Consensus on Clock in Universally Composable Timing Model

Clock Syncronization Protocol for Full Nodes of a Blockchain

Handan Kılınç Alper

Web3 Foundation

handan@web3.foundation

Abstract

In blockchain space, there are elaborate proof-of-stake based protocols with some assumptions related to clock synchronization, i.e. that all of them know the current time of the protocol. However, this assumption is satisfied by relying on the security of centralized systems such as Network Time Protocol (NTP) or Global Positioning System (GPS). Clearly, any attack on these systems (which happened in the past) can cause corruption of blockchains that rely on the clock that they provide. To solve this problem in the nature of the decentralized network, we first define a general universally composable (GUC) model that captures the notion of consensus on a clock. Simply, a consensus clock is a clock that is agreed upon by honest parties by considering the clocks of all parties. In the end, we give a simple but useful protocol relying on a blockchain network. Our protocol is secure according to our new model. It can be used by full nodes of a blockchain who need to have a common time notion to preserve the correctness and the security of the blockchain protocol. One advantage of our protocol is that it does not cause any extra communication overhead on the underlying blockchain protocol.

1 Introduction

The first popular decentralized cryptocurrency, Bitcoin, maintains its public distributed ledger with a proof-of-work (PoW) based consensus protocol. PoW has been studied for many decades and their impact in the development of blockchain technologies is undeniable. However, it is a well-known fact that this protocol consumes vast amounts of energy. Consequently, there are ongoing efforts in the community to replace PoW with more energy-efficient alternatives, that preserve its decentralized features. *Proof of stake* (PoS) is arguably the most promising replacement among the many recently proposed solutions [16, 5, 18, 1]. It has the same nature of PoW: while in the latter the next block producer is selected from a set of nodes at random proportional to their computational power, in PoS, they are randomly selected proportional to their stake.

One of the most critical issues in PoS is constructing a selection mechanism which cannot be biased by an adversary. Many solutions to the problem of unbiased random selection have been proposed, based on random oracles [34, 14], publicly verifiable secret sharing (PVSS) schemes [26], verifiable random functions (VRF) [15, 13, 21, 22], and threshold cryptography [22]. Using any such selection mechanism, each party must decide if she is eligible to produce the next block within a specific time interval. However, a second critical issue, which has not

yet been treated satisfactorily, is how this party detects this time interval correctly. Indeed, in order to preserve security, it is paramount for an honest party to release her produced block at the right time. If she is late, then the next producers continue to build on the chain without seeing her block; as a result, the honest party may not have any chance to contribute to the chain growth. If she is early, a similar problem arises.

One may ask why not parties of a blockchain use their computer clock to deduce their time interval. It is a known fact that a party cannot rely on only the computer clock because computer clocks are controlled by the vibration of a crystal oscillator which are not very accurate (e.g. few seconds drift in one day). Therefore, these parties usually update their computer clocks with the Network Time Protocol (NTP) which is the main clock synchronization protocol in internet for a long time. In past, we had cases where NTP servers were attacked [30, 36, 32]. In one incident (on November 19, 2012) [8], two of the critical servers of NTP went back to 12 years ago. This caused many important external servers to blackout such as Active Directory authentication servers. If the same attack happens in a proof-of-stake blockchain protocol, honest parties stop producing blocks because they think that their time did not come while malicious full nodes produce blocks and dominate the blockchain. Another option is to use Global Posioning System (GPS) clock with a little more investment which is more accurate than NTP and does not have the problem of corrupted servers. However, GPS is vulnerable to spoofing attacks [39, 25, 33, 24, 40] (e.g., delay signals). Even if there is no attack sometimes bad weather conditions can cause inaccurate signals sent by the GPS satellite. All of these existing solutions show that relying on centralized system is a major security vulnerability and it goes against the ultimate goal of building a fully decentralized system.

Sharing timestamps among parties of a blockchain is an appropriate solution, but (without an additional mechanism like the one we propose) it is not clear how to use these timestamps in a system with malicious actors. For example, consider a method that says synchronize the clock based on the timestamp in the last block. This is not a secure solution because first not all parties have the same block as a last block and second it gives power to a malicious party to change the clocks of honest parties as he wants. So, solutions that gives some power to the adversary to change the clocks of honest parties is not an ideal solution. Beyond this, it is also important not to shift the clock of honest parties dramatically not to stop the block production for a while. Considering these issues, we propose a new synchronization protocol on top of a blockchain protocol that interprets timestamps (clock values) in blocks and arrival times of blocks to obtain a consensus clock between parties.

The other issue we encounter was proving the security of such synchronization mechanism. For this, we decide to prove the security of our protocol in a universally composable (UC) model because we want to make sure that our protocol can be composed on top of blockchain protocols securely. The closest UC model for our purposes is the GUC network time model designed by Canetti et al. [12] which models the clock adjustment of a client with servers of which can be corrupted. In their model, clients want to synchronize their clock close to the reference clock. On the other hand, when we consider the synchronization issue in a blockchain protocol, we see that we do not actually need any notion of reference clock. In simple terms, the functionality we want to have is to update clocks of honest parties such that they are close enough to each other and they do not increase or decrease dramatically for the security of the blockchain. For our purposes, we do not need to consider how close these clocks to a reference clock. Therefore, we design a new general universally composable (GUC) consensus clock. In our model, the parties has local timers which let them to construct

their own clock based on a consensus clock. We note that the notion of consensus on clock is not a brand new idea. Indeed, it has been studied in other areas outside of cryptography [29, 35, 23, 3, 41, 37, 28]. However, there is not any security model capturing this notion except ours. We note that our model is generic and can be used for the security of any synchronization protocol without a reference clock.

Our motivation for this kind of security model without a reference clock comes from the blockchain consensus mechanisms. In such distributed and decentralized consensus protocols, accepting one static clock (e.g., reference clock in [12]) as a right clock goes against the grain of consensus. Imagine a blockchain protocol where each party has a synchronized clock at the beginning. As time progresses, the synchronization may be broken because of random drift on clocks. At some point, parties may end up with a situation where no relation exists between clocks of parties. In this case, the question is the parties should synchronize their clock with respect to which clock. We have at least two options for a reference clock:

- An authority defines the reference clock to be synchronized similar to the GUC model Canetti et al. [12].
- Parties agree on the reference clock to be synchronized

We believe most people agree on the second option is more appropriate if we want to construct completely distributed system without trust on a single entity. Therefore, we construct a new GUC clock model which avoids any central clock deciding what is the reference clock to be synchronized.

