theta}(w))^{\lceil \Delta+\Theta}$ is as described in Lemma 2,

$$\mathsf{Pr}[z_i = 0^{\lceil \Delta+\Theta}] \geq \alpha(1-c)^{\Delta+1}(\gamma + (1-c)^\Theta\beta) \geq (1+\epsilon)/2.$$

Therefore, we can directly apply Theorem 1 and have

$$\mathsf{Pr}[\mathsf{div}_0(\rho_\Delta(\rho_{\Delta,\Theta}(w))^{\lceil \Delta+\Theta}) > k - \Delta - \Theta] \leq \exp(\ln R - \Omega(k - \Delta - \Theta)).$$

As a result of this, for all $F \dashv_{\Delta\Theta} w$,

$$\mathsf{Pr}[\mathsf{div}_\Delta(F) > k] \leq \exp(\ln R - \Omega(k - \Delta - \Theta)).$$

□

Before proving the chain growth, we define two types of special slots: Δ-right isolated and $\Delta+\Theta$-right isolated slots. We basically extend the definition of Δ-right isolated slot given in [10]. We call a slot $\Delta+\Theta$-*right isolated* if the slot leader is one late party and the next $\Delta+\Theta$ slots are empty. We call a slot Δ-*right isolated* if the slot leader is only one synchronized honest party and the next consecutive Δ slots are empty. These slots are very critical in chain growth because honest and synchronized party's block will be in the best chain for sure if there are at least Δ empty slots after. Similarly, a block of honest but late party will be in the best chain if there are at least $\Delta+\Theta$ empty slots after. The reason behind it that an honest block arrives to other honest parties at most Δ slots later after it is released. If this honest block is late then it arrives at most $\Delta+\Theta$ slots later. Therefore, in order to guarantee the chain growth, we need to have more than half Δ-right isolated and $\Delta+\Theta$-right isolated slots so that the best chain includes more honest blocks.

Theorem 3 (Chain Growth)**.** *Let $k, R, \Delta, \Theta \in \mathbb{N}$ and let $\alpha = \alpha_S + \alpha_L = \gamma\alpha + \beta\alpha$ is the total relative stake of honest parties. Then, the probability that an adversary $\mathcal{A}$ violates the chain growth property for all $y > 1$ with parameters $s \geq \frac{2(\Delta+\Theta)y}{y-1}$ and $\tau = \frac{p}{2y}$ in the first epoch or the second epoch is $p_{cg} \leq (\Delta+\Theta+1)(R-s)p\exp(-\frac{(s-3(\Delta+\Theta+1))p}{8(\Delta+\Theta+1)})$ where $p = (1-c)^{\Delta+1}c\alpha(\gamma+(1-c)^{\Theta}\beta)$*

Proof. We follow the similar proof technique given for the chain growth in Ouroboros Praos [10] with different assumptions. We have tighter security bound than the proof in [10].

Consider a chain owned by an honest party in sl_u in epoch e_1 or e_2 and a chain owned by an honest party in $sl_v \geq sl_u + s$. We need to show that there are τs slots between sl_u and sl_v such that the blocks produced in these slots are available to the next slot leaders just before their slot begins. Therefore, we need to find the expected number of $\Delta+\Theta$-right isolated slots between sl_u and sl_v given that the relative stake of late parties is $\alpha_L = \beta\alpha$ and expected number of Δ-right isolated slots given that the relative stake of synchronized honest parties is $\alpha_S = \gamma\alpha$. Remark that a slot can be either $\Delta+\Theta$-right isolated or Δ-right isolated or neither of them.

Consider chains C_u and C_v in slots sl_u and sl_v owned by the honest parties, respectively where sl_u is the first slot of the epoch e_1. We can guarantee that C_u is one of the chains of all parties in $sl_u + \Delta + \Theta$ and the chain C_v is one of the chains of all parties if it is sent in slot $sl_v - \Delta - \Theta$. Therefore, we are interested in slots between $sl_u + \Delta + \Theta$ and $sl_v - \Delta - \Theta$. Let us denote the set of these slots by $S = \{sl_u + \Delta + \Theta, sl_u + \Delta + \Theta + 1, ..., sl_v - \Delta - \Theta\}$. Remark that $|S| = s - 2\Delta - 2\Theta$.

