Network Time with a Consensus on Clock

Handan Kılınç Alper

Web3 Foundation

handan@web3.foundation

Abstract

There are elaborate proof-of-stake based protocols in the UC-synchronous communication models, i.e. that all parties are aware of the current round of the protocol. However, in real-world blockchain protocols, such communication models are usually realized by computer clocks of nodes that progress to the next round after a certain amount of time passed from the previous round. These timers are adjusted by *centralized* systems such as Network Time Protocol (NTP) because these adjustments are indispensable to reduce the effects of random drifts on timers. On the other hand, an attack on these systems (which has happened in the past) can cause corruption of blockchains that rely on the time data that they provide. So, we are facing the dilemma of relying on the *centralized* solution to adjust our timers or risking the security of our *decentralized* protocols. In this paper, we propose a Global Universal Composable (GUC) model for the physical clock synchronization problem in the decentralized systems (not just blockchain protocols). In this way, we model the functionality that e.g. NTP provides in a decentralized manner. In the end, we give a simple but useful protocol relying on a blockchain network that realizes our model. Our protocol can be used by the full nodes of a blockchain that need synchronous *physical* clocks in the real world 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

In physics, time is defined as 'what a clock measures'. Nowadays, time is usually measurement of the number of vibrations happening on crystal oscillators in a clock e.g., a crystal oscillator vibrate 32768 times per second. So, the frequency of vibrations directly affects a computer's understanding of time. Unfortunately, the frequency of vibrations varies over time with small changes in the environment such as temperature, pressure, humidity thus causing clock drifts. As a result of this, the time knowledge of a clock starts to be inconsistent with the rest of the world. The computer clocks that are connected to the Internet mostly use Network Time Protocol (NTP) to determine the time instead of just relying on crystal oscillators so that the cumulative drift on the computer clock stays close to zero. In this way, not only clocks measure the time correctly but also the protocols such as certificate validation which require synchronized clocks can be executed securely.

The physical clocks are critical for the real-world implementation of some decentralized protocols which require synchronous communication [13, 14, 27, 33, 24, 21]. One of the

important ones are proof-of-stake based blockchain protocols [22, 12, 3, 4, 11, 16, 38]. Parties in PoS protocols usually realize this communication by measuring time with their local clocks in the real world implementations e.g., They start a new round every T ticks of the local clock corrected by NTP. So, the security of them reduces to the NTP's security which relies on the security of *centralized* servers. Unfortunately, the safety track record of NTP servers is not clean [26, 31, 28]. In one incident (on November 19, 2012) [5], two critical NTP servers were set to time twelve years in the past. This caused many important external servers, such as Active Directory authentication servers, to fail for a while. If the same attack happens in a PoS blockchain protocol even in a short time, honest parties would stop producing blocks because they think that their round did not come, while malicious full nodes would continue to produce blocks, populating the blockchain entirely with maliciously-produced blocks. Apart from NTP, some clocks are synchronized with the Global Positioning System (GPS). This requires a little more investment in the set-up but it is more accurate than NTP and does not have the potential problem of corrupted servers. However, GPS is also vulnerable to spoofing attacks [34, 20, 29, 19, 36] (e.g., delay signals). Even in the absence of an actual attack, mere poor weather conditions can cause inaccurate signals to be received from the GPS satellite. All of these existing solutions show that relying on an external system to build a correct local clock in decentralized protocols such as blockchains could be a major security vulnerability, as well as going against the ultimate goal of building a fully decentralized system.

In this paper, we model the problem of synchronization of (physical) clocks in a decentralized manner in the Generalized Universal Composable (GUC) model. In our model, parties have local timers (modelling crystal oscillators) and build local clocks that advances in every certain number of ticks of the timer. The environment can modify the frequency of ticks of timers. In such a model, clearly, the local clocks may drift apart even if they were the same at some point. Therefore, we constructed another functionality called consensus clock which realigns them. The consensus clock functionality receives the local clocks of parties in a protocol as a vote. Then, it gives new local clocks to all parties which are close enough to each other and close enough to their votes. Thus, the consensus clock functionality provides a way to decide the current right clock to be aligned in a decentralized network. We note that our aim *is not* designing a new model for the synchronous communication which has elaborate UC-security models [8, 21, 23]. We deal with physical clocks which have an ability to measure time while global clock functionalities in the synchronous communication models are logical clocks which proceed based on some events in the protocol independent from actual time.

In more detail, our contributions are as follows:

- We construct a GUC model that captures the notion of consensus on clock and allows parties in a decentralized network to align their clock with it. Our model realizes situations where a party has a local clock constructed based on the ticks of his local timer and wants to synchronize the local clock with the other existing local clocks in the network without any reference clock. We define a global functionality which sets 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 drifted timers in the real world. Our other functionality receives initial clocks from parties as a vote and provides new clocks to them which are close to each other and not drifted apart from the initial clocks. To the best of our knowledge, our model is the first security model for the notion of consensus on clock.
- We construct a basic clock synchronization protocol (BCSP) which shows that as long as parties agree on the messages with timestamps delivered during the protocol, they

can construct close clocks without further communication. In BCSP, parties record the clock information given in the messages and order them in the end. In the end, they construct a new clock based on the clock information of order a defined in the protocol. Even if order a is different for different parties, we show that the difference between new clocks is not more than the maximum network delay and clock drift happening during the protocol. Thus, the final difference depends on the network delay, the duration of the protocol, and the frequency error of the clocks.

- We construct the *Relative Time* protocol on top of blockchain protocols that realizes consensus on clock functionality in our model. The relative time protocol is based on BCSP where parties select the median of the sorted clock information in the blocks. The assumption of agreement on messages in BSCP is satisfied by the security of the blockchain protocol. The parties construct local clocks as in our model that count the rounds of the blockchain protocol and behave based on the round information provided by their local clocks. Periodically, all parties update their clocks by running the relative time protocol. The relative time protocol does not add extra communication to the network or any extra weight to blocks. It just uses the round information in the block (which already exists) and the local arrival time of blocks. We achieve this by taking advantage on the existing consensus mechanism on the blockchain. Our protocol can be adapted to all blockchain protocols that fit our abstraction [22, 12, 3, 11]. Thus, we solve the physical clock synchronization problems of PoS-based blockchain protocols in a decentralized manner without putting an extra overhead to the network and the blockchain.

We note that our relative time protocol is in the implementation process to be used in Polkadot [38, 6] which aims to connect multiple blockchain networks.

1.1 Related Works

UC Clock Models: The network time model by Canetti et al. [10] is the first model that defines clocks with the ability to measure time. The security of a local clock is defined based upon the closeness of a reference clock. So, the ultimate goal in this model is to obtain local clocks close enough to the reference clock. This type of clock model is useful for the protocols accepting one clock as valid and expect from all other parties to follow this clock. For example, the Public Key Infrastructure (PKI) limits the validity of certificates and expects from all users to consider the validity of a certificate within the same duration. Differently, in our model, the ultimate goal of parties is to have clocks that are close enough to eachother without any reference clock. It is not important how local clocks are close to a reference clock (e.g. a clock defined by NTP, GPS). Our model is useful for protocols (e.g., blockchain protocols) which do not need to rely on any real-world notion of the correct time for security and completeness.

There are UC models [8, 21, 23] designed to emulate the synchronous communication between parties but they do not provide ways for realizing the functionalities from real-world solutions such as network time protocols or local physical clocks. The clock functionalities in these models are different from physical clocks. They keep the round of a protocol and make sure that all parties are in the same or close rounds which is decent to model the synchronous communication. We note that we do not aim to construct a model for synchronous communication even though there are some name resemblance among clock notions. Instead, we model the physical clock notion and functionality to keep them close enough when they

drift apart.

PoS protocols: The security of some PoS blockchain protocols [22, 12, 3, 11] is preserved in the synchronous communication model however the only solution to preserve it in the real world implementations to rely on parties' physical clocks which are adjusted by centralized protocols such as NTP. Ouroboros Chronos [4] which is an improved version of Ouroboros Genesis adds a mechanism that helps adjusting the physical clocks of parties without relying on any external Internet service such as NTP as our relative time protocol. The synchronization mechanism in Ouroboros Chronos is tailored for Ouroboros Genesis while our protocol is more generic in this sense which can be constructed on top of coherent PoS protocols with our abstraction. As such, our protocol does not modify the block generation mechanism of the underlying PoS protocol and does not introduce any extra messages in the network.