1.1 Our Contribution

In more detail, our contribution is as follows:

- We construct a GUC model that captures the notion of consensus on clocks. Our model realizes situations where a party has a local timer with a defined progress rate but wants to obtain a consensus clock based on the progress in the local timer (e.g., a situation where a party joins a blockchain network with a local timer and does not know what is the right clock, and wants to create a new clock showing the current clock value). We define a global functionality which defines the rate of timers globally and another functionality of a local timer which does not necessarily follow the global rate to capture the notion of a drifted timers in the real world. Our other functionality provides the consensus clock to honest parties. According to our definition, the consensus clock can change as the time progresses based on the clocks of parties but this change is limited. To the best of our knowledge, our GUC model is the first model which models the notion of consensus on clock. We note that we do not design our model specific to blockchain protocols. It can be used by any protocol who aim to achieve consensus on clocks.
- We construct a generic protocol called *Relative Time* protocol that realizes the our consensus functionality. Our protocol makes sure that honest clocks are close enough the consensus clock. Even if a local timer of a party drifts or network delay exists, our protocol achieve consensus on clocks. Our protocol can be adapted to all blockchain protocols that fit our abstraction. Thus, we solve the synchronization problem in PoS-based blockchain protocols where an eligible block producer creates a block in a certain time intervals.
- Our protocol can be used by parties who are actively involved the blockchain protocol as well as a group of parties who do not have any contribution on the blockchain protocol (not a full node) but simply want to have a synchronization in the group. Therefore, our

protocol can be used to solve the synchronization problem of other protocols different than blockchain.

We note that our relative time protocol is currently used in Polkadot [42] which aims to connect multiple blockchain networks.

1.2 Related Works

UC Timing Model: The closest timing model to ours is called network time model by Canetti et al. [12]. Their model has a reference clock (a global clock) that keeps a counter incremented by the adversary. The network time model captures the physical clocks where a party can access immediately but that may be arbitrarily shifted and which can be updated by asking other parties' clock. The security of the clock is defined based the closeness of the reference clock. So, the ultimate goal in this model to obtain clocks close enough to the reference clock. Differently, in our model, the ultimate goal of parties is to have clocks that are close to each other. It does not important how close the consensus clock to a reference clock (e.g. a clock defined by NTP, GPS). This type of clock update is useful for protocols which do not need to rely on any information of a real world clock for security and completeness. In a nutshell, our model is a version of network time model where the reference clock is defined based on the consensus on a clock.

PoS protocols: In Ouroboros [26] and Ouroboros Praos [15] protocols, there is a common timeline that is divided into *slots*, where each slot may or may not have a block producer. Their security is based on the assumption that all parties know when each slot starts and ends. The Ouroboros Genesis [6] protocol is similar to Ouroboros Praos, and it explicitly mentions that the violation of this assumption breaks the security of the chain selection rule. The timeline in the Dfinity consensus protocol [22] progresses based on certain events, but it is not clear how everyone can agree on the time at which any event occurs in a partially synchronous network. On the other hand, the Algorand protocol [13, 21] executes Byzantine agreement on each block, hence 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 [26], Ouroboros Praos [15] or Ouroboros Genesis [6] because a party cannot know whether a block is produced by an honest party, or whether it is sent during the correct time slot. Snow White [14, 34] is the only protocol where this timing issue is considered in their analysis, and the authors propose 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.

Recently, a new protocol called Ouroboros Chronos [7] is proposed to solve the timing issues in Ouroboros Genesis [6] that we mention. Their adversarial model is different than ours since they assume that there exists a core set of parties (alert) who are honest and synchronized. The synchronization algorithm that they propose helps new joining parties to synchronize themselves with the alert parties. The existence of such core set of parties is based on the assumption that their clocks follow almost the same rate. However, we know that nowadays computer clocks do not have such accurate rate. Our model and their model differ in this point because we assume that the clock of parties can arbitrarily shift considering the nature of real world computer clocks based on crystal oscillators. Therefore, synchronization between parties cannot be maintained for a long time in our model unless there is a mechanism to preserve it as we propose. In addition to have a different adversarial model, our protocol is generic which can be applied any blockchain protocol compatible with our abstraction (e.g. Ouroboros Praos, Genesis [15, 6]). Ouroboros Clepsydra [4] has the similar adversarial model

as Chronos.

Synchronization Protocols: Independently from blockchain, synchronization protocols [17, 19, 31, 38, 27] are deeply studied in previous years since it is a important component in distributed systems. Many protocols [29, 35, 23, 3] exist for consensus clock with different assumptions. One approach based on dividing network into some well connected clusters [17] that aim to achieve consensus between clusters. There are also a fully distributed approaches [41, 37] based on clock skew and network delay estimation. Even though many works exist related to consensus clock, there exists no formal cryptographic security model as we defined for these type of functionality to best of our knowledge. Different from previous works, our protocol works on top of a secure blockchain protocol with a consensus mechanism which lets parties also reach consensus on clocks. Our advantage comparing the previous protocols is that building the synchronization protocol on top of an existing consensus mechanism.

2 Preliminaries

Notations: 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 Blockchain

A protocol that defines the construction of a blockchain is called a blockchain protocol. Garay et al. [20] 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, $C_1, C_2, ...$ represent round (clock value). The formal definition of the notion of C is in Section 3.1.

Definition 2.1 (Common Prefix (CP) Property [20])**.** *The common prefix property with parameters $k \in \mathbb{N}$ ensures that any blockchains B_1, B_2 owned by two honest parties at the onset of rounds $C_1 < C_2$ satisfy that $B_1^{\lceil k}$ is the prefix of B_2 where $B_1^{\lceil k}$ is B_1 without last k blocks.*

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 2.2 (Chain Growth (CG) Property [20])**.** *The chain growth property with parameters $\tau \in (0, 1]$ and $s_{cg} \in \mathbb{N}$ ensures that if the length of a blockchain owned by an honest party at the onset of a round C_u is ℓ_u and the length of the same blockchain at least s_{cg} round before is ℓ_v, then the $\ell_u - \ell_v \geq \tau s_{cg}$.*

In other words, the CG property guarantees a minimum growth after s_{cg} rounds later on an honest chain. τ defines how fast the blockchain grows.

Definition 2.3 (Chain Quality (CQ) Property [20])**.** *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 a blockchain owned by an honest party is at least μ.*

The CQ property ensures the existence of sufficient honest blocks on any blockchain owned by an honest party.