Now, we define a random variable $X_t \in \{0,1\}$ where $t \in S$. $X_t = 1$ if t is $\Delta+\Theta$ or Δ-right isolated with respect to the probabilities $p_\perp, p_{0_\ell}, p_{0_s}$. Remark that X_t's are not independent. Therefore, we divide them into groups where they are independent. Observe that X_t and $X_{t'}$ are independent if $|t - t'| \geq \Delta + \Theta + 1$. We define each group as $S_z = \{t \in S : t \equiv z \bmod \Delta + \Theta + 1\}$. Each element of S_z is a Bernoulli random variable with the probability

$$p = p_{0_s}p_\perp^{\Delta} + p_{0_\ell}p_\perp^{\Delta+\Theta} \geq \alpha_s c(1-c)^{\Delta+1} + \alpha_\ell c(1-c)^{\Delta+\Theta+1} = (1-c)^{\Delta+1}c\alpha(\gamma+(1-c)^{\Theta}\beta)$$

.

The size of each group is $|S_z| = \lfloor \frac{s-2\Delta-2\Theta}{\Delta+\Theta+1} \rfloor > \frac{s-3(\Delta+\Theta+1)}{\Delta+\Theta+1}$. Given this, we apply the Chernoff lower bound to each S_z with $\delta = 1/2$ and $\mu = |S_z|p$.

$$\mathsf{Pr}[\sum_{t\in S_z} X_t < |S_z|p/2] \leq e^{-\frac{|S_z|p}{8}} \leq e^{-\frac{(s-3(\Delta+\Theta+1))p}{8(\Delta+\Theta+1)}}$$

Then, we can conclude that

$$\mathsf{Pr}[X = \sum_{t\in S} X_t \leq \frac{|S|p}{2}] \leq (\Delta+\Theta+1)e^{-\frac{(s-3(\Delta+\Theta+1))p}{8(\Delta+\Theta+1)}} \tag{2}$$

We bound in Equation (2) the probability of having less than or equal to $\frac{(s-2(\Delta+\Theta))p}{2}$ number of $\Delta+\Theta$ and Δ-right isolated slots. Remark that

$$\frac{|S|p}{2} = \frac{(s-2(\Delta+\Theta))p}{2} \geq \frac{(\frac{2(\Delta+\Theta)y}{y-1} - 2(\Delta+\Theta))p}{2} = \frac{p(\Delta+\Theta)}{y-1} = \tau s$$

Let's denote by H the number of $\Delta+\Theta$ and Δ-right isolated slots between sl_u and sl_v.

$$\mathsf{Pr}[H < \tau s = \frac{|S|p}{2}] < \mathsf{Pr}[X < \frac{|S|p}{2}] \leq (\Delta+\Theta+1)\exp(-\frac{(s-3(\Delta+\Theta+1))p}{8(\Delta+\Theta+1)}). \tag{3}$$

We find that in the first s slot of an epoch the chain grows τs block with the probability given in the Equation (3). Now, consider the chain growth from slot sl_{u+1} to sl_{v+1}. We know that the chain grows at least $\tau s - 1$ blocks between sl_{u+1} to sl_v. So, the chain grows one block for sure if sl_{v+1} is Δ or 2Δ-right isolated which with probability p. If we apply the same for each $sl_u < sl < sl_{R-s}$ we obtain

$$(\Delta+\Theta+1)(R-s)p\exp(-\frac{(s-3(\Delta+\Theta+1))p}{8(\Delta+\Theta+1)})$$

□

Theorem 4 (Chain Quality)**.** *Let $k, \Delta, \Theta \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $\alpha(1-c)^{\Delta+1}(\gamma+(1-c)^{\Theta}\beta) \geq (1+\epsilon)/2$ where $\alpha = \alpha_S + \alpha_L = \gamma\alpha + \beta\alpha$ is the relative stake of honest parties. Then, the probability of an adversary $\mathcal{A}$ whose relative stake is at most $1-\alpha$ violates the chain growth property in epoch e_1 or e_2 with parameters k and $\mu = 1/k$ in R slots with probability $p_{cq} \leq Re^{-\Omega(k)}$.*

Proof. The proof is very similar to the proof in [10]. It is based on the fact that the number of $\Delta+\Theta$ and Δ-right isolated slots between slots that corresponds the blocks of any k length portion of the best chain are more than normal slots because of the assumption $\alpha(1-c)^{\Delta+1}(\gamma+(1-c)^{\Delta+1}\beta) \geq (1+\epsilon)/2$. Remark that probability of having $\Delta+\Theta$-right isolated slot given that the slot is not empty is $\alpha\beta(1-c)^{\Delta+\Theta+1}$, having Δ-right isolated slot given that the slot is not empty is $\alpha\gamma(1-c)^{\Delta+1}$ and sum of them are greater than $\frac{1}{2}$ because of the assumption. □