Clock Synchronization Protocols: Clock synchronization protocols [13, 14, 27, 33, 24] have been extensively studied in previous work, as it is a important component in distributed systems. Many protocols [25, 30, 17, 2] exist for consensus clocks with different assumptions. The advantage of consensus synchronization is that it does not require a tree topology or a root node. One approach is based on dividing the network into some well-connected clusters [13, 39] that aim to achieve consensus between clusters. There are also fully distributed approaches [37, 32, 18] based on clock skew and network delay estimation. The main idea of the consensus clock is to agree on clock information collected from neighbor nodes and to be synchronized accordingly. Our relative time protocol is also based on the same idea. Differently, we consider the case where the agreement is done via a blockchain protocol. We analyze the minimum security requirement from the blockchain protocol for good synchronization. We aim not only the synchronization between nodes but also we want to preserve the distance between the old clock and the new clock after the synchronization. To the best of our knowledge, the previous consensus clock protocols do not consider this requirement. In addition, we provide the first formal cryptographic security model for the consensus clock.

2 Security Model

In this section, we introduce our new GUC-model for the consensus clock, and then give the UC-model for a partially synchronous network similar to the model in [12, 3]. We note that we refer physical clocks that measure the time in our model. They are not logical clocks.

We have multiple interactive Turing machines (ITM)'s and 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 an identifier consisting of a session identifier sid and the ITM identifier pid. A party P_i in UC model is an ITI with the identifier $(\mathsf{sid}_i, \mathsf{pid}_i)$. The environment $\mathcal{Z}$ activates all ITM's and the adversary and then creates ITIs. $\mathcal{A}$ is also activated whenever the ideal functionalities are invoked. $\mathcal{A}$ can corrupt parties $P_1, P_2, ..., P_n$. If a party is corrupted, then whenever it is activated, its current state is shared with $\mathcal{A}$. See Appendix A for more detail about UC model.

2.1 The Model for the Consensus of Clocks

In this section, we construct a model where parties have access to their local timers (e.g., computer clocks in real life). Local timers are supposed to tick according to a certain global metric time, but they may not follow the metric time because of adversarial interruptions.

We use local timers to define local clocks in our model. In such an environment, we model how parties synchronize their clocks. Our model consists of the following functionalities:

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

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

- ***(Registration)*** If a functionality $\mathcal{F}$ sends a message $(\mathsf{Register}, \mathsf{sid}, \mathcal{F})$, $\mathcal{G}_{\mathsf{refRate}}$ adds $(\mathcal{F}, \mathsf{sid})$ to $\mathsf{list}_{\mathcal{F}}$ and sends $(\mathsf{Registered}, \mathsf{sid}, \mathcal{F})$ to $\mathcal{F}$.
- ***(Progress in Time)*** In every tick according to the defined rate, $\mathcal{G}_{\mathsf{refRate}}$ sends $(\mathsf{Tick}, \mathsf{sid}, P)$ to all $(P, \mathsf{sid}) \in \mathsf{list}_{\mathcal{F}}$

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

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

$\mathcal{G}_{\mathsf{refRate}}$ defines a metric time. It can be considered as a global timer that ticks with respect to the metric time (e.g., second). The entities who want to be notified in every tick register with $\mathcal{G}_{\mathsf{refRate}}$. Whenever $\mathcal{G}_{\mathsf{refRate}}$ ticks, it informs them. The difference between $\mathcal{G}_{\mathsf{refClock}}$ [10] and $\mathcal{G}_{\mathsf{refRate}}$ is that $\mathcal{G}_{\mathsf{refRate}}$ does not have any absolute values related to time. We note that ITI's *cannot* contact with $\mathcal{G}_{\mathsf{refRate}}$. The details are in Figure 1.

One may question whether the concept of a reference rate contradicts the distributed system without a global clock. It does not, because a metric system for time is universally defined based on a physical phenomenon (e.g., [1]). In real world, all physical clocks are designed to follow a global definition of time although they have difficulties to follow it at some point because of their nature.

We note that we do not aim to construct clocks that precisely follow this global rate. This functionality is intended to be able to measure the real time.

2.1.2 Local timer ($\mathcal{G}^{P}_{\mathsf{timer}}$):

We define the global functionality $\mathcal{G}^{P}_{\mathsf{timer}}$ to model the local clock. The local timer ticks with a rate that may vary in time (as crystal oscillators of clocks in the real world). We choose the name timer because its absolute value does not matter as a clock. The local timers measure how much time passed relative to P with the help of $\mathcal{G}^{P}_{\mathsf{timer}}$. We define $\mathcal{G}^{P}_{\mathsf{timer}}$ in *the GUC model* to capture the fact that the real world timers interact with arbitrary protocols. $\mathcal{G}^{P}_{\mathsf{timer}}$ lets functionalities to see the progress of the timer. This property makes any functionality to construct the same clock as P . See Figure 2 for the details of $\mathcal{G}^{P}_{\mathsf{timer}}$.

Next, we define notions related to clocks which are constructed with respect to local timers to label the time. Clocks can be used to count locally the round of a protocol which progresses depending on time.

Definition 2.1 (Clock Interval)**.** *Clock interval $T \in \mathbb{N}$ is the minimum time that a clock measures.*

Functionality $\mathcal{G}_{\mathsf{timer}}^{P}$

$\mathcal{G}_{\mathsf{timer}}^{P}$ is identified with a parameter $\mathsf{sid}_{\mathsf{timer}}$ and its owner P.

- ***(Functionality Registration)*** When $\mathcal{G}_{\mathsf{timer}}^{P}$ receives a message $(\mathsf{Register}, \mathsf{sid}_{\mathcal{F}}, \mathcal{F}, P)$ from a functionality $\mathcal{F}$, $\mathcal{G}_{\mathsf{timer}}^{P}$ replies with $(\mathsf{Registered}, \mathsf{sid}_{\mathcal{F}}, \mathcal{F}, P)$ and adds $\mathcal{F}, \mathsf{sid}_{\mathcal{F}}$ to the list $\mathsf{list}_{\mathcal{F}}$.
- ***(Ticking)*** When $\mathcal{Z}$ sends an $(\mathsf{Increase}, P)$ request, $\mathcal{G}_{\mathsf{timer}}^{P}$ sends the message $(\mathsf{Ticked}, ¶, \mathsf{timer})$ to P. It also sends the message $(\mathsf{Ticked}, \mathsf{sid}_{\mathcal{F}}, \mathcal{F}, P)$ to $\mathcal{F}$ where $(\mathcal{F}, \mathsf{sid}_{\mathcal{F}}) \in \mathsf{list}_{\mathcal{F}}$.

Figure 2: The global functionality $\mathcal{G}_{\mathsf{timer}}^{P}$

Definition 2.2 (Clock)**.** *A clock $\boldsymbol{\mathcal{C}}_\ell$ runs the algorithm (See Algorithm 1) initiated by the party P_ℓ with the parameters: clock initial timer value $\bar{t}_\ell$, the timer start value s_ℓ and the clock interval T. The clock outputs the clock value c_ℓ whenever $\mathcal{G}_{\mathsf{timer}}^{P_\ell}$ ticks.*

Algorithm 1 $\textsc{Clock}(\bar{t}_\ell, s_\ell, T)$

1: $\mathsf{timer}_\ell = s_\ell$
2: **while** $\mathcal{G}_{\mathsf{timer}}^{P_\ell} \rightarrow \mathsf{tick}$ **do**
3: $\quad \mathsf{timer}_\ell = \mathsf{timer}_\ell + 1$
4: $\quad$ **output** $c_\ell = \frac{\mathsf{timer}_\ell - \bar{t}_\ell}{T}$

Remark that $\bar{t}_\ell$ serves as a local reference point that lets a clock determine its clock value at a given timer value (i.e., starting from $\bar{t}$, every T ticks in timer, increase the clock value).

Definition 2.3 (Frequency)**.** *The frequency of a clock $\boldsymbol{\mathcal{C}}_\ell$ between $\mathsf{timer}_\ell = t$ and $\mathsf{timer}_\ell = u$ is the number of ticks from $\mathsf{timer}_\ell = t$ till $\mathsf{timer}_\ell = u$. We let $f_\ell(u,t)$ be the frequency of $\boldsymbol{\mathcal{C}}_\ell$ between u and t. Here, $f_\ell(u,t) = t - u$.*

Definition 2.4 (Frequency Error)**.** *The frequency error of a clock $\boldsymbol{\mathcal{C}}_\ell$ between $\mathsf{timer}_\ell = t$ and $\mathsf{timer}_\ell = u$ is $\epsilon_\ell(u,t) = f_\ell(u,t) - f_{\mathsf{ref}}(u,t)$ where $f_{\mathsf{ref}}(u,t)$ is the number of ticks by $\mathcal{G}_{\mathsf{refRate}}$ from $\mathsf{timer}_\ell = u$ to $\mathsf{timer}_\ell = t$.*