We also define a new property which is a necessary property in our protocol.

Definition 2.4 (Chain Density (CD) Property)**.** *The chain density property with parameters $s_{cd} \in \mathbb{N}$ ensures that if a blockchain owned by an honest party at the onset of a round C_u is B then any portion of B spanning s_{cd} prior rounds with n blocks contains number of n_h honest blocks where $\frac{n_h}{n} > \frac{1}{2}$.*

Chain density shows that a blockchain adopted by an honest party contains more honest blocks than the malicious ones in a sufficiently long proportion of that blockchain. A blockchain constructed based on the longest chain rule implies the chain density property if it satisfies chain growth and chain quality [26].

2.2 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 [9, 10] for further details about the UC-model.

Definition 2.5. *(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 $\mathcal{Z}$,*

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

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

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

- (Registration) If a functionality $\mathcal{F}$ sends a message $(\mathsf{Register}, sid, \mathcal{F})$, $\mathcal{G}_{\mathsf{refRate}}$ adds $\mathcal{F}$ to $\mathsf{ListRate}$ and sends $(\mathsf{Registered}, sid, \mathcal{F})$ to P.
- (Progress in Time) As the time progresses according to the defined rate, $\mathcal{G}_{\mathsf{refRate}}$ sends $(\mathsf{Increase}, sid', P)$ to all $P \in \mathsf{ListRate}$

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

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.

Generalized UC model [11] (GUC) formalizes the global setup in a UC-model. In GUC model, $\mathcal{Z}$ can interact with arbitrary protocols and ideal functionalities $\mathcal{F}$ can interact with GUC functionalities $\mathcal{G}$.

3 UC-Model for Relative Time

In this section, we describe our new GUC-model for consensus clocks and the partial synchronous network for blockchain protocols.

In our model we have multiple ITMs that we call each as a party P_i. $\mathcal{Z}$ activates all parties and the adversary and creates ITIs. $\mathcal{A}$ is also activated whenever the ideal functionalities invoke. $\mathcal{A}$ can corrupt parties $P_1, P_2, ..., P_n$. 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 $\mathcal{A}$.

We first introduce our new GUC-model for consensus clocks notion and then give the UC-model for partial synchronous network similar to the model in [15, 6].

3.1 GUC-Model of Relative Time

A similar model to ours "GUC network time model" is introduced by Canetti et al. [12] which models the clock adjustment of a client with servers of which can be corrupted. In their model, clients want to adjust their clock close to the reference clock. In our model, we do not have necessarily a reference clock that progresses depending on inputs of $\mathcal{Z}$. Instead, the reference clock is determined based on the clock of of active parties. We use the notion of a *clock value* that is defined for a specific time interval. A clock value is a metric time in the metric system in the real world. In a nutshell, we want to define a GUC-model where each party immediately accesses to its local timer which may not progress as the same rate of other parties' timers. The rate of all local timers is defined in a global setup but the local timers may not follow the same rate while they progress. In our model, the only absolute value is clock value and the reference clock is determined with a consensus mechanism.

Our model consists of the following functionalities:

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

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

- (Registration) $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ sends $(\mathsf{Register}, sid, \mathcal{F}_{\mathsf{timer}}^{\Sigma,P})$ and receives back $(\mathsf{Registered}, sid, \mathcal{F}_{\mathsf{timer}}^{\Sigma,P})$. This is done once after the initialization of the functionality.
- (Increasing timer) When $\mathcal{Z}$ sends $(\mathsf{Increase}, sid, P)$, $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ increments timer and lets $\Sigma = \mathsf{timer} - \mathsf{timer}^*$.
- (Increasing timer^*) When $\mathcal{G}_{\mathsf{refRate}}$ sends $(\mathsf{Continue}, sid, \mathcal{F}_{\mathsf{timer}}^{\Sigma,P})$, $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ increments timer^* and lets $\Sigma = \mathsf{timer} - \mathsf{timer}^*$.
- If P sends $(\mathsf{Get\_Timer}, sid, P)$, $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ sends the message $(\mathsf{Timer}, sid, \mathsf{timer}, P)$.

Figure 2: The global functionality $\mathcal{F}_{local}^{\Sigma,P}$

3.1.1 Reference Rate ($\mathcal{G}_{\mathsf{refRate}}$)

This functionality defines a global clock rate. The functionalities who want to be notified in every progress of time register to $\mathcal{G}_{\mathsf{refRate}}$. Whenever time progresses according to the defined rate, $\mathcal{G}_{\mathsf{refRate}}$ signals all registered parties as a sign of time progress. The difference between $\mathcal{G}_{\mathsf{refClock}}$ [12] and $\mathcal{G}_{\mathsf{refRate}}$ is that $\mathcal{G}_{\mathsf{refRate}}$ does not have any notion of clock or time. It just informs the all functionalities to increase their own timer. We note that only ITI's *cannot* contact with $\mathcal{G}_{\mathsf{refRate}}$. The details are in Figure 1.

One may question whether the notion of reference rate contradicts with a trustless distributed system. It does not, because a progress of a time is globally defined and unchangeable fact in the real world (e.g., "One second is the time that elapses during 9.192631770×10^9 cycles of the radiation produced by the transition between two levels of the cesium 133 atom [2]").

We note that we do not aim to construct protocols that follow this rate. This functionality is useful to have the notion of relative time.

3.1.2 Local timer ($\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$)

The functionality $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ represents a local timer of a party P. $\Sigma \in \mathbb{Z}$ represents how much the local timer is drifted from the correct relative time (e.g. 1 hour passed according to $\mathcal{G}_{\mathsf{refRate}}$ since the initialization of P but the local timer says 58 minutes passed. So, the total drift is -2 minutes). $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ is accessible by the party P without any delay. $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ stores two types of timer: timer and timer^*. It increments timer whenever it receives a message from $\mathcal{Z}$ and increments timer^* whenever it receives a signal from $\mathcal{G}_{\mathsf{refRate}}$. However, it never shares timer^* (the real timer) with the party. The reason of having timer^* is to keep the amount of total drift (Σ) in any time. We define it in Figure 2.

$\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ keeps the uncorrupted timer timer^* too because keeping the parameter Σ is useful in ideal-real world simulations. Indeed, $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ does not differ from real world computer clocks because timer^* (the real clock) is never shared with the parties and the execution of the local timer does not depend on it. So, a usual local timer in the real world can be proven as a

realization of $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$.