Theorem 5 (CP, CQ, CG in e_3 and e_4)**.** *Fix parameters $k, R, \Delta, \theta \in \mathbb{N}$, $\epsilon \in (0,1)$, $y > 1$, t and q. Let $R \geq 8yk/c(1+\epsilon)$ be the epoch length, and*

$$\alpha(1-c)^{\Delta+1}(\gamma+(1-c)^{\Theta}\beta) \geq (1+\epsilon)/2.$$

The epochs e_3 and e_4 satisfy the CP property with the parameter k, CG property with the parameter $s \geq \frac{2(\Delta+\Theta)y}{y-1}$ and $\tau = \frac{p}{2y}$ and CQ property with the parameters k and $\mu = 1/k$ with probability $1 - 3Rtq(p_{cq} + p_{cp} + p_{cg})$. Here, t is the number of malicious stakeholders and q is the bound in the random oracle model.

Proof. We prove in Theorem 2, 3 and 4 that we have CP, CG and CQ properties in the first two epochs of Ouroboros Clepsydra. The difference of e_1 and e_2 from other epochs comes from the fact that the unbiased randomness beacon and the stake distribution are defined in the genesis block while these values are retrieved from the blockchain in other epochs. Now, we show that given an unbiased randomness beacon in epoch e_1 and e_2 and a static stake distribution in epochs e_1, e_2 and e_3, we have an epoch e_4 which is indistinguishable from e_1 and e_2. It implies that all epochs are indistinguishable from e_1 and e_2.

According to Ouroboros Clepsydra the randomness beacon generated in e_1 is used as a randomness beacon in epoch e_3 and the stake update in e_1 is valid from the beginning of e_4. So, we need to show that the stake update in e_1 becomes static before starting e_4 and the randomness beacon from e_1 is not biased.

We prove the indistinguishably between e_1, e_2 and e_3, e_4 with the following game. The probability of an adversary win a game i is denoted by p_i.

Game 1: In this game, the adversary wants to break one of the security properties CP, CG or CQ in epoch e_3 and e_4 in the $\mathcal{F}_{\mathsf{init}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{DSIG}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{localL}}^{P,\Sigma}, \mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$- hybrid world. Its success probability in this game is p_1.

Game 2: The Game 2 is the same with Game 1 except that the honest chains grows at least $2k$ blocks in every $R \geq 8yk/c(1+\epsilon)$ slots during e_1 and e_2. We know that in Game 1, the chain growth property is violated for e_1 or e_2 at most with probability p_{cg} when $s \geq \frac{2(\Delta+\Theta)y}{y-1}$ and $\tau = \frac{p}{2y}$. As a result of it, we can conclude that in Game 1, the chain grows at least $2k$ blocks in every $R \geq 8yk/c(1+\epsilon)$ slots except with probability p_{cg} as shown below:

$$\tau R = \frac{p}{2y} \frac{8yk}{c(1+\epsilon)} \geq 2k$$

Remark that Game 1 and Game 2 are the same unless the chain growth property is not violated during epochs e_1 or e_2 in Game 1. So, we can conclude that $|p_2 - p_1| \leq p_{cg}$ with using the difference lemma.

We note that in half of the epoch which is $R/2$ slots, the chain of honest parties grows at least k blocks in Game 2.

Game 3: Game 3 is the same with Game 2 except that when a malicious party sends a chain which breaks the common prefix property with the parameter k in e_1 and e_2, the honest parties do not take it into account in the maxvalid algorithm. So, in Game 3, the adversary cannot break the common prefix property in epochs e_1 and e_2. Since, the probability of breaking CP property is at most p_{cp} in Game 2, $|p_3 - p_2| \leq p_{cp} \Rightarrow |p_3 - p_1| \leq p_{cp} + p_{cg}$.

Remark that in Game 3, the valid blocks generated in e_1 will be in the best chain of all honest parties in the middle of epoch e_2 since the chain grows at least k blocks in half epoch. These blocks are finalized meaning that they cannot be removed from the blockchain. This guarantees that all honest parties obtain the same randomness beacon of e_3 before starting e_3.

Game 4: Game 4 is the same with Game 3 except that we have at least one honest block in every k block of the chains of honest parties in e_1 and e_2. Thanks to the chain quality property shown in Theorem 4, we know that there is at least one honest block in the best

chain of every k blocks except with the probability p_{cq}. It means that $|p_4 - p_1| \leq p_{cq} + p_{cp} + p_{cg}$. Clearly, the adversary cannot break the CP, CG, CQ properties of the block chain in e_1 and e_2 in Game 4.