When $\epsilon_\ell(u,t)$ is positive, it means that $\mathcal{G}_{\mathsf{timer}}^{P}$ is faster than $\mathcal{G}_{\mathsf{refRate}}$. Otherwise, it means that $\mathcal{G}_{\mathsf{timer}}^{P}$ is slower than $\mathcal{G}_{\mathsf{refRate}}$. The frequency error shows the deviation from the correct time. We also define the frequency error of $\boldsymbol{\mathcal{C}}_\ell$ with respect to another clock $\boldsymbol{\mathcal{C}}_k$ as $\epsilon_{\ell \triangleright k}(u_\ell, t_\ell) = f_\ell(u_\ell, t_\ell) - f_k(u_k, t_k)$. Here, u_k is the value shown on timer_k while $\mathsf{timer}_\ell = u_\ell$ and t_k is the value shown on timer_k while $\mathsf{timer}_\ell = t_\ell$. Since $f_{\mathsf{ref}}(u_\ell, t_\ell) = f_{\mathsf{ref}}(u_k, t_k)$,

$$\begin{aligned}
\epsilon_{\ell \triangleright k}(u_\ell, t_\ell) &= f_\ell(u_\ell, t_\ell) - f_k(u_k, t_k) \\
&= f_\ell(u_\ell, t_\ell) - f_{\mathsf{ref}}(u_\ell, t_\ell) - f(u_k, t_k) + f_{\mathsf{ref}}(u_k, t_k) \\
&= \epsilon(u_\ell, t_\ell) - \epsilon(u_k, t_k)
\end{aligned}$$

The frequency error of real world clocks defined in terms of part-per-million (PPM). Some best clocks have 10^{-8} PPM-error which corresponds to around 1.5 nanosecond-error in a day, some usual clocks have error between 1-20 ppm. The environment conditions (e.g., temperature, air pressure, magnetic fields). We collected the frequency error of computer clocks during 5 months and analyse the result in Section 4.

We assume that the timer of the clock was ticking with respect to the reference clock until the clock is actually initiated by the party. This assumption does not change the properties of the clock but it simplifies the relational analysis between two clocks $\boldsymbol{\mathcal{C}}_\ell$ and $\boldsymbol{\mathcal{C}}_k$. Accordingly, we define the clock equivalence as follows: The equivalent clock of $\boldsymbol{\mathcal{C}}_\ell$ is $\boldsymbol{\mathcal{C}}_{\ell^k}$ based on $\boldsymbol{\mathcal{C}}_k$. $\boldsymbol{\mathcal{C}}_{\ell^k}$ outputs the clock value whenever $\mathcal{G}^{P_\ell}_{\mathsf{timer}}$ ticks. The start value of $\boldsymbol{\mathcal{C}}_{\ell^k}$ is s_{ℓ^k} and the initial timer value is $\bar{t}_{\ell^k} = s_{\ell^k} - (s_\ell - \bar{t}_\ell)$. Here, s_{ℓ^k} is the timer value of another clock $\boldsymbol{\mathcal{C}}_k$ when $\boldsymbol{\mathcal{C}}_\ell$ is initiated. Remark that $\boldsymbol{\mathcal{C}}_\ell$ and $\boldsymbol{\mathcal{C}}_{\ell^k}$ output the same clock value i.e., $c_\ell = \frac{\mathsf{timer}_\ell - \bar{t}_\ell}{T} = \frac{f_\ell(\bar{t}_\ell, \mathsf{timer}_\ell)}{T} = \frac{f_\ell(s_\ell, \mathsf{timer}_\ell) - |s_\ell - \mathsf{timer}_\ell|}{T} = \frac{f_\ell(s_{\ell^k}, \mathsf{timer}_{\ell^k}) - |s_{\ell^k} - \mathsf{timer}_{\ell^k}|}{T} = \frac{f_\ell(\bar{t}_{\ell^k}, \mathsf{timer}_{\ell^k})}{T} = c_{\ell^k}$.

The equivalent clock of $\boldsymbol{\mathcal{C}}_\ell$ based on $\boldsymbol{\mathcal{C}}_k$ is useful in our analysis when we need to compare the timer values on clocks when certain events happen. For example, we can obtain the difference between initial timer values of $\boldsymbol{\mathcal{C}}_\ell$ and $\boldsymbol{\mathcal{C}}_{\ell^k}$ by computing $(\bar{t}_{\ell^k} - \bar{t}_k)$. Whenever we need to compare two timer values on timer_ℓ and timer_k, we consider the equivalent clock of timer_ℓ based on $\boldsymbol{\mathcal{C}}_k$.

We next define the notion of null clock. The parties who does not have enough information to construct a clock start the protocol with a null clock.

Definition 2.5 (Null Clock)**.** *A null clock runs the algorithm* NullClock *(Algorithm 2) which does not have clock initial timer value and clock interval as an input. Thus, it does not output any clock value but labels the each tick of* $\mathcal{G}^{P_\ell}_{\mathsf{timer}}$.

Algorithm 2 NullClock(s_ℓ)

1: $\mathsf{timer}_\ell = s_\ell$
2: **while** $\mathcal{G}^{P_\ell}_{\mathsf{timer}} \rightarrow \mathsf{tick}$ **do**
3: $\quad \mathsf{timer}_\ell = \mathsf{timer}_\ell + 1$

Definition 2.6 (Clock Difference)**.** *Given two clocks* $\boldsymbol{\mathcal{C}}_k$ *and* $\boldsymbol{\mathcal{C}}_\ell$*, the time difference of clocks* $\boldsymbol{\mathcal{C}}_k$ *and* $\boldsymbol{\mathcal{C}}_\ell$ *when* timer_k *is* t_k *and* timer_ℓ *is* t_ℓ *is* $\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_\ell = |c_k - c_\ell| = \frac{|(t_k - \bar{t}_k) - (t_\ell - \bar{t}_\ell)|}{T}$.

Alternatively, the clock difference is equal to

$$
\begin{aligned}
\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_\ell &= \boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_{\ell^k} \\
&= \frac{|(t_k - \bar{t}_k) - (t_{\ell^k} - \bar{t}_{\ell^k})|}{T} \\
&= \frac{|(t_k - t_{\ell^k}) - (\bar{t}_k - \bar{t}_{\ell^k})|}{T} \\
&= \frac{|s_k + f_k(s_k, t_k) - s_{\ell^k} - f_\ell(s_{\ell^k}, t_{\ell^k}) - (\bar{t}_k - \bar{t}_{\ell^k})|}{T} \\
&= \frac{|s_{\ell^k} + f_k(s_{\ell^k}, t_k) - s_{\ell^k} - f_\ell(s_{\ell^k}, t_{\ell^k}) - (\bar{t}_k - \bar{t}_{\ell^k})|}{T} \\
&= \frac{|\epsilon_{k \triangleright \ell} - (\bar{t}_k - \bar{t}_{\ell^k})|}{T} \qquad (1)
\end{aligned}
$$

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

$\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ is parametrized with T and $\Theta = (\Theta_c, \Theta_p)$ given by $\mathcal{Z}$ and identified by a session id $\mathsf{sid}_{\mathsf{C\_Clock}}$. It maintains a set of ITI's $P_i = (\mathsf{pid}_i, \mathsf{sid})$ in $\mathcal{P}_h$. $\mathcal{P}_h$ is initially empty but it is populated when a party registers with its identifier pid_i and its session identifier sid.

- (***Timer Registration:***) When an honest $P_i = (\mathsf{pid}_i, \mathsf{sid})$ registers, $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ sends $(\mathsf{Register}, \mathsf{sid}_{\mathsf{C\_Clock}}, \mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}})$ to $\mathcal{G}^{\Sigma,P_i}_{\mathsf{timer}}$ and receives back $(\mathsf{Registered}, \mathsf{sid}_{\mathsf{C\_Clock}}, P)$.
- ***(Initial Vote)*** After each party $P_j \in \mathcal{P}_h$ sends the message $(\mathsf{InitialTimer}, \mathsf{sid}, \mathsf{pid}_j, \bar{t}_j, s_j)$ where $\bar{t}_j$ is the initial timer value of the P_j's clock $\boldsymbol{\mathcal{C}}_j$ or the value null, $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ constructs clock $\boldsymbol{\mathcal{C}}_j$. It sends $(\mathsf{Clock}, \mathsf{sid}, \mathcal{A}, \boldsymbol{\mathcal{C}}_j)$ to $\mathcal{A}$ for all $P_j \in \mathcal{P}_h$.
- ***(New Clocks)*** $\mathcal{A}$ replies with $(\mathsf{NewClock}, \mathsf{sid}, \mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}, \{\tilde{\boldsymbol{\mathcal{C}}}_j\}_{(\mathsf{pid}_j, \mathsf{sid}) \in \mathcal{P}})$. If there exists $\tilde{\boldsymbol{\mathcal{C}}}_j = \mathsf{null}$, $\mathcal{F}^{T,\Theta}_{\boldsymbol{\mathcal{C}}}$ aborts and sends $(\mathsf{Abort}, \mathsf{sid}, \mathsf{pid}_j, \perp)$ to all $P_j \in \mathcal{P}_h$. Otherwise, it continues with the next phase.
- ***(Consensus)*** If $\boldsymbol{\mathcal{C}}_j - \tilde{\boldsymbol{\mathcal{C}}}_j \leq \Theta_c$ for all $P_j \in \mathcal{P}_h$ that voted not null clock and $\tilde{\boldsymbol{\mathcal{C}}}_i - \tilde{\boldsymbol{\mathcal{C}}}_j \leq \Theta_p$ for all pairs $P_i, P_j \in \mathcal{P}_h$, it sends $(\mathsf{SynchronizeWith}, \mathsf{sid}, \mathsf{pid}_j, \tilde{\boldsymbol{\mathcal{C}}}_j)$ to each party $P_j \in \mathcal{P}_h$. Otherwise, it aborts and sends $(\mathsf{Abort}, \mathsf{sid}, \mathsf{pid}_j, \perp)$ to all $P_j \in \mathcal{P}_h$