Definition 3.1 (Correspondence of Timer Values)**.** *Assume that P_i and P_j sends $(\mathsf{Get\_Timer}, sid, P_i)$ to $\mathcal{F}_{\mathsf{timer}}^{T,P_i}$ and $(\mathsf{Get\_Timer}, sid, P_j)$ to $\mathcal{F}_{\mathsf{timer}}^{T,P_j}$ at the same time, respectively and $\mathcal{F}_{\mathsf{timer}}^{T,P_i}$ responds with $\mathsf{timer}_i = t_i$ and $\mathcal{F}_{\mathsf{timer}}^{T,P_j}$ responds with $\mathsf{timer}_j = t_j$. We call t_i is the corresponding timer value of t_j in timer_j in that moment (the moment that parties asked) and similarly we call that t_j is the corresponding timer value of t_i in timer_j in that moment.*

We denote by an algorithm $\mathsf{map}(\mathsf{timer}_i, \mathsf{timer}_j, t_i)$ the corresponding value of t_i in timer_j.

Before defining our consensus clock functionality, we first give some definitions to define clocks.

Definition 3.2 (Clock Value)**.** *A clock value is a natural number $\mathbb{N}$. Given that a clock value $C_{init} \in \mathbb{N}$ is assigned as a start time of a timer value t_{init}, the clock value of a timer $t \geq t_{init}$ is defined as follows:*

$$Clock(t, T, (C_{init}, t_{init})) = C_{init} + \lfloor \frac{t - t_{init}}{T} \rfloor \tag{1}$$

where $T \in \mathbb{N}$.

(C_{init}, t_{init}) serves as a local reference point to obtain a clock (defined below) that lets a party to determine the progress of its local clock (i.e., every T progress in the timer, the clock is incremented).

Definition 3.3 (Clock)**.** *A clock is a counter that increments every T progress in a timer after an initial assignment (C_{init}, t_{init}) (i.e., the clock value C_{init} started at timer t_{init}). The clock $\mathcal{C}$ of a party when timer shows t_{curr} is defined with $((C_{init}, t_{init}), C_{curr})$ where $C_{curr} = Clock(t_{curr}, T, (C_{init}, t_{init}))$.*

The font with calligraphy e.g. $\mathcal{C}$ represents the clock at t_{curr} while C represents the clock value which is any output from Equation (1).

We say that two clocks $\mathcal{C}_i = ((C_i, t_i), C)$ at time t and $\mathcal{C}_j = ((C_j, t_j), C')$ at time t' are *equivalent* if $C = C'$. Here, t is the corresponding value of t'.

Remark that if two clocks are equivalent at one time, they do not have to stay equivalent forever since the timers used to construct these two clocks may not follow the same rate later on.

Definition 3.4 (Time Difference of Clocks)**.** *Given that two parties provide the clocks $\mathcal{C}_i = ((C_{init}, t_{init}), C)$ and $\mathcal{C}_j = ((C'_{init}, t'_{init}), C')$ at the same time, we define $|\mathcal{C}_i - \mathcal{C}_j|$ as follows: Let $t_i = t_{init} + (C - C_{init})T$, $t'_i = t_{init} + (C' - C_{init})T$ (the start time of C and C' with respect to P_i's timer) and $t_j = t'_{init} + (C - C'_{init})T$, $t'_i = t_{init} + (C' - C_{init})T$ (the start time of C and C' with respect to P_j's timer). Then, the difference is $|\mathcal{C}_i - \mathcal{C}_j| = |t_j - t'_j| = |t_i - t'_i|$.*

We note that the time difference between clocks does not have to be constant because the timers can drift backward or forward. Therefore, we define the difference between clocks instantly.

Functionality $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$

$\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ is parameterized with T, Θ given by $\mathcal{Z}$. $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ first sends $(\mathsf{Register}, sid, \mathcal{F}_{\mathsf{Clock}}^{T,\Theta})$ and receives back $(\mathsf{Registered}, sid, P)$. Thus, it constructs its own timer.

- **(Initial Consensus)** Each party $P_i \in \mathcal{P}_h$ sends the message $(\mathsf{Get\_Clock}, sid, P_j, \mathcal{C}_j)$ where $\mathcal{C}_j$ is the clock of the party P_j. $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ runs the algorithm $\texttt{Clock\_Consensus}(\{\mathcal{C}_j\}_{P_j \in \mathcal{P}_h})$. If `Clock_Consensus` outputs $\perp$, then $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ sends $(\mathsf{Current\_Clock}, sid, \perp)$ to each P_j and aborts the protocol. If it outputs $\bar{\mathcal{C}}_h$, it sends $\{(\mathsf{Get\_Clock}, sid, P_j, \mathcal{C}_j)\}_{P_j \in \mathcal{P}_h}$ to $\mathcal{A}$.
- **(New Clocks)** $\mathcal{A}$ replies with $(\mathsf{Init\_Clock}, sid, \mathcal{F}_{\mathsf{Clock}}^{T,\Theta}, \tilde{C}_1, \tilde{C}_2, ..., \tilde{C}_n)$ for all $P_i \in \mathcal{P}_h \cup \mathcal{P}_c$.
- **(New Consensus)** $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ runs the algorithm $\texttt{Clock\_Consensus}(\{\mathcal{C}_j\}_{P_j \in \mathcal{P}_h \cup \mathcal{P}_c})$. If it outputs $\perp$ or if it outputs $\bar{\mathcal{C}}$ such that $|\bar{\mathcal{C}}_h - \bar{\mathcal{C}}| > \Theta$ or if there exists a $P_i \in \mathcal{P}_h$ such that $|\bar{\mathcal{C}} - \tilde{\mathcal{C}}_i| > \Theta$ sends $(\mathsf{Current\_Clock}, sid, P_i, \bar{\mathcal{C}})$ to each $P_i \in \mathcal{P}_h$. Otherwise, it sends $(\mathsf{Current\_Clock}, sid, P_i, \tilde{\mathcal{C}}_i)$ to each $P_i \in \mathcal{P}_h$.

Figure 3: The functionality $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$

3.1.3 Consensus Clock Provider ($\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$)

$\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ has a similar role with $\mathcal{G}_{\mathsf{Clock}}$ in the network time model [12]. Differently, in our functionality, the current clock is determined based on a consensus algorithm `Clock_Consensus` which outputs the current agreed clock or $\perp$ given a set of clocks provided by parties.