This implies that the randomness beacon of e_4 (resp. e_3) which is generated from the blocks of e_2 (resp. e_1) starts to leak after the last honest block is generated during the second half of an epoch. So, the leakage starts at earliest in slot number $R/2 \geq 4yk/c(1+\epsilon)$ of an epoch e_2 (resp. e_1). Here, leakage means that the adversary starts to learn the randomness beacon candidates at earliest after slot $R/2$ of e_1 and e_2.

Game 5: Game 5 is the same with Game 4 except that $\mathcal{F}_{\mathsf{RO}}$ replies to only slot leaders during the right slot time and only one time in each slot if the input is a valid block between slot numbers $sl^1_{R/2}$ and sl^1_R and between slot numbers$sl^2_{R/2}$ and sl^2_R. In the worst case, in Game 4 the following scenario happens: In the second half of e_1 or e_2, the chain grows only k blocks, the only honest block among the last k blocks is generated in slot $sl^1_{R/2}$ or $sl^2_{R/2}$. It means that the adversary has a chance to decide to produce or not to produce the rest of slots in the corresponding epoch. Therefore, the adversary can obtain at least $Rtq/2$ randomness beacon candidate from $\mathcal{F}_{\mathsf{RO}}$ and select one of them in Game 4. Clearly, the success probability of an adversary in Game 5 is $p_5 = \frac{2p_4}{Rtq}$ because the adversary in Game 5 does not have randomness beacon candidates as in Game 4. Therefore $|p_5 - p_1| \leq \frac{Rtq}{2}(p_{cq} + p_{cp} + p_{cg})$.

Now, we show that the adversary in e_1 and e_2 in Game 1 and the adversary in Game 5 against e_3 has the same success probabilities because the randomness beacon used in e_3 is unbiased and the stake distribution in e_3 is static by genesis. In more detail, in Game 5,

- the randomness beacon of e_3 selected from a uniform distribution as the randomness beacon of the genesis block.
- the randomness beacon of e_3 is generated in epoch e_1 and finalized in the middle of e_2. So, it cannot be changed after.

Since the stake distribution is static in e_3 by the genesis and the randomness beacon is from a uniform distribution in Game 5, we can conclude that e_3 has the same security properties as in e_1 and e_2 in Game 5.

Game 6: Game 6 is the same as Game 5 except that we apply the same reductions to e_3 in Game 1-4. So, $|p_6 - p_1| \leq Rtq(p_{cq} + p_{cp} + p_{cg})$.

Now, in Game 6, we can show that the adversary against e_1, e_2 and e_3 in Game 5 and the adversary in Game 6 against e_4 has the same success probabilities because the randomness beacon used in e_4 is unbiased and the stake distribution update in e_4 becomes static before the randomness beacon leakage. In more detail, in Game 6,

- the bullet points in Game 5 is valid here if we replace e_3 with e_4, e_1 with e_2 and e_3 with e_4.
- the stake distribution update which is done in e_1 is finalized in the middle of epoch e_2 before the leakage of randomness beacon of e_4. Remember that the leakage starts after half of the slots passed. This means that the stake-distribution update and the randomness beacon do not have any correlation. So, the stake distribution becomes static before e_4.

Game 7: Game 7 is the same as in Game 6 except that we apply the same reductions to e_4 in Game 1-4. So, $|p_7 - p_1| \leq \frac{3}{2}Rtq(p_{cq} + p_{cp} + p_{cg})$.

Since the adversary cannot break CP, CQ, CG properties in epoch e_3 and e_4 in Game 7, $p_7 = 0$. So.

$$p_1 \leq \frac{3}{2} Rtq(p_{cq} + p_{cp} + p_{cg})$$

□

Theorem 6 (Security of Ouroboros Clepsydra)**.** *Fix parameters $E, k, R, \Delta, \theta \in \mathbb{N}$, $\epsilon \in (0,1)$, $y > 1$, t and q. Let E is the total number of epochs during the life time of Ouroboros Clepsydra, $R \geq 8yk/c(1+\epsilon)$ be the epoch length, and*

$$\alpha(1-c)^{\Delta}(\gamma + (1-c)^{\Theta}\beta) \geq (1+\epsilon)/2.$$

All epochs of the protocol satisfy the CP property with the parameter k, CG property with the parameter $s \geq \frac{2(\Delta+\Theta)y}{y-1}$ and $\tau = \frac{p}{2y}$ and CQ property with the parameters k and $\mu = 1/k$ with probability $1 - (E-1)Rtq(p_{cq} + p_{cp} + p_{cg} + \mathsf{negl})$ where t is the number of adversaries.