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

2.1.3 Consensus on Clock ($\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$)

We construct $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ which helps parties to shorten the distance between their clocks when they drift apart. We define $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ with the clock interval T and the desynchronization parameter $\Theta = (\Theta_c, \Theta_p)$. We denote the set of honest parties by $\mathcal{P}_h$. $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ works as follows:

(**Timer Registration:**) $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ registers itself to the local timer functionalities $\mathcal{G}^{P}_{\mathsf{timer}}$ of honest parties to construct their clocks.

(Initial Vote): Each party P_i sends the clock initial timer value $\bar{t_i}$ and the timer start value s_i of their clocks to $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$. If P_i does not have any clock, he sends null and the timer start value s_i as an indicator that he does not have a clock. After receiving all of inputs from honest parties, $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ constructs clocks of each P_i as $\boldsymbol{\mathcal{C}}_i$ according to the Definition 2.2 and null clocks according to Definition 2.5 if they exist. $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ sends $\boldsymbol{\mathcal{C}}_i$ of each P_i to $\mathcal{A}$. In meantime, $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ waits for a response from $\mathcal{A}$.

(New Clocks:) $\mathcal{A}$ responds with new clocks $\tilde{\boldsymbol{\mathcal{C}}}_i$ for each $P_i \in \mathcal{P}_h$ If there exists a null clock among new clocks then $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ aborts.

(Consensus): At this point, $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ checks if a consensus exists between new clocks. It exists if the difference between the new clock and the clock of a party is at most Θ_c and each pairwise difference of new clocks is at most Θ_p. If the consensus exists, he sends the new

clocks provided by the adversary to each corresponding party. Otherwise, he aborts. More details of $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$ are in Figure 3.

In a nutshell, this functionality aims to provide close clocks to honest parties which do not drift apart much from the initial clock of parties. Having the bound Θ_c between the initial clock and the new clock is useful not to slow down the protocols relying on the output of $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$. For example, we do not want to end up with a new clock showing year 2001 when the corresponding initial clock shows 2019. In this case, parties may wait 18 years to execute an action that is supposed to be done in 2019. If a protocol does not have such concern, θ_c should be considered as ∞.

2.2 UC- Partially Synchronous Network Model

Our network model $\mathcal{F}_{\mathsf{DDiffuse}}$ is similar to the network model of Ouroboros Praos [12, 22, 15]. 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 $\delta_{\max}$. Here, $\delta_{\max}$ represents number of $\delta_{\max}$-increment message by $\mathcal{G}_{\mathsf{refRate}}$.

$\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$: Each party P_i has access to its message inbox populated by $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$. $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ first registers to $\mathcal{G}_{\mathsf{refRate}}$ and creates a local timer to measure the real time. Whenever $\mathcal{G}_{\mathsf{refRate}}$ sends a message with Tick, it increments it. When P_i sends a message, $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ sends it to $\mathcal{A}$ and starts to measure the time until receiving message from $\mathcal{A}$ that says 'send' or until δ ticks passed since sending the message to $\mathcal{A}$. When the time measurement is done for a message, $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ puts it to the inbox of all parties. In this model, a message arrives to all parties at most $\delta_{\max}$ ticks later than the time it is sent.

2.3 The Result from Combination of Models

In this section, we describe a protocol that we call basic clock synchronization protocol (BCSP) in the hybrid model $\mathcal{G}_{\mathsf{timer}}^{P}$ and $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$. Parties of BCSP obtain new clocks where their distance from other new clocks according to Definition 2.2 is within a bound that depends on the network delay and the frequency error of old clocks during the protocol. Our relative time protocol runs in the blockchain protocol is based on BCSP.

2.3.1 Basic Clock Synchronization Protocol (BCSP):

Consider m honest ITI's $P_1, ..., P_m$ initiated with parameters $n \geq 1$, $0 \leq a \leq n$ given by $\mathcal{Z}$ and clock parameters: timer start value s_i, clock initial timer value $\bar{t}_i$ and clock interval T. After the initiation, each party P_k constructs a clock $\boldsymbol{\mathcal{C}}_k$ that runs $\textsc{Clock}(\bar{t}_k, s_k, T)$. In this protocol, whenever $\mathcal{Z}$ requests to P_k to send a message, P_k obtains the current clock value c_i without any delay after the request and sends a message B_i including c_i (as a timestamp) with $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$. When an ITI P_ℓ receives a message B_i, it stores $u_{i\to\ell} = t_{i\to\ell} - c_i T$ to a list list_ℓ where $t_{i\to\ell}$ is the arrival time of B_i according to P_ℓ's timer timer_ℓ. We consider $u_{i\to\ell}$ as a candidate initial timer value of the new clock which is close to the P_i's initial timer value. After receiving n messages, each party P_k constructs a new clock $\tilde{\boldsymbol{\mathcal{C}}}_k$ where the initial timer value is $\mathtt{sort}(\mathsf{list}_\ell)[a]$ is the a^{th} element of sorted list_ℓ and the start value is the end of the protocol e_k i.e, new clock $\tilde{\boldsymbol{\mathcal{C}}}_k$ runs $\textsc{Clock}(\mathtt{sort}(\mathsf{list}_\ell)[a], e_k, T)$.

The critical assumption in the BCSP is that parties do not receive more than n messages. So, it implies that they agree on them. We have this assumption to abstract the consensus mechanism and authentication layer for the messages from the clock synchronization based

on the arrival times of them. However, BSCP requires a consensus mechanism to achieve this assumption in its real-world applications. Our analysis shows that the correctness of timestamps in the messages does not matter as long as all parties obtain the list which is constructed from the same messages.

Before starting the analysis of the protocol, we should find the corresponding value of $u_{j\to\ell}$ for all B_j on timer $\boldsymbol{\mathcal{C}}_\ell$ on the equivalent clock $\boldsymbol{\mathcal{C}}_{\ell^k}$. This helps us to compute the distance between $u_{j\to\ell}$ and $u_{i\to k}$. We know that the message B_j is sent when $\mathsf{timer}_\ell = t_{j^\ell}$ and $\mathsf{timer}_k = t_{j^\ell}$, $u_{j\to k} = t_{j^k} + f_k(t_{j^k}, t_{j^k} + \delta_{j\to k}) - c_j T = t_{j^k} + \delta^\epsilon_{j\to k} - c_j T$ and $u_{j\to\ell} = t_{j^\ell} + f_k(t_{j^\ell}, t_{j^\ell} + \delta_{j\to\ell}) - c_j T = t_{j^\ell} + \delta^\epsilon_{j\to\ell} - c_j T$ where $\delta_{j\to\ell}$ and $\delta_{j\to k}$ are the network delays of message B_i. We also know that the corresponding value of t_{j^ℓ} on clock $\boldsymbol{\mathcal{C}}_\ell$ is $t = s_{\ell^k} + f_\ell(s_{\ell^k}, t_{j^\ell})$. Since $t_{j^k} = s_{\ell^k} + f_k(s_{\ell^k}, t_{j^k})$, $t = t_{j^k} + f_\ell(s_\ell, t_{j^\ell}) - f_k(s_{\ell^k}, t_{j^k})$. So, the corresponding value of $u_{j\to\ell}$ on the equivalent clock $\boldsymbol{\mathcal{C}}_{\ell^k}$ is $u_{j\to\ell^k}$ which is

$$\begin{aligned} u_{j\to\ell^k} &= t + \delta_{j\to\ell} - c_j T \\ &= t_{j^k} - f_k(s_{\ell^k}, t_{j^k}) + f_\ell(s_\ell, t_{j^\ell}) + \delta_{j\to\ell} - c_j T \\ &= t_{j^k} + \epsilon_{\ell\triangleright k}(t_j) + \delta_{j\to\ell} - c_j T \end{aligned}$$

Therefore,

$$u_{j\to k} - u_{j\to\ell^k} = \delta^\epsilon_{j\to k} - \delta^\epsilon_{j\to\ell} - \epsilon_{\ell\triangleright k}(t_j) \tag{2}$$