$\mathcal{G}_{\mathsf{Clock}}^{T,\Theta}$ is defined with the parameter T and Θ. T as in Equation (1) is the amount of time that defines the duration between one progress in the clock. Θ is the desynchronization parameter. $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ provides agreed clock to a requester party based on a consensus.

We call $\mathcal{P}_h$ is the set of honest parties and $\mathcal{P}_c$ is the set of corrupted parties

(Initial Consensus): At first, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ finds the initial consensus between honest parties. If it does not exist from the beginning, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ aborts the protocol. In more detail, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ first collects clocks from honest parties $P_i \in \mathcal{P}_h$. We note that some clocks can be null meaning that the honest party does not have any clock. $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ runs the algorithm `Clock_Consensus` with the honest clocks and obtains either $\perp$ or
a consensus clock of honest parties $\bar{\mathcal{C}}_h$. If it is $\perp$, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ sends an abort message to the adversary $\mathcal{A}$ and the honest parties and the protocol ends. Otherwise, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ gives the honest clocks to the adversary $\mathcal{A}$.

(New Clocks:) After receiving clocks of honest parties, $\mathcal{A}$ gives new clocks for honest parties $\{\tilde{\mathcal{C}}_i\}_{P_i \in \mathcal{P}_h}$ and clocks of the corrupted ones $\{\tilde{\mathcal{C}}_i\}_{P_i \in \mathcal{P}_c}$ to $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$.

(New Consensus): At this point, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ checks if a new consensus is possible with the new clocks. If the new consensus clock exists and it is close enough to the initial one, then all parties continue with their new clocks provided by $\mathcal{A}$. In more detail, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ runs the algorithm `Clock_Consensus` with the new honest clocks and corrupted clocks. If `Clock_Consensus` outputs $\perp$, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ sends $\bar{\mathcal{C}}_h$ (the initial consensus) to the honest parties. If it outputs a consensus clock $\bar{\mathcal{C}}$, $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ continues as follows: If $|\bar{\mathcal{C}} - \bar{\mathcal{C}}_h| \leq \Theta$ and $|\bar{\mathcal{C}} - \tilde{\mathcal{C}}_i| \leq \Theta$ for all $P_i \in \mathcal{P}_h$, it sends the $\tilde{\mathcal{C}}_i$ to each party $P_i \in \mathcal{P}_h$. Otherwise, it sends $\bar{\mathcal{C}}_h$ to each party $P_i \in \mathcal{P}_h$. More

details are in Figure 3.

In a nutshell, this functionality aims to provide clocks to honest parties which do not drift apart from the initial consensus that they have and which is close enough to the current consensus clock. Having the difference limit between the initial consensus clock and the new consensus clock is useful not to slow down the protocols relying on the output of $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$. For example, (a very extreme example) we do not want to end up with a new consensus clock that says that we are in the year 2001 even if we are in 2019. In this case, parties may wait 18 years to execute an action that is supposed to be done in 2019.

We note that `Clock_Consensus` can be defined based on the consensus definition of a protocol. For instance, we define our `Clock_Consensus` specific to our protocol.

We may have a stronger version of this functionality where honest parties do not have the initial consensus clock and still obtain a consensus clock. Our protocol in the next section runs on top of a blockchain protocol which assumes initial parties are synchronized. Therefore, we do not consider the stronger version.

3.2 UC- Partially Synchronous Blockchain Network Model

Our network model $\mathcal{F}_{\mathsf{DDiffuse}}$ is similar to the network model of Ouroboros Praos [15, 26, 20]. Differently, it accesses to $\mathcal{G}_{\mathsf{refRate}}$ in order to have the notion of relative time. Now, we define the functionality $\mathcal{F}_{\mathsf{DDiffuse}}$ which models a partially synchronous network with the time delay δ. Here, δ represents number of δ-increment message by $\mathcal{G}_{\mathsf{refRate}}$.

$\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$: The message handling functionality Diffuse for a blockchain network first introduced by Garay et al. [20] in the synchronous network where the message delivery is executed in a certain amount of known time. Then, David et al. [15] define a new functionality "delayed diffuse" (DDiffuse) for a blockchain network that realizes a partially synchronous network where a message arrives to others eventually but parties do not know how long it takes. DDiffuse is parameterized with the network delay parameter Δ and makes sure that all messages arrive to the receivers at most Δ-slots later. Here, slot is a duration to produce one block. However, it does not use any clock to determine how many slots passed since a message was sent. $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ is a version of DDiffuse with the access of $\mathcal{G}_{\mathsf{refRate}}$. It simply sends a given message from a party P_i to all other parties within a bound δ. $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ first registers to $\mathcal{G}_{\mathsf{refRate}}$ and creates a local timer $\mathsf{timer} = 0$. Whenever $\mathcal{G}_{\mathsf{refRate}}$ sends a message with Increase, it increments timer.

Each honest party P_i can access its inbox anytime. $\mathcal{A}$ can read all messages sent by the parties and decide their delivery order before they arrive to inboxes of honest parties. For any message coming from an honest party, $\mathcal{A}$ can label it as $\mathsf{delayed}_i$. When $\mathcal{F}_{\mathsf{DDiffuse}}$ receives $\mathsf{delayed}_i$ for a message to P_i, it marks it with the current local timer value t_i. A delayed message is not moved to the inbox of the P_i until $\mathcal{A}$ let $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ move it or the timer reaches $t_i + \delta$. In the end, all parties always receive any message sent by a party.

4 Realization of Consensus Clock

In this section, we describe our relative time protocol for blockchain protocols that let parties obtain a clock close to the consensus clock. Our relative time protocol realizes $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ with a consensus algorithm Clock_Consensus. Before describing the protocol, we first define the algorithm Clock_Consensus (Algorithm 1) that outputs the consensus clock. We note

that this is our way of defining consensus on clocks while every protocol can have different Clock_Consensus algorithm. After defining the algorithm, we describe our relative time protocol that lets parties agree on a clock according to Clock_Consensus.

4.1 Clock Consensus

Clock_Consensus algorithm receives clocks of parties as an input and outputs one of them as a consensus clock. We say that two clock is *synchronous* if they output the same clock at any time between a consecutive κ progress by G_{refRate}. For example, assume that κ is 5 seconds. A clock starts to output the clock value C in the first second and another clock starts to output C in the fourth second are synchronized even though they do not output the same clock in the sixth second (i.e., one outputs $C+1$ and the other outputs C in the sixth second). We say more than two clocks are synchronized if all pairs of these clocks are synchronized. In order to find the synchronized parties, Clock_Consensus first finds the corresponding time values (Definition 3.1) of $t_j = \tilde{t}_j + (C - \tilde{C}_j)T$ (start time of C according to timer_j) on timer of $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ which is $\mathsf{timer}_{\mathcal{F}_{\mathsf{Clock}}}$.