Proof. We apply the same reductions in the proof of Theorem 5 for each epoch e_i and e_{i+1} where $3 < i \leq E-2$ and obtain that e_{i+2} and e_{i+3} satisfies all the security properties. Therefore, the probability that all epochs satisfy the security properties is

$$1 - \frac{(E-1)}{2} Rtq(p_{cq} + p_{cp} + p_{cg})$$

□

5 The Relative Time Protocol

The Relative Time is a protocol that let the stakeholders adjust their local clock. A stakeholder who wants to synchronize its clock does not have to send or receive a specific message to/from other stakeholders. It only needs to watch the arrival time of the blocks.

The relative time protocol works as follows: The newly joined stakeholder P collects the arrival time of n blocks. Whenever it receives a new valid block B'_i, it sends $(\mathsf{Get\_Clock}, sid, P)$ to $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ and obtains the arrival time t_i of B'_i. Let us denote the slot number of B'_i by sl'_i. Here, a valid block means that a block which is valid as described in Ouroboros Clepsydra and which is not equivocated. We note that these blocks do not have to be in a certain order. After collecting n arrival times, P chooses a slot number $sl \geq sl_{max}$ where sl_{max} is the greatest slot number among these n valid blocks. Then, P runs the median algorithm (Algorithm 3). The median algorithm finds candidate arrival times of sl according to each arrival time and then picks the median of them.

Algorithm 3 Median(sl)

1: $lst \leftarrow \emptyset$
2: **for** $i = 0$ to n **do**
3: $\quad a_i \leftarrow sl - sl'_i$
4: $\quad$ **store** $(t_i + a_i T)$ to lst
5: $lst \leftarrow \mathtt{sort}(lst)$
6: **return** $\mathtt{median}(lst)$

Assume that t_{sl} is the output of the median algorithm. Then, P considers t_{sl} is the start of the slot sl and find the current slot number with the algorithm $\mathsf{S\_Map}((sl, t_{sl}), t_{curr})$ (Algorithm 4) where t_{curr} is the current time.

Theorem 7. *Assuming that $\alpha\gamma \geq (1+\epsilon)/2$ where $\alpha\gamma$ is the relative stake of honest and synchronized parties, the relative time protocol in $\mathcal{F}_{\mathsf{DDiffuse}}, \mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}, \mathcal{F}_{\mathsf{local}}^{T,\Sigma}$ and $\mathcal{F}_{\mathsf{RO}}$ hybrid models realizes $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ where $\Theta = \Delta + 1$ and $|\Sigma| < T$.*

Proof. In order to prove the theorem, we construct a simulator $\mathcal{S}$ where $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{DDiffuse}}, \mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}, \mathcal{F}_{\mathsf{local}}^{T,\Sigma}$ and $\mathcal{F}_{\mathsf{RO}}$.

When a party P contacts with $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ with the message (Sync, sid, P), $\mathcal{G}_{\mathsf{Slot}}^{T,\Theta}$ relays it to $\mathcal{S}$. P in the real protocol supposed to run the Relative time protocol so $\mathcal{S}$ simulates P in the real protocol by running the Relative time protocol. The simulation is straightforward. $\mathcal{S}$ sends $(\mathsf{Register}, sid, P)$ to $\mathcal{G}_{\mathsf{refRate}}$ in order to emulate $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$ and behave as same as $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$. $\mathcal{S}$ simulates the honest parties in Ouroboros Clepsydra as well. For each active party P_j, it retrieves T_i from $\mathsf{localclock}_{\mathcal{Z}}$ of $\mathcal{F}_{\mathsf{local}}^{\Sigma,P_j}$. Then, it learns the slot $sl = \mathsf{S\_Map}(\Pi_j, T_i)$ for P_j and checks if it is the slot leader. If P_j is the leader, it produces the block according to Ouroboros Clepsydra. $\mathcal{S}$ sends the block of P_j to $\mathcal{A}$ (since $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{DDiffuse}}$). If $\mathcal{A}$ moves the block to the inbox of P, $\mathcal{S}$ stores the time that the block moved to the inbox of P as a arrival time of this block. If the block is delayed by $\mathcal{A}$, $\mathcal{S}$ waits until $\mathcal{A}$ permits the block move to the inbox of P. If the permission is not received Δ slots later, $\mathcal{S}$ moves the block to the inbox of P. In either case, it stores the arrival time of the block in the inbox of P from $\mathsf{localclock}_{\mathcal{Z}}$ of $\mathcal{F}_{\mathsf{local}}^{\Sigma,P}$. Remark that $\mathcal{S}$ know the duration of Δ-slot because it receives the exact rate from $\mathcal{G}_{\mathsf{refRate}}$ while simulating the local clocks.