Lemma 2.1. *Given that* $\mathtt{sort}(\mathsf{list}_k)[a] = u_{i\to k}$ *and* $\mathtt{sort}(\mathsf{list}_\ell)[a] = u_{j\to\ell}$, $|u_{j\to\ell^k} - u_{i\to k}| \leq \delta^\epsilon_{\max} - \epsilon_{\ell\triangleright k}(t_x)$ *where* B_x *is a message sent in the protocol and the frequency of all timers within* $\delta_{\max}$*-ticks is at most* $\delta^\epsilon_{\max}$*.*

Proof. We have the following cases:

1. $u_{j\to k} \leq u_{i\to k}$ and $u_{i\to\ell^k} \leq u_{j\to\ell^k}$: Using the the facts $u_{i\to\ell^k} - u_{j\to\ell^k} \leq 0$ and $u_{i\to k} - u_{i\to\ell^k} \leq \delta^\epsilon_{i\to k} - \delta^\epsilon_{i\to\ell} - \epsilon_{\ell\triangleright k}(t_i)$, we obtain $u_{i\to k} - u_{j\to\ell^k} \leq \delta^\epsilon_{i\to k} - \delta^\epsilon_{i\to\ell} - \epsilon_{\ell\triangleright k}(t_i) \leq \delta^\epsilon_{\max} - \epsilon_{\ell\triangleright k}(t_i)$.

2. $u_{i\to k} \leq u_{j\to k}$ and $u_{j\to\ell^k} \leq u_{i\to\ell^k}$: Using the the facts $u_{i\to k} - u_{j\to k} \leq 0$ and $u_{j\to k} - u_{j\to\ell^k} \leq \delta^\epsilon_{j\to k} - \delta^\epsilon_{j\to\ell} - \epsilon_{\ell\triangleright k}(t_j)$, we obtain $u_{i\to k} - u_{j\to\ell^k} \leq \delta^\epsilon_{j\to k} - \delta^\epsilon_{j\to\ell} - \epsilon_{\ell\triangleright k}(t_j) \leq \delta^\epsilon_{\max} - \epsilon_{\ell\triangleright k}(t_j)$.

3. $u_{j\to k} \leq u_{i\to k}$ and $u_{j\to\ell^k} \leq u_{i\to\ell^k}$: In this case there are $n - a$ more elements in $\mathtt{sort}(\mathsf{list}_k)$ after $u_{i\to k}$ and $n - a$ more elements in $\mathtt{sort}(\mathsf{list}_\ell)$ after $u_{j\to\ell^k}$. Let's denote the last $n - a$ elements of $\mathtt{sort}(\mathsf{list}_k)$ and $\mathtt{sort}(\mathsf{list}_\ell)$ by $\mathcal{A}_k$ and $\mathcal{A}_\ell$, respectively. $u_{i\to\ell^k} \in \mathcal{A}_\ell$ but $u_{i\to k} \notin \mathcal{A}_k$ but $|\mathcal{A}_k| = |\mathcal{A}_\ell|$. Therefore, there must exit x such that $u_{x\to k} \in \mathcal{A}_k$ but $u_{x\to\ell^k} \notin \mathcal{A}_\ell$. This implies that $u_{j\to k} \leq u_{i\to k} \leq u_{x\to k}$ and $u_{x\to\ell^k} \leq u_{j\to\ell^k} \leq u_{i\to\ell^k}$.

   Using the the facts $u_{i\to k} - u_{x\to k} \leq 0$, $u_{x\to\ell^k} - u_{j\to\ell^k} \leq 0$ and $u_{x\to k} - u_{x\to\ell^k} \leq \delta^\epsilon_{x\to k} - \delta^\epsilon_{x\to\ell} - \epsilon_{\ell\triangleright k}(t_x)$, we obtain $u_{i\to k} - u_{j\to\ell^k} \leq \delta^\epsilon_{x\to k} - \delta^\epsilon_{x\to\ell} - \epsilon_{\ell\triangleright k}(t_x) \leq \delta^\epsilon_{\max} - \epsilon_{\ell\triangleright k}(t_x)$.

4. $u_{i\to k} \leq u_{j\to k}$ and $u_{i\to\ell^k} \leq u_{j\to\ell^k}$: Using the the facts $u_{i\to\ell^k} - u_{j\to\ell^k} \leq 0$, and $u_{i\to k} - u_{i\to\ell^k} \leq \delta^\epsilon_{i\to k} - \delta^\epsilon_{i\to\ell} - \epsilon_{\ell\triangleright k}(t_i)$, we obtain $\delta^\epsilon_{i\to k} - \delta^\epsilon_{i\to\ell} - \epsilon_{\ell\triangleright k}(t_i) \leq \delta^\epsilon_{\max} - \epsilon_{\ell\triangleright k}(t_i)$.

□

Theorem 2.2. *The clock difference of two new clocks $\tilde{\mathcal{C}}_k$ and $\tilde{\mathcal{C}}_\ell$ just after running BCSP protocol is at most $\frac{|2\epsilon_{\max}+\delta^{\epsilon}_{\max}|}{T}$ where the frequency error between old clocks during the protocol is at most $\epsilon_{\max}$ and at least $-\epsilon_{\max}$ and the frequency of all timers within $\delta_{\max}$-ticks is at most $\delta^{\epsilon}_{\max}$.*

Proof. The BCSP protocol ends when $\mathsf{timer}_k = e_k$ and when $\mathsf{timer}_\ell = e_\ell$. Let's denote the corresponding value of e_ℓ on the equivalent clock $\boldsymbol{\mathcal{C}}_{\ell^k}$ is e_{ℓ^k}. Without loss of generality, let us assume that $e_k \geq e_{\ell^k}$. In this case, when P_k receives the last message of the BCSP protocol, P_ℓ has already constructed its clock. Let's denote the timer value of $\tilde{\mathcal{C}}_\ell$ when P_k constructs $\tilde{\mathcal{C}}_k$ by t_ℓ

$$
\begin{aligned}
|\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_\ell| &= \frac{|(e_k - u_{i\to k}) - (t_\ell - u_{j\to \ell})|}{T} \\
&= \frac{|\epsilon_{\ell\triangleright k}(e_k) + (u_{j\to \ell^k} - u_{i\to k})|}{T} \text{ (from Equation (1))} \\
&\leq \frac{|\epsilon_{\ell\triangleright k}(e_k) + \delta^{\epsilon}_{\max} - \epsilon_{\ell\triangleright k}(t_x)|}{T} \\
&\leq \frac{|2\epsilon_T + \delta^{\epsilon}_{\max}|}{T} \qquad (3)
\end{aligned}
$$

□

This theorem shows us that if we can construct a protocol where parties agree on the messages with timestamps and construct a new clock with respect to the order of messages as described in BCSP, they can construct clocks with the difference at most $\frac{|2\epsilon_T+\delta^{\epsilon}_{\max}|}{T}$ just after the protocol ends. After the protocol ends, the difference may increase according to the frequency error of the clocks ($\epsilon_{\ell\triangleright k}(e_k, t_k)$). Therefore, they need to rerun the BCSP at some point to decrease the difference again to $\frac{|2\epsilon_T+\delta^{\epsilon}_{\max}|}{T}$. Our relative time protocol is based on running the BCSP protocol on blockchain periodically. Since the BCSP protocol requires agreement on messages, we get help from the nature of the blockchain protocol.

3 Realization of Consensus on Clock

In this section, we describe our relative time protocol for blockchain protocols that realizes the functionality $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$. Before describing the protocol, we give some preliminary definitions related to security of blockchain.

3.1 Blockchain

A blockchain protocol is an interactive protocol between parties to construct a blockchain. Garay et al. [15] define some properties given below to obtain a secure blockchain protocol. In these definitions, the blockchain protocol follows the logical clock $r_1, r_2, ...$ in the synchronous communication [8, 21, 23]. We note that these rounds do not have to advance based on the measurement of time as our clocks.

Definition 3.1 (Common Prefix (CP) Property [15])**.** *The CP property with parameters $k \in \mathbb{N}$ ensures that any blockchains $\boldsymbol{B_i}, \boldsymbol{B_j}$ possessed by two honest parties at the onset of rounds r_i, r_j where $r_i < r_j$ satisfy that $\boldsymbol{B_i}^{\lceil k}$ is the prefix of $\boldsymbol{B_j}$ where $\boldsymbol{B_i}^{\lceil k}$ is the blockchain without the last k blocks of $\boldsymbol{B_i}$.*

In other words, the CP property ensures that blocks which are k blocks before the last block of an honest party's blockchain is always prefix of the the blockchain that is going to be owned by an honest party. We call the blocks which is going to be the prefix of all future honestly constructed blockchains *finalized* blocks and the blockchain including the finalized blocks *final blockchain*.