The start times of C according to given clocks help to find the synchronized parties. If honest parties are not synchronized, then the consensus is not satisfied and the algorithm outputs $\perp$. Otherwise, the algorithm finds the median of the start times and output the clock that corresponds the median as consensus clock.

Algorithm 1 Clock_Consensus($\{\mathcal{C}_j\}$) where $\mathcal{C}_j = ((\tilde{C}_j, \tilde{t}_j), C_j)$

1: $\mathsf{start\_lst} = \emptyset$
2: **pick** $C \in \mathbb{N}$ **such that** for all j such that $C > C_j$
3: **for all** $\mathcal{C}_j \neq null$ **do**
4: $\quad t'_j \leftarrow \mathsf{map}(\mathsf{timer}_j, \mathsf{timer}_{\mathcal{F}_{\mathsf{Clock}}}, t_j)$
5: $\quad$ **add** t'_j **to** start_lst
6: **for all** $t'_u, t'_v \in \mathsf{start\_lst}$ **that** belong to honest clock **do**
7: $\quad$ **if** $|t'_u - t'_v| > \kappa$ **then**
8: $\quad\quad$ **return** $\perp$
9: $t'_i \leftarrow \mathtt{Median}(\mathsf{start\_lst})$
10: **return** $\mathcal{C}_i$

4.2 Relative Time Protocol

We describe our protocol by assuming that there exists a blockchain protocol where parties produce blocks when it is their turn. In more detail, the blockchain protocol is defined as follows: After the genesis block released, each party starts their local timer. In every T progress of their timer according to $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ (See Figure 2), they increment their clock. Their clock starts from 0 when they receive the genesis block. The blockchain protocol has a selection mechanism which tells parties at which clock value they are supposed to produce a block. So, selected parties produce blocks only when their clock reaches the right clock value. Whenever they produce a block they also add their current clock value to the block as a timestamp. The environment $\mathcal{Z}$ activates parties just before the genesis block. It then can stop these parties or add new parties. We believe that Ouroboros [26], Ouroboros Praos [15], Ouroboros Genesis [6], Dfinity [22] and Snow White [14] can easily fit into this abstraction.

According to our abstraction, all initial active parties when the genesis block is released has clocks that they differ at most δ (network delay bound). However, as the time progresses, the total drift in their local clock (Σ in $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$) may change and so the difference between clocks can increase. In addition, new parties which are activated after the genesis block need to synchronize themselves with others to continue the blockchain protocol. Therefore, parties need to run the relative time protocol to obtain a clock which is close enough to the consensus clock. Our protocol does not provide perfect synchronization but it preserves a maximum difference between consensus clock and the clock that the algorithm offers to the parties during the execution of the blockchain protocol. The relative time protocol works as follows:

We divide the protocol in epochs. In each epoch, all active parties run the relative time protocol and update their clocks according to output of the protocol in the beginning of the next epoch. The first epoch starts just after the genesis block is released. The other epochs start when the clock value of the last finalized block is C_e which is the smallest clock value such that $C_e - C_{e-1} \geq s_{cd}$ where C_{e-1} is the clock value of the last finalized block in epoch $e-1$. Here, s_{cd} is the parameter of the chain density (CD) property (Definition 2.4). If the previous epoch is the first epoch then $C_{e-1} = 0$. We define the last finalized block as follows: Retrieve the best blockchain according to the chain selection rule of the blockchain protocol, trim the last k blocks of the best chain, the last block of the trimmed best chain is the last finalized block. Here, k is defined according to the common prefix property (Definition 2.1).

The party P stores the arrival time of valid blocks constantly. Whenever it receives a new valid block B_i', it sends $(\mathsf{Get\_Timer}, sid, P)$ to $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P}$ and obtains the arrival time t_i of B_i' according to his local timer. Let us denote the clock value of B_i' by C_i'. Here, a valid block should be valid according to the underlying blockchain protocol. We note that the clock values in these blocks do not have to be in a certain order because desynchronized or malicious parties may not send their blocks on time. At the end of the epoch, P retrieves the arrival times of **valid and finalized blocks** which has a clock value C_x where $C_{e-1} < C_x \leq C_e$. Let's assume that there are n such blocks that belong to the current epoch. Then, P selects a clock value $C > C_e$ [1]. Then, P runs the median algorithm (Algorithm 2). The median algorithm finds some candidate start times of C using the arrival time of blocks and then picks the median of them.

Algorithm 2 Median(C)

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

Assume that $\tilde{t}_s$ is the output of the median algorithm. Then, P considers $\tilde{t}_s$ as a start time of the clock value C (i.e., a local reference clock $(C, \tilde{t}_s)$) and finds the clock value C_{curr} of his current timer t_{curr} according to reference (C, t_C): $C_{curr} = Clock(t_{curr}, T, (t_C, C))$. P updates its clock when the new epoch starts.

[1] P can select any clock value but we prefer to make it the same for all parties for the sake of simplicity in proof.

The security of our protocol based on the security of CP and CD properties. CP property guarantees that all honest parties accept the same blocks as finalized blocks. Therefore, all honest parties run the median algorithm with using the arrival time of the same blocks. Thus, since the network delay is at most δ, the difference between median outputs of each honest parties is at most δ. So, after each epoch they return to synchronization level that they have just after the genesis block. The difference of a new clock and old clock of an honest party is limited thanks to the CD property that the blockchain protocol provides. The reason of this is that CD property guarantees that more than half of the blocks used in the median algorithm belong to honest parties. Because of a nice property of the median operation the output of the median should be between the minimum and maximum honest of clocks. The formal proof is as follows:

Theorem 4.1. *Assuming that the blockchain protocol preserves the common prefix property with the parameter k, and the chain density property with the parameter s_{cd} as long as the maximum difference between honest clocks is Θ and $2\delta + 2|\Sigma| \leq \Theta$ where $|\Sigma|$ is the maximum total drift between epochs and $\kappa \leq \delta$ in Consensus_Clock algorithm (Algorithm 1), the relative time protocol in $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ and $\mathcal{F}_{\mathsf{timer}}^{T,\Sigma}$-hybrid model realizes $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ except with the probability $p_{cp} + p_{cd}$ which are the probability of breaking CP and CD properties, respectively.*