After collecting n arrival times, $\mathcal{S}$ runs the median algorithm with a slot number sl. It adjusts $\mathsf{S\_Map}$ by referencing that sl starts at the output of the median algorithm t. Then, $\mathcal{S}$ computes the current slot number $sl' - 1 = \mathsf{S\_Map}((sl, t), t_{curr})$ where t_{curr} is the current time according to $\mathcal{F}_{\mathsf{local}}^{P,\Sigma}$. When the local time reaches the start time of sl', it sends sl' to $\mathcal{F}_{\mathsf{Slot}}$ and outputs sl' as an output of P in the real protocol. In the ideal functionality P does not output sl' if $sl_{curr} - sl' > \Theta$ or $sl_{curr} < sl'$. Now, we show that one of these happens with the probability $\exp(-\frac{\delta^2\mu}{2})$ where $\frac{\epsilon}{1+\epsilon} \leq \delta \leq 1$ and $\mu = n(1+\epsilon)/2$.

We first show that more than half of n arrival times are computed based on honest and synchronized parties except with the probability $\exp(\frac{\delta^2\mu}{2})$. We define a random variable $X_v \in \{0, 1\}$ which is 1 if t_v is the arrival time of an honest and synchronized block. Then the expected number of honest and synchronized blocks among n blocks is $\mu = n(1+\epsilon)/2$ because $\alpha\gamma \geq (1+\epsilon)/2$. We bound this with the Chernoff bound:

$$\mathsf{Pr}[\sum_{v=1}^{n} X_v \leq \mu(1-\delta)] \leq \exp(-\frac{\delta^2\mu}{2})$$

Given that $\frac{\epsilon}{1+\epsilon} \leq \delta \leq 1$, $\mu(1-\delta) \leq n/2$. So, the probability of having more than half honest and synchronized block out of n blocks is $1 - \exp(-\frac{\delta^2\mu}{2})$.

Now, we do the analysis of the algorithm assuming that more than half of the blocks among n blocks are sent by honest and synchronized parties. Let us assume that the median is $t = t_i + a_i T$. If a block of sl'_i is sent by an honest and synchronized party, the $t'_i \leq t_i \leq t'_i + \Delta T + \Sigma$ where t'_i is the time that honest and synchronized party sends the block of sl'_i.

All these times are with respect to the local clock of P. Given this, the correct slot number number at time t is sl_{corr} which is bounded as follows:

$$\begin{aligned} sl_{corr} &= sl'_i + \lceil \frac{t - t'_i}{T} \rfloor \\ &= sl'_i + \lceil \frac{t_i + a_i T - t'_i}{T} \rfloor \\ &\leq sl'_i + \lceil \frac{t_i + a_i T - t_i - \Delta T - \Sigma}{T} \rfloor \\ &= sl + \Delta + \lceil \frac{-\Sigma}{T} \rfloor \\ &\leq sl + \Delta + 1 \end{aligned} \tag{4}$$

Remark that this inequality holds for all slot numbers that are computed with using one of the honest and synchronized parties' blocks. This shows that more than half of the time values in sorted lst is between the time that corresponds $sl_{corr} - \Delta - 1$ and sl_{corr} because more than half of the blocks among n blocks are sent by honest and synchronized parties. Therefore, if the median $t = t_i + a_i T$ corresponds to time derived from a malicious parties' block, this time should be between $sl_{corr} - \Delta - 1$ and sl_{corr}. □

6 Conclusion

In this paper, we designed Ouroboros Clepsydra which is a modified version of Ouroboros Praos [10]. The stakeholders Ouroboros Clepsydra can learn the slot number when they lose this information. However, the obtained slot time may not be the correct slot number, but we assumed that it can be at most Θ slot behind the real one. Based on this assumption, we proved that Ouroboros Clepsydra satisfies the CP, CG and CQ properties.