We modify the chain quality property (CQ) by Garay et al. and give chain density property. CQ property ensures the existence of sufficient honest blocks on any blockchain owned by an honest party.

Definition 3.2 (Chain Density (CD) Property)**.** *The CD property with parameters* $s_{cd} \in \mathbb{N}, \mu \in (0,1]$ *ensures that any portion* $\boldsymbol{B}[r_u : r_v]$ *of a final blockchain* $\boldsymbol{B}$ *spanning between rounds* r_u *and* $r_v = r_u + s_{cd}$ *contains at least* $|\boldsymbol{B}[r_u : r_v]|\mu$ *honest blocks where* $|\boldsymbol{B}[r_u : r_v]|$ *is the number of blocks in* $\boldsymbol{B}[r_u : r_v]$.

The CD property ensures a minimum ratio of honest blocks in the final blockchain. In our protocol, we need CD property with $\mu > 0.5$ which implies that the sufficiently long span of the final blockchain contains more honest blocks than malicious ones.

3.2 Hybrid Blockchain Protocol

We describe the high-level blockchain protocol $\pi_{\mathcal{H}}$ in the hybrid model with $\mathcal{F}^{\delta}_{\mathsf{DDiffuse}}, \mathcal{G}^{P}_{\mathsf{timer}}$ and $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$. We do not give the details about some algorithms of $\pi_{\mathcal{H}}$ in gray lines in Algorithm 3 because they are related with the synchronization of physical clocks. We describe $\pi_{\mathcal{H}}$ on top of a blockchain protocol which works in rounds. Rounds are specific time intervals (T ticks according to $\mathcal{G}_{\mathsf{refRate}}$) associated for the block production. In each round, some parties are selected as a block producer who generates and sends their blocks for the corresponding round. Each block contains its round number. Parties follow the rounds based on their own local physical clocks. We also define larger interval called epoch. The interval of an epoch does not depend on time but depends on happening of certain events explained below. In the end of each epoch, the parties update their clocks with the output of $\mathcal{G}^{T,\Theta}_{\mathsf{C\_Clock}}$ to obtain close clocks which possibly drifted apart during the epoch. We abstract the hybrid blockchain protocol $\pi_{\mathcal{H}}$ as shown in Algorithm 3.

A block producer P_i in Algorithm 3 runs firstly the Initialization algorithm and obtains current state of the protocol. The blockchain state includes all necessary information (e.g., currently produced blockchains) in order to participate $\pi_{\mathcal{H}}$ actively. If the state is the genesis block, as soon as receiving the genesis block from $\mathcal{F}^{\delta}_{\mathsf{DDiffuse}}$, P_i lets the clock initial timer value $\bar{t}_i = 0$ and the timer start value $s_i = 0$. Then, he constructs his clock $\boldsymbol{\mathcal{C}}_i$ which runs the algorithm $\mathsf{Clock}(\bar{t}_i, s_i, T)$ (Algorithm 1). Then, he lets $r_e = 0$ which is completed epoch round. Otherwise, P_i registers to the protocol to participate it by running Register. Since state does not have the clock information, he creates a null clock. Then in order to obtain a clock, he sends his null clock $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ and obtains a new clock (not null). After contacting with $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ or after the genesis-block release, P_i follows the rounds with his clock which increments in every T ticks received from $\mathcal{G}^{P_i}_{\mathsf{timer}}$. In every start of the new round, he checks whether it is his round to produce block by running IsMyRound algorithm. Meanwhile, if he receives a block, he updates the state with UpdateState. After each state change, P_i checks whether the new epoch starts with the IsNewEpoch algorithm. Given that the end of the previous epoch's round is r_e, IsNewEpoch returns true if the round of the last finalized block (See Definition 3.1) in state is r such that $r - r_e \geq s_{cd}$. Here, s_{cd} is the parameter of the

chain density (CD) property (Definition 3.2). If the new epoch starts, P_j sends his clock to $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$ and obtains a clock $\tilde{\mathcal{C}}_i$ from $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$. Then, he continues the protocol with his new clock.

Algorithm 3 $P_i(\mathsf{sid}, \mathsf{pid})$ in $\pi_{\mathcal{H}}$

1: **run** INITIALIZATION(P_i)
2: state $\leftarrow \mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$
3: **if** state $= B_0$ // if it is genesis **then**
4: $\bar{t}_i \leftarrow \mathcal{G}_{\mathsf{timer}}^{P_i}$
5: **let** $\mathcal{C}_i$ run CLOCK$(0, 0, T)$
6: $r_e \leftarrow 0$
7: registered $=$ true
8: **else**
9: **run** REGISTER(P_i)
10: registered $=$ true
11: state $\leftarrow \mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$
12: $\mathcal{C} \leftarrow$ null
13: **send** $(\mathsf{InitialTimer}, \mathsf{sid}, \mathsf{pid}, \mathsf{null})$ to $\mathcal{G}_{\mathsf{C\_Clock}}^{T,\Theta}$
14: **receive** $(\mathsf{SynchronizeWith}, \mathsf{sid}, \mathsf{pid}, \mathcal{C}_i)$
15: **while** P_i.registered $=$ true **do**
16: $r \leftarrow \mathcal{C}_i$
17: **if** ISMYROUND(state) **then**
18: $B \leftarrow$GENERATEBLOCK(state)
19: **send** (B, P_i) with $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$
20: **if** B **received from** $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ **then**
21: state $\leftarrow$ UPDATESTATE(state, B)
22: **if** ISNEWEPOCH(r_e, state) **then**
23: **send** $(\mathsf{InitialTimer}, \mathsf{sid}, \mathsf{pid}, \mathcal{C}_i)$ to $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$
24: **receive** $(\mathsf{SynchronizeWith}, \mathsf{sid}, \mathsf{pid}, \tilde{\mathcal{C}}_i)$
25: $r_e \leftarrow$NEWEPOCH(r_e, state)
26: **run** DEREGISTER(P_i) **return**

Clearly, the difference between the clocks of block producers in π_B after releasing the genesis block is δ and Θ_p after synchronizing with the consensus clock given by $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$. During an epoch, this difference can increase $\Sigma_{\max}$ more because of the frequency error on local timers. In addition, after obtaining new clock from $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$, the difference between the old and new clock of the party can be most Θ_c ticks. Given that the party updates his clock in round r_x, the new clock can show that the party is in round $r_x \pm \frac{\Theta_c}{T}$. If it is $r_x - \frac{\Theta_c}{T}$, he should rerun the rounds between $r_x - \frac{\Theta_c}{T}$ and r_x. If it is $r_x + \frac{\Theta_c}{T}$, he has to skip the rounds between r_x and $r_x + \frac{\Theta_c}{T}$. Therefore, having small Θ_c is critical for the security of the blockchain.

Now we describe our relative time protocol that we can replace $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$ in $\pi_{\mathcal{H}}$.

3.2.1 Relative Time Protocol

We replace the lines between 13-14 and 23-24 in Algorithm 3 with our relative time protocol which realizes $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$ and obtain a new hybrid protocol π. Remark that they are the lines that P_i obtains the clock from $\mathcal{F}_{\mathsf{C\_Clock}}^{T,\Theta}$.

In our protocol, as soon as P_k registers or receives the genesis block, he constantly stores the arrival time of *valid* blocks. The validity of the block is defined according to the underlying blockchain protocol. We also require that a valid block should include the round for which it is produced. Whenever it receives a new *valid* block B_i, it obtains the arrival time $t_{i\to\ell}$ which is the timer value $t_{i\to k}$ on timer_k when B_i arrives. Let us denote the round of B_i by r_i. We note that rounds in these blocks do not have to arrive in a certain order because desynchronized or malicious parties may not send their blocks on time. If P_i has a null clock meaning that he just registered, he waits until storing all arrival time of blocks in one full epoch.

When an epoch ends, P_k retrieves the arrival times of **valid and finalized blocks** which have a round r_x where $r_e < r_x \leq r$ where r is the output of NEWEPOCH. Let us assume

that there are n such blocks that belong to the current epoch and these blocks are in a set $\mathcal{B}$. Let us also assume that the list of their arrival time relative to the P_k's timer is $\{t_{i\to k}\}_{\forall B_i \in \mathcal{B}}$. Then, P_k runs the median algorithm (Algorithm 4) which finds prospective initial timer value of each clock of blocks relative to his timer and adds them to a list list_k. In the end, it sorts list_k and returns the median of list_k. Remark that list_k constructed in MEDIAN algorithm is the same with list_k in BCSP in Section 2.3. Differently, in BCSP, parties stop after receiving n messages, but in the relative time protocol, the parties stop when the epoch ends.