Proof. In order to prove the theorem, we construct a simulator $\mathcal{S}$ where $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ and $\mathcal{F}_{\mathsf{timer}}^{T,\Sigma}$. When $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ gives the clock of honest parties $\{\mathcal{C}_i\}_{P_i \in \mathcal{P}_h}$ to $\mathcal{S}$, $\mathcal{S}$ starts to simulate each honest party in the real protocol according to these clocks. The simulation is straightforward. $\mathcal{S}$ sends $(\mathsf{Register}, sid, \mathcal{S})$ to $\mathcal{G}_{\mathsf{refRate}}$ in order to emulate $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P_i}$ and behave same as $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P_i}$ for all $P_i \in \mathcal{P}_h$. $\mathcal{S}$ simulates the honest parties in the underlying blockchain protocol too based on clocks given by $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$. For this, $\mathcal{S}$ rewinds or forwards the adversary to the round with the minimum current clock value (the second part of a clock tuple as defined in Definition 3.3) in clocks $\{\mathcal{C}_i\}_{P_i \in \mathcal{P}_h}$. For each active party P_j, it retrieves t_i from timer of $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P_j}$. Then, it learns the clock value at t', $C = Clock(t', T, (t_j, \tilde{C}_j))$ for P_j where t_j is the timer value when $\mathcal{F}_{\mathsf{Clock}}^{T,\Theta}$ gave honest clocks and produces a block if P_j is eligible to produce a block during C according to underlying blockchain protocol. If P_j is eligible, $\mathcal{S}$ sends the block of P_j to $\mathcal{A}$ (since $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ too). If $\mathcal{A}$ moves the block to the inbox of other honest parties, $\mathcal{S}$ stores the time that the block moved to the inbox of honest parties as the arrival time of this block. If the block is delayed by $\mathcal{A}$, $\mathcal{S}$ waits until $\mathcal{A}$ permits the block to move it. If the permission is not received after δ-increment signal by $\mathcal{G}_{\mathsf{refRate}}$, $\mathcal{S}$ moves the block to the inbox of honest parties. In either case, it stores timer of $\mathcal{F}_{\mathsf{timer}}^{\Sigma,P_i}$ when a block arrives from any other party. Remark that $\mathcal{S}$ knows the duration of δ because it receives the exact rate from $\mathcal{G}_{\mathsf{refRate}}$ while simulating the local clocks. During the simulation, $\mathcal{S}$ learns the clocks of corrupted parties in the epoch since it simulates $\mathcal{F}_{\mathsf{timer}}^{P,\Sigma}$ for a corrupted party as well. In the end of the epoch, $\mathcal{S}$ runs the Median algorithm (Algorithm 2) and updates the clocks of honest parties accordingly. $\mathcal{S}$ sends the clocks of corrupted parties and honest parties. We note that it is possible that a corrupted party sent the blocks according to different clocks. In this case, $\mathcal{S}$ sends all these different clocks as well. Finally, $\mathcal{S}$ outputs the clocks of honest parties.

The output of an honest party in the real world and the honest party in the ideal world are not the same if

1. there is no consensus according to Clock_Consensus or

2. the difference between the initial consensus and the final consensus clock is more than Θ or
3. the difference between the final consensus clock and an honest party's clock is more than Θ.

Now, we analyze the probability of having such bad events in our simulation in any epoch.

(1. Case and 3. Case): According to our Clock_Consensus (Algorithm 1), the consensus on clocks exists if and only if the difference between honest clocks is at most $\kappa \leq \delta$. Therefore, if we show that a consensus on clocks exists after the update then we also show that the difference between clocks of honest parties and the new consensus clock is at most $\delta \leq \Theta$. So, we first show that a consensus on clocks after the update exists given that **CP property is not broken during an epoch** except with probability p_{cp}. All honest parties run the median algorithm with the arrival time of the same blocks thanks to CP property. $\mathcal{F}^{\delta}_{\mathsf{DDiffuse}}$ guarantees that a block arrives all honest parties within δ increment in $\mathcal{G}_{\mathsf{refRate}}$. Therefore, the time difference of arrival time of any block differs at most δ between honest parties too. It implies that the time difference between median of all honest parties' lst in Algorithm 2 can be at most δ. Therefore, each honest P_i, P_j obtain a local reference clock (C, t_i), (C, t_j), respectively, such that $|t_i - t'_j| \leq \delta$ where t'_j is the corresponding timer value of t_j in the timer of P_i. Thus, the time difference between clocks after the update is at most δ too. Since $2\delta + 2|\Sigma| \leq \Theta$, consensus between clocks exists at the beginning of the epoch according to Algorithm 1. Now, we need to show that **the CP property is satisfied during all epochs** with induction. We know that at the beginning of the first epoch, the maximum difference between clocks of honest parties is δ because of our assumption after release of the genesis block. During the first epoch, the difference between the honest parties can be at most $2|\Sigma| + \delta < \Theta$ because of the clock drifts. Therefore, the CP property is preserved during the first epoch. Assume that the CP property is satisfied during the epoch x. Then, we show that the CP property is satisfied during the epoch $x + 1$. We know that if CP property is satisfied then the difference between clocks of honest parties is at most δ after running the the median algorithm at the end of the epoch x. So, honest parties start the epoch $x+1$ with the the clocks which has the difference δ at most. Because of the same reasons of the first epoch, the CP property is satisfied during the epoch $x + 1$ as well.

(2. Case) We know that the clocks of honest parties before update has a consensus since $\mathcal{F}^{T,\Theta}_{\mathsf{Clock}}$ gave them. Therefore, the time difference between their clocks is at most δ. Let us assume that the total drift of timers of honest parties during the simulation is at most $|\Sigma|$. Therefore, the clock difference of honest clocks can be at most $\delta + 2|\Sigma| \leq \Theta$ during the simulation. It is $2|\Sigma|$ because the drift can be forward or backward in the timeline. Therefore, the CD property is satisfied during an epoch. It means that majority of the blocks (at least $\lfloor \frac{n}{2} \rfloor + 1$ finalized blocks in the epoch) used in the median algorithm are honest ones except with the probability p_{cd}.