We also constructed a new GUC model to capture the notion of relative time. In this model, we constructed a general functionality which informs other functionalities whenever their clock should progress. This functionality is realizing the rate of clocks in real life. Besides, we constructed a new functionality for local clocks which may not follow the real clock rate. This functionality realizes the local clocks in real life which may be drifted.

In the end, we proposed an algorithm that realizes Θ-slot behind assumptions. We proved that this algorithm is GUC-secure in our relative time model with $\Theta = \Delta + 1$ and the maximum drift is equal to at most T which is the slot time. Remark that if there is not any drift in the local clock during the relative time protocol, $\Theta = \Delta$. Our protocol does not depend on any central clock and it can be adapted in any proof-of-stake blockchain algorithm similar to Ouroboros Clepsydra.

References

[1] Proof of authority. `.https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains`.

[2] G. Ateniese, I. Bonacina, A. Faonio, and N. Galesi. Proofs of space: When space is of the essence. In *International Conference on Security and Cryptography for Networks*, pages 538–557. Springer, 2014.

[3] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930. ACM, 2018.

[4] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `https://eprint.iacr.org/2000/067`.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 136–145. IEEE, 2001.

[6] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *Theory of Cryptography Conference*, pages 61–85. Springer, 2007.

[7] R. Canetti, K. Hogan, A. Malhotra, and M. Varia. A universally composable treatment of network time. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 360–375. IEEE, 2017.

[8] J. Chen and S. Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[9] P. Daian, R. Pass, and E. Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake. *Cryptology ePrint Archive*, 2017.

[10] B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[11] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[12] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In *Annual Cryptology Conference*, pages 585–605. Springer, 2015.

[13] B. Fisch. Tight proofs of space and replication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 324–348. Springer, 2019.

[14] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[15] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[16] T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[17] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[18] R. Pass and E. Shi. The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 380–409. Springer, 2017.

A An Instantiating of Ouroboros Clepsydra

Ouroboros Clepsydra consists of sequential non-overlapping epochs $(e_1, e_2, \ldots)$, each of which consists of a number of sequential block production slots $(e_i = \{sl_1^i, sl_2^i, \ldots, sl_R^i\})$ up to some bound R. At the beginning of an epoch, stakeholders are assigned to a slot as a *slot leader*, often one party or no party, but sometimes more than one party. These assignments are initially secrets known only to the assigned slot leader themselves, but eventually they publicly claim their slots when they produce a new block in one.

Each stakeholder P_j has a key containing two types of secret/public key pair:

- a verifiable random function (VRF) key $(\mathsf{sk}_j^v, \mathsf{pk}_j^v)$, and
- a signing key for blocks $(\mathsf{sk}_j^s, \mathsf{pk}_j^s)$

Each stakeholder P_j keeps a local set of blockchains $\mathbb{C}_j = \{C_1, C_2, ..., C_l\}$. All these chains have some common blocks, at least the genesis block, up until some height. We assume that each stakeholder has a local transaction list that contains the transactions to be added to blocks. All transactions in a block is validated with a transaction validation function. A transaction is a basically signature signed by a signing key.

Before giving the details, we define the probability of being selected as a slot leader:

$$p_i = \phi_c(\alpha_i) = 1 - (1 - c)^{\alpha_i}$$

where α_i is the relative stake of the stakeholder P_i and c is a constant. So, each stakeholder is selected proportional to its stake. Remark that the function ϕ is that it has the "independent aggregation" property as remarked in [10], which informally means the probability of being selected as a slot leader does not increase as a stakeholder splits his stakes across virtual stakeholders.

ϕ is used to set a threshold τ_i for each stakeholder P_i:

$$\tau_i = 2^{\ell_{vrf}} \phi_c(\alpha_i) \tag{5}$$

where ℓ_{vrf} is the length of the VRF's first output.

There are three phases in Ouroboros Clepsydra.

Phase 1: Initialization Phase This phase differs according to being at the beginning of the first epoch and the other epochs.

In the beginning of the first epoch, the genesis block released. As soon as a stakeholder receives it, it marks the arrival time of the genesis block as the start of sl_1^1 (the first slot of the first epoch).

The genesis block contains two random number (r_1, r_2) for use during the first and the second epoch for slot leader assignments, the initial stake's of stake holders $(st_1, st_2, ..., st_n)$ valid in first three epochs (e_1, e_2, e_3) and their corresponding session public keys $(\mathsf{pk}_1^v, \mathsf{pk}_2^v, ..., \mathsf{pk}_n^v)), ((\mathsf{pk}_1^s, \mathsf{pk}_2^s, ..., \mathsf{pk}_n^s)$.