Algorithm 4 MEDIAN($\{t_{i\to k}\}_{\forall B_i \in \mathcal{B}}$)

1: $\mathsf{list}_k \leftarrow \emptyset$
2: **for** $B_i \in \mathcal{B}$ **do**
3: **store** $u_{i\to k} = (t'_i - r_i T)$ to list_k // prospective initial timer value of clock of the party that produced B_i
4: **return** `median`(list_k)

Assuming that $u_{i\to k}$ is the output of the median algorithm, P_k constructs a new clock with the clock initial timer value $u_{i\to k}$ and the timer start value is the timer value when the epoch ends.

Remark that the median algorithm does not depend on the current clock. Therefore, even if a party has a null clock meaning that he does not have any clock, he should run the relative time protocol for one epoch to obtain a clock that is close enough to others. Of course, such parties cannot actively contribute to the block production mechanism during the epoch that they joined because they do not have the round information. Hence, if a party wants to join the blockchain protocol, he should join at least one epoch before to have enough time to construct his clock.

Next, we show that the relative time protocol realizes the hybrid protocol $\pi_{\mathcal{H}}$ assuming that the underlying blockchain protocol preserves the CP and CD properties as long as the clock difference of parties is less than or equal to $\lfloor \frac{3\epsilon_{\max}+\delta^{\epsilon}_{\max}}{T} \rfloor$. Parties bootstrap their clocks according to the arrival time of the genesis block at first. At this point, their clock difference is at most $\frac{\delta^{\epsilon}_{\max}}{T}$. So, the blockchain protocol preserves the CP and CD properties after the genesis block. Assuming that the frequency error between clocks is at most $\epsilon_{\max}$ in one epoch, the difference between clocks cannot change more than $\epsilon_{\max}$ in one epoch. We show that the difference between new clocks is at most $\frac{\lfloor 2\epsilon_{\max}+\delta^{\epsilon}_{\max} \rfloor}{T}$ when parties construct their new clock in the end of the epoch. Thus, we preserve the assumption of having $\lfloor \frac{3\epsilon_{\max}+\delta^{\epsilon}_{\max}}{T} \rfloor$-difference between clocks during the lifetime of the blockchain protocol.

Theorem 3.1. *Assuming that π_B preserves the common prefix property with the parameter k, and the chain density property with the parameter $s_{cd}, \mu > 0.5$ as long as the maximum difference between honest clocks is $\lfloor \frac{3\epsilon_{\max}+\delta^{\epsilon}_{\max}}{T} \rfloor$ and $\Theta_p = \frac{\lfloor 2\epsilon_{\max}+\delta^{\epsilon}_{\max} \rfloor}{T}, \Theta_c = \frac{\lfloor \epsilon_{\max}+2\delta^{\epsilon}_{\max} \rfloor}{T}$, the relative time protocol in $\mathcal{F}^{\delta}_{\mathsf{DDiffuse}}$ and $\mathcal{G}^{T,\Sigma}_{\mathsf{timer}}$-hybrid model realizes $\mathcal{F}^{T,\Theta}_{\mathsf{C\_Clock}}$ except with the probability $p_{cp} + p_{cd}$ which are the probability of breaking CP and CD properties, respectively.*

Here, the frequency error between old clocks in one epoch is at most $\epsilon_{\max}$ and at least $-\epsilon_{\max}$ and the frequency of all timers within $\delta_{\max}$-ticks is at most $\delta^{\epsilon}_{\max}$.

The security of our protocol is based on the security of the CP and CD properties. The CP property guarantees that all honest parties accept the same blocks as finalized blocks. Therefore, all honest parties run the MEDIAN algorithm using the arrival time of the same

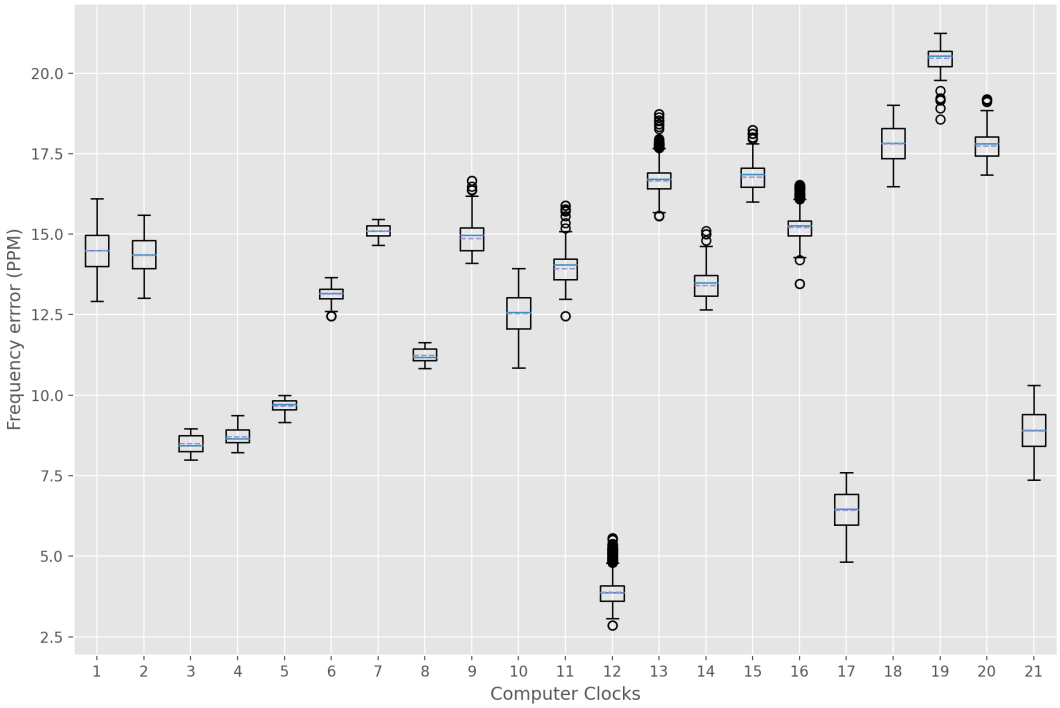

Figure 4: Clock frequency error data. The dashed line is the standard deviation and the line is the median. Each box shows the distribution of the data based on five numbers: maximum, third quartile, median, first quartile and minimum in order. The circles are outliers.

blocks. Thus, we can use the result of Theorem 2.2. In this way, after each epoch, the difference between the clocks are bounded. The difference between a new clock and an old clock of an honest party is limited thanks to the CD property that the blockchain protocol provides. The reason for this is that CD property guarantees that more than half of the blocks used in the median algorithm belong to honest parties. Thanks to a nice property of the median operation, the output of the median should be between the minimum and maximum honest of clocks.

Proof. In order to prove the theorem, we construct a simulator $\mathcal{S}$ where $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$ in the relative time protocol π and simulates the adversary in $\pi_{\mathcal{H}}$. It first registers $\mathcal{G}_{\mathsf{refRate}}$ to emulate $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$. The simulation is straightforward. As soon as, $\mathcal{Z}$ initiates a party P_i, $\mathcal{S}$ simulates P_i in π. Whenever P_i in $\pi_{\mathcal{H}}$ sends message with $\mathcal{F}_{\mathsf{DDiffuse}}^{\delta}$, P_i in π sends the same message and $\mathcal{S}$ waits until $\mathcal{A}$ permits the message to move the inbox of the other parties. If the permission is not received after δ consecutive ticks by $\mathcal{G}_{\mathsf{refRate}}$, $\mathcal{S}$ moves the block to the inbox of honest parties in π and $\pi_{\mathcal{H}}$. In either case, it stores the arrival time of the block according to the parties clock which could be null. When $\mathcal{S}$ receives clocks $\boldsymbol{\mathcal{C}}_j$'s from $\mathcal{F}_{\mathsf{C\_Clock}}$, $\mathcal{S}$ runs the Median algorithm (Algorithm 4) and updates the clocks of honest parties accordingly. $\mathcal{S}$ sends the updated clocks of honest parties to $\mathcal{F}_{\mathsf{C\_Clock}}$. Finally, $\mathcal{S}$ outputs the

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

1. the difference between new clocks provided by $\mathcal{S}$ is more than Θ_p.
2. the difference between new clocks and the old clocks is more than Θ_c
3. one of the new clocks of honest parties is null.

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

(1. Case): Thanks to the CP property, all honest parties run the median algorithm with the arrival time of the same blocks. It means that list in Algorithm 4 includes the prospective initial timer value of clocks of the same blocks. Therefore, the difference between the new clocks is at most $\frac{|2\epsilon_{\max}+\delta^{\epsilon}_{\max}|}{T}$ thanks to Theorem 2.2 i.e., for all $P_i, P_j \in \mathcal{P}_h$, $\boldsymbol{\mathcal{C}_j} - \boldsymbol{\mathcal{C}_i} \leq \frac{|2\epsilon_{\max}+\delta^{\epsilon}_{\max}|}{T}$ just after running the relative time protocol.