Now, we show the difference between the new consensus clock $\bar{\mathcal{C}}$ and the consensus clock $\bar{\mathcal{C}}_h$ just before computing the new one is at most $\Theta = T$ assuming that the case where $\lfloor \frac{n}{2} \rfloor + 1$ of the collected blocks during the simulation sent by honest parties. Let us assume that for a honest party P, the median algorithm outputted $\tilde{t} = t_i + a_i T$ where t_i is the arrival time of the block with clock C'_i. If the block with the clock value C'_i is sent by an honest party P_k, it is sent at the start time t'_i of the clock value C'_i. Because of $\mathcal{F}^{\delta}_{\mathsf{DDiffuse}}$, this block may be delayed. Therefore, $t'_i \leq t_i \leq t'_i + \delta$. We note that all these timer values are corresponding timer values to P's timer. Let us assume that C starts at time t^*_i according to P_k's clock in a moment that it sent the block with C'_i. The difference between the clock of P_k before updating the

clock and the clock of P after updating its clock is $\tilde{\mathcal{C}} - \mathcal{C}_k = \tilde{t} - t_i^*$. Since $t'_i \leq t_i \leq t'_i + \delta$ and $t_i^* = t'_i + a_i T$ we can conclude that $0 \leq \tilde{\mathcal{C}} - \mathcal{C}_k \leq \delta$. We also have $0 \leq \bar{\mathcal{C}} - \mathcal{C}_k \leq \delta$ because this inequality holds for all clocks including the new consensus clock after the update. We know that $|\bar{\mathcal{C}} - \mathcal{C}_h| \leq \delta + 2|\Sigma|$ so $\bar{\mathcal{C}} - \mathcal{C}_h \leq 2\delta + 2\Sigma \leq \Theta$.

We now show that the same inequality holds even if the median $\tilde{t}$ is computed from the clock value of an adversarial block. In this case, there exists a $t_x, t_y \in \mathsf{lst}$ (lst in Algorithm 2) where t_x and t_y are generated from clock values of honest block such that $t_x \leq \tilde{t} \leq t_y$ because of at least $\lfloor \frac{n}{2} \rfloor + 1$ of the collected blocks sent by honest parties. Since $\bar{\mathcal{C}} - \mathcal{C}_h \leq 2\delta + 2\Sigma \leq \Theta$ holds for all clocks between t_x and t_y, it should hold for adversarial $\tilde{t}$.

□

5 Conclusion

In this paper, we proposed a generic synchronization protocol that works on top of a blockchain protocol. Our synchronization protocol takes advantage of regular messaging process (e.g., blocks are sent regularly) to preserve consensus between honest parties' clocks. We also designed the first formal security model to capture the notion of consensus on clocks. Our security model is not specific to blockchain protocols. It can be used to show the existence of consensus clock in arbitrary protocol. We proved that our protocol secure in our new GUC security model.

Our protocol realizes our security model as long as the total network delay and the total drift in an epoch is not greater than $T/2$. Therefore, the parameters of blockchain (e.g., T) should be selected by considering this fact. We know that according to our assumption δ and Σ is not known in advance but it can be guessed based on some statistical analysis in the real world.

References

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

[2] Second (s or sec).

[3] H. Aissaoua, M. Aliouat, A. Bounceur, and R. Euler. A distributed consensus-based clock synchronization protocol for wireless sensor networks. *Wireless Personal Communications*, 95(4):4579–4600, 2017.

[4] H. K. Alper. Ouroboros clepsydra: Ouroboros praos in the universally composable relative time model.

[5] 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.

[6] 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.

[7] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros chronos: Permissionless clock synchronization via proof-of-stake. Cryptology ePrint Archive, Report 2019/838, 2019. https://eprint.iacr.org/2019/838.

[8] L. Bicknell. NTP issues today. outages mailing list. https://mailman.nanog.org/pipermail/nanog/2012-November/053449.html.

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

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

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

[17] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[18] 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.

[19] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149. ACM, 2003.

[20] 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.

[21] 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.

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

[23] J. He, P. Cheng, L. Shi, J. Chen, and Y. Sun. Time synchronization in wsns: A maximum-value-based consensus approach. *IEEE Transactions on Automatic Control*, 59(3):660–675, 2013.

[24] T. E. Humphreys, B. M. Ledvina, M. L. Psiaki, B. W. O'Hanlon, and P. M. Kintner. Assessing the spoofing threat: Development of a portable gps civilian spoofer. In *Radionavigation laboratory conference proceedings*, 2008.

[25] R. G. Johnston and J. Warner. Think GPS cargo tracking= high security? think again. *Proceedings of Transport Security World*, 2003.

[26] 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.

[27] C. Lenzen, P. Sommer, and R. Wattenhofer. Pulsesync: An efficient and scalable clock synchronization protocol. *IEEE/ACM Transactions on Networking (TON)*, 23(3):717–727, 2015.

[28] B. Luo, L. Cheng, and Y.-C. Wu. Fully distributed clock synchronization in wireless sensor networks under exponential delays. *Signal Processing*, 125:261–273, 2016.

[29] M. K. Maggs, S. G. O'keefe, and D. V. Thiel. Consensus clock synchronization for wireless sensor networks. *IEEE sensors Journal*, 12(6):2269–2277, 2012.

[30] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the network time protocol. In *NDSS*, 2016.

[31] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM, 2004.

[32] D. L. Mills. Computer network time synchronization. In *Report Dagstuhl Seminar on Time Services Schloß Dagstuhl, March*, volume 11, page 332. Springer, 1997.

[33] P. Papadimitratos and A. Jovanovic. Gnss-based positioning: Attacks and countermeasures. In *MILCOM 2008-2008 IEEE Military Communications Conference*, pages 1–7. IEEE, 2008.

[34] 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.

[35] L. Schenato and F. Fiorentin. Average timesynch: A consensus-based protocol for clock synchronization in wireless sensor networks. *Automatica*, 47(9):1878–1886, 2011.

[36] J. Selvi. Breaking ssl using time synchronisation attacks. In *DEF CON Hacking Conference*, 2015.

[37] R. Solis, V. S. Borkar, and P. Kumar. A new distributed time synchronization protocol for multihop wireless networks. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 2734–2739. IEEE, 2006.

[38] P. Sommer and R. Wattenhofer. Gradient clock synchronization in wireless sensor networks. In *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 37–48. IEEE Computer Society, 2009.

[39] N. O. Tippenhauer, C. Pöpper, K. B. Rasmussen, and S. Capkun. On the requirements for successful GPS spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86. ACM, 2011.

[40] J. Warner and R. Johnston. A simple demonstration that the global positioning system (gps) is vulnerable to spoofing, j. of secur. *Adm*, (1-9), 2002.

[41] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 142–153. ACM, 2005.

[42] G. Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2016.