After the beginning of the first epoch, the initialization phase corresponds to a phase for a newly joining party to obtain the up to date chain (best chain) and synchronizing the slot time with its local time. A newly joining stakeholder P_j first signs its stake st_j with its secret key sk^s and sends it to other stakeholders as a transaction. Next, P_j obtains the current valid blockchain from an authority that provides the current best chain[2]. In addition, P_j runs the algorithm $\mathsf{Get\_Time}(sl) \rightarrow \tau$ in order to learn its local time that corresponds sl and synchronize its local clock according to τ (i.e., if sl starts at τ, $sl + t$ starts at $\tau + tT$). We give an example $\mathsf{Get\_Time}$ algorithm in Section 5.

We note that instead of using a trusted authority to obtain the best chain, we could also use $\mathsf{maxvalid\text{-}mc}$ algorithm from Ouroboros Genesis [3]. However, this may complicate the security proof. Therefore, we decide to use a trusted authority which is the case in the real life.

Phase 2: Chain Extension In this phase, each slot leader should produce and publish a block. All other stakeholder attempt to update their chain by extending with new valid blocks they observe. This phase is identical with Ouroboros Praos [10].

We assume each stakeholder P_j has a set of chains $\mathbb{C}_j$ in the current slot sl_k^m of the epoch e_m. Each has a best chain C selected in sl_{k-1}^m by $\mathsf{maxvalid}$ algorithm (Algorithm 2), and the length of C is ℓ-1. We call that P_j is a slot leader of the slot sl_k^m if the first output (d) of the VRF evaluated as below is less than the threshold τ_j (See Equation 5).

$$\mathsf{VRF}_{\mathsf{sk}_j^v}(r_m||sl_k^m) \rightarrow (d, \pi)$$

where r_m is the epoch randomness. If P_j is a slot leader of slot sl_k^m then it produces a block of sl_k^m. Remark that the more P_j has stake, the more he has a chance to be selected as a slot leader.

If P_j is the slot leader of sk_k^m, it generates a block to be added on C. The block B_ℓ should contain the slot number sl_k^m, the hash of the previous block $H_{\ell\text{-}1}$, the VRF output d, π, transactions tx, and the signature $\sigma = \mathsf{Sign}_{\mathsf{sk}_j^s}(sl_k^m||H_{\ell\text{-}1}||d||pi||tx))$. P_i add the new block on top of C and sends B_ℓ.

In any case (being a slot leader or not being a slot leader), when P_j receives a block $B = (sl, H, d', \pi', tx', \sigma')$ produced by any stakeholder P_t, it validates the block with $\mathsf{Validate}(B)$. $\mathsf{Validate}(B)$ should check the followings in order to validate the block:

- if the signature is valid (i.e., if $\mathsf{Verify}_{\mathsf{pk}_t^s}(\sigma') \rightarrow \mathsf{valid}$),
- if the stakeholder is the slot leader (i.e, $\mathsf{Verify}_{\mathsf{pk}_t^v}(\pi', r_m||sl) \rightarrow \mathsf{valid}$ and $d' < \tau_t$),
- if P_t did not produce another block for another chain in slot sl (no equivocation),
- if there exists a chain C' with the header H.

If a block passes all the validation steps, P_j adds B to C'. Otherwise, it ignores the block.

At the end of the slot, P_j decides the best chain with the algorithm $\mathsf{maxvalid}(\mathbb{C}, C, C')$ where C is the the best chain selected by $\mathsf{maxvalid}$ (Algorithm 2) in the last run and C' is the new chain generated in sl_k^m.

[2]These authorities exist in real life. They are trusted because they have a good incentive (e.g. earn money) to act honestly. Remark that their malicious action can be easily proven and they directly lose their reputation

Phase 3: Epoch Formation Stake distribution and the randomness beacon form an epoch.

Before starting a new epoch e_m, a stakeholder P_j has to complete updating its stake (if it needs). For this, it should sign its stake with the current epoch number e_m and publish it as a transaction. The new stake update is going to be valid in epoch e_{m+3}. Therefore, the new stake distribution of an epoch e_{m+3} is retrieved from e_m.

The randomness for an epoch e_{m+2} is computed as follows: Concatenate all the VRF outputs of blocks in the current epoch e_m (let us assume the concatenation is ρ). The randomness in epoch e_{m+1}:

$$r_{m+1} = H(r_m||m+2||\rho)$$

Therefore, a randomness beacon of epoch e_{m+2} is retrieved from an epoch e_m.