(2. Case) The clock difference of parties after all parties receive the genesis block is at most $\frac{\delta^{\epsilon}_{\max}}{T}$. During the first epoch, this difference increases to $\epsilon_{\max}$. So, the clock difference of honest parties during the first epoch is $\frac{\delta^{\epsilon}_{\max}+\epsilon_{\max}}{T}$. In the end of the relative time protocol for a later epoch e, the difference of the new clocks are at most $\frac{|2\epsilon_{\max}+\delta^{\epsilon}_{\max}|}{T}$. During the next epoch $e+1$, this difference increases to $\epsilon_{\max}$. Thus, the clock difference of honest parties is at most $\frac{|3\epsilon_{\max}+\delta^{\epsilon}_{\max}|}{T}$ during the simulation. So, 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}.

We now show the difference between the new clock $\tilde{\boldsymbol{\mathcal{C}}}_k$ and the old clock $\boldsymbol{\mathcal{C}}_k$ just before the simulation starts is at most Θ_c assuming that $\lfloor \frac{n}{2} \rfloor + 1$ of the finalized blocks during the simulation were sent by honest parties. Let us assume that P_k's median algorithm outputs $u_{i\to k}$ corresponding to the block B_i for round r_i sent by the honest party P_ℓ. Given that the equivalent clock of $\boldsymbol{\mathcal{C}}_\ell$ based on $\boldsymbol{\mathcal{C}}_k$ is $\boldsymbol{\mathcal{C}}^k_\ell$, we let the corresponding initial timer value be $\bar{t}_{\ell^k}$ on timer_{ℓ^k}. We know from Lemma 2.1 that $|\bar{t}_k - \bar{t}_{\ell^k}| \leq |\delta_{\max} + \epsilon_{\max}|$ where $\bar{t}_k$ is the initial timer value of the old clock $\boldsymbol{\mathcal{C}}_k$. Also, $|\bar{t}_{\ell^k} - u_{i\to\ell}| = |\bar{t}_{\ell^k} - t_{i\to k} + r_i T| \leq |t_{i^k} - t_{i\to k}| \leq \delta_{\max}$ where t_{i^k} is the value of timer_{ℓ^k} when B_i is sent. From these two inequalities, we obtain that $|\bar{t}_k - u_{i\to k}| \leq |2\delta_{\max} + \epsilon_{\max}|$. Thus, $\boldsymbol{\mathcal{C}}_k - \tilde{\boldsymbol{\mathcal{C}}}_k = \frac{|\bar{t}_k - u_{i\to k}|}{T} \leq |\frac{2\delta_{\max}+\epsilon_{\max}|}{T}$

If B_i is not sent by an honest party, the same inequality holds. In this case, there exists $u_{y\to k}$ where $u_{i\to k} \leq u_{y\to k}$ and $u_{x\to k} \leq u_{i\to k}$ where B_y and B_y are sent by honest parties because at least $\lfloor \frac{n}{2} \rfloor + 1$ elements of list_k constructed with respect to the honest parties' blocks. We showed that $|\bar{t}_k - u_{x\to k}| \leq |2\delta_{\max} + \epsilon_{\max}|$ and $|\bar{t}_k - u_{y\to k}| \leq |2\delta_{\max} + \epsilon_{\max}|$ when B_x and B_y are sent by honest parties. So, if $u_{i\to k} \geq \bar{t}_k$, $|u_{i\to k} - \bar{t}| \leq |u_{y\to k} - \bar{t}| \leq |2\delta_{\max} + \epsilon_{\max}|$. Otherwise, $|\bar{t}_k - u_{i\to k}| \leq |\bar{t}_k - u_{x\to k}| \leq 2\delta_{\max} + \epsilon_{\max}|$. Thus, $\boldsymbol{\mathcal{C}}_k - \tilde{\boldsymbol{\mathcal{C}}}_k = \frac{|\bar{t}_k - u_{i\to k}|}{T} \leq |\frac{2\delta_{\max}+\epsilon_{\max}|}{T}$.

(3. Case): Since CD and CP property is preserved between epochs as shown in case 1, 2 and 3, there are finalized blocks between epochs so list in Algorithm 4 is never empty, so new clocks are never null.

□

4 Frequency Error of Computer Clocks

Our result depends on the network delay, the accuracy of the system clocks and length of the epoch. When the epoch length increases, the effect of the frequency error in the final difference is increasing as well because of the drift. In order to see this effect in the real world computers, we collect the frequency error of 21 different computer clocks during five

months. The Network Time Protocol daemon (npdp) collects the drift data (also known as frequency error) of the system clock written to `/var/lib/ntp/ntp.drift` in terms of parts per million (PPM). This file is updated every hour. We give the result of our data in Figure 4. The data shows us that the frequency error does not vary too much in time but the frequency error is not close to 0. It means that system clocks are stable but not accurate. The stability of clocks between clock 2 and clock 9 are much better. If the relative time protocol is run with respect to these clocks, the frequency error difference between clocks is at most $\epsilon_{19\triangleright 12} = 20.22 - 2.82 = 17.4$ ppm. It means that if one epoch takes t-seconds, it implies that the frequency error between two clocks is at most $t \times 17.4 \times 10^{-6}$ seconds. Accordingly, the difference between clocks after running the relative time protocol is at most $\frac{\delta_{\max}+2t\times 17.4\times 10^{-6}}{T}$. For example, if one epoch takes 12 hours, the frequency error in the end of the epoch is $(12 \times 60 \times 60) \times 20 \times 10^{-6} \approx 0.75$ seconds. So, the maximum difference of clocks in the end of an epoch is $\frac{\delta_{\max}+1.5}{T}$ seconds.

The frequency error in real world computers are related to the oscillator type of the system clock. Nowadays, computers use crystal oscillators (XO). External XO's are stable but their accuracy (10-100 ppm) is worse than other types of XO's. TCXOs has the accuracy 1-20 ppm and OCXOs has the accuracy 0.5 ppm [35]. They are designed to minimize the temperature effect. The more advanced XO is MEMS which has the accuracy 0.008 ppm [35]. The block producers in our protocol can minimize the frequency error by investing the oscillator so that they achieve much better synchronization in our protocol. Thus, the affect of the frequency error to the difference of the clocks can be negligible.

5 Conclusion

We constructed a GUC model that models the real world clocks and lets clocks preserve synchronization among them without a reference clock. Our model is the first GUC model that captures the notion of consensus on clock.

We proposed a generic synchronization protocol that works on top of a blockchain protocol. Our synchronization protocol takes advantage of a regular messaging process (e.g., blocks are sent regularly) to preserve consensus between honest parties' clocks. If blockchain protocols preserve the CD and CP properties with the clock difference $\frac{3\epsilon_{\max}+\delta^{\epsilon}_{\max}}{T}$, then the difference between clocks are bounded with $\frac{2\epsilon_{\max}+\delta^{\epsilon}_{\max}}{T}$ in the beginning of the epoch. In addition, our protocol preserves the difference between the synchronized clock and the previous clock of a party which is crucial to preserve the security of the blockchain protocols.

References

[1] Atomic second. https://en.wikipedia.org/wiki/Second.

[2] 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.

[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] 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.

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

[6] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, et al. Overview of polkadot and its design considerations. *arXiv preprint arXiv:2005.13456*, 2020.

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

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

[21] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer, 2013.

[22] 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.

[23] A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.

[24] 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.

[25] 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.

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

[27] 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.

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

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

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

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

[32] 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.

[33] 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.

[34] 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.

[35] F. Tirado-Andrés and A. Araujo. Performance of clock sources and their influence on time synchronization in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 15(9):1550147719879372, 2019.

[36] 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.

[37] 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.

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

[39] J. Wu, L. Zhang, Y. Bai, and Y. Sun. Cluster-based consensus time synchronization for wireless sensor networks. *IEEE Sensors Journal*, 15(3):1404–1413, 2014.

A Preliminaries

A.1 Universally Composable (UC) Model:

The 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 a protocol (without a trusted entity) is called UC-secure if running the protocol with the ideal functionality $\mathcal{F}$ is indistinguishable by any external environment $\mathcal{Z}$ from the protocol running in the real-world.

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 an identifier consisting of a session identifier sid and the ITM identifier pid. A party P in UC model is an ITI with the identifier $(\mathsf{sid}, \mathsf{pid})$.

π in the 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 the Ideal World: The ideal world consists of an incorruptible ITM $\mathcal{F}$ which executes π in an ideal way. The adversary $\mathcal{S}$ (called simulator) in the ideal world has ITMs which forward all messages provided by $\mathcal{Z}$ to $\mathcal{F}$. These ITMs can be considered corrupted parties and are represented as $\mathcal{F}$. 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 to [7, 8] for further details about the UC-model.

Definition A.1. *(UC-security of π) Let π be the real-world protocol and $\mathcal{F}$ be the ideal-world functionality of π. We say that 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 the 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 [9] (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 $\mathcal{G}$.