Network Time with a Consensus on Clock

Handan Kılınç Alper

Web3 Foundation

Abstract

Proof-of-stake based protocols in synchronous communication models assume that all parties possess common knowledge of the current round of the protocol. This model is typically realized in real-world applications by relying on physical clocks of nodes that advance to the next round based on a specific time interval since the previous round. These clocks are adjusted by *centralized* systems such as Network Time Protocol (NTP) to mitigate the impact of random drifts. On the other hand, an attack on these systems (which has happened in the past) can cause corruption of blockchains that rely on it. So, we are facing the dilemma of relying on the *centralized* solution to adjust our clocks or risking the security of our *decentralized* protocols. In this paper, we propose a Generalized Universal Composable (GUC) framework for the physical clock synchronization problem in the decentralized system. Additionally, we present a straightforward yet practical protocol to be implemented on a blockchain network that embodies our model. Our protocol addresses the needs of parties running in a distributed system that rely on synchronized physical clocks to maintain the accuracy and security, without relying on a centralized system. Notably, our protocol introduces no additional communication overhead to the underlying blockchain protocol.

1 Introduction

The measurement of time on computers is commonly based on the number of vibrations occurring in crystal oscillators within their clocks. For instance, a crystal oscillator may vibrate 32768 times per second, serving as the foundation for time calculation. However, these vibrations are prone to fluctuations caused by environmental factors such as temperature, pressure, and humidity. As a result, clock drifts occur, leading to inconsistencies between a computer's perception of time and the global standard. To address this issue, computer clocks connected to the Internet often rely on Network Time Protocol (NTP) to establish accurate time synchronization. By utilizing NTP, computers can maintain a cumulative drift close to zero, ensuring precise time measurements. It helps to facilitate the secure execution of protocols such as certificate validation.

Decentralized protocols, particularly proof-of-stake (PoS) based blockchain protocols, rely on synchronized communication and the accuracy of physical clocks [13, 14, 26, 32, 23, 20]. In PoS protocols, parties initiate new rounds based on the measurement of time using their computer clocks, typically synchronized with Network Time Protocol (NTP). Consequently, the security of these protocols becomes dependent on the security of centralized NTP servers. Unfortunately, the track record of NTP servers raises concerns due to various attacks and

incidents [25, 31, 28]. For instance, in a notable incident, two critical NTP servers were deliberately set back in time, causing significant disruptions to external servers relying on it [4]. If a similar attack were to occur in a PoS blockchain protocol, honest parties would halt block production under the mistaken belief that their round has not yet arrived, while malicious nodes could continue producing blocks, flooding the blockchain with maliciously-generated blocks. While some clocks can synchronize with the Global Positioning System (GPS) as an alternative to NTP, this approach also has its vulnerabilities. GPS synchronization requires additional setup and offers improved accuracy compared to NTP, without reliance on potentially compromised servers. However, GPS systems are susceptible to spoofing attacks, such as signal delays [33, 19, 29, 18, 35]. Additionally, adverse weather conditions can cause inaccuracies in GPS signal reception. These existing solutions highlight the potential security vulnerabilities associated with relying on external systems to establish accurate local clocks in decentralized protocols like blockchains. Such reliance contradicts the fundamental goal of constructing fully decentralized systems and poses a significant security risk.

In this paper, we model the problem of synchronization of (physical) clocks in a decentralized manner in the GUC model. In our model, parties possess local timers, representing crystal oscillators, and construct local clocks that progress based on a certain number of timer ticks. The frequency of timer ticks can be modified by the environment, introducing the possibility of clock drift, where local clocks deviate from each other over time, even if they were initially synchronized. To address this issue, we introduce a new functionality called the consensus clock. The consensus clock functionality takes the local clocks of participating parties as votes and then provides updated local clocks to all parties. These updated clocks are designed to be sufficiently close to each other and to the votes cast by the parties. In this way, the consensus clock functionality enables the decentralized network to align and synchronize their clocks effectively. It is important to note that our objective is not to design a new model for synchronous communication, which typically involves elaborate models [8, 20, 22]. Instead, we specifically address the synchronization of physical clocks, which have the ability to measure time, while the global clock functionalities in synchronous communication models rely on logical clocks that progress based on protocol events independent of 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 caters to scenarios where each party maintains a local clock constructed based on the ticks of their local timer and aims to align this local clock with the other clocks in the network, all without relying on a reference clock. We define a functionality which defines the rate of timers globally and another functionality for local timers which does not necessarily follow the global rate to capture the notion of drifted timers in the real world. Our functionality called consensus on clock provides synchronization. It 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 their original clocks. Our model represents a novel contribution as the first security model specifically tailored to the notion of consensus on clocks.
- We propose a basic clock synchronization protocol (BCSP) that enables parties to construct closely aligned clocks through agreement on message timestamps during the protocol. In BCSP, parties record the clock information provided in the messages and order them locally accordingly. At the end of the protocol, each party sets their new clock to be the clock at a specified order. Despite potential variations in the local order of clocks among different parties, we show that the difference between the new clocks is bounded

by the maximum network delay and clock drift experienced during the protocol. BCSP offers an effective solution for achieving clock synchronization among parties without requiring additional communication beyond the agreed-upon message timestamps.

- We construct the *Relative Time* protocol on top of blockchain protocols that realizes the consensus on clock functionality. Our protocol builds upon BCSP, where parties select the median of the sorted clock information contained in the blocks. The assumption of message agreement in BCSP is satisfied by the security guarantees provided by the underlying blockchain protocol. In our protocol, parties periodically update their clocks by running the relative time protocol, utilizing only the arrival time and round of the blocks since the last update. By leveraging the consensus mechanism inherent in the blockchain, we achieve decentralized physical clock synchronization for PoS-based blockchain protocols without imposing extra overhead on the network and the blockchain itself. Our protocol is adaptable to a variety of blockchain protocols [21, 12, 2, 11, 5] that align with our abstraction. Furthermore, our protocol can be adopted by external distributed systems for clock synchronization simply by listening to the blockchain network that utilizes our protocol. This allows these external systems to synchronize their clocks with ours without actively participating in the blockchain protocol.

1.1 Related Works

UC Clock Models: The network time model by Canetti et al. [10] is the first UC 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. In contrast, our model focuses on achieving close alignment between local clocks without relying on a specific reference clock. The objective is to ensure that the clocks of different parties are sufficiently synchronized with each other, rather than being closely aligned to a specific external reference (e.g., a clock provided by NTP or GPS). This model is particularly suitable for protocols, such as blockchain protocols, that do not require precise adherence to real-world notions of time for security and completeness.

There are UC models [8, 20, 22] designed to emulate the synchronous communication. The clock functionalities in these models are different from physical clocks since there is no notion of time measurement. They maintain the round of a protocol based on events and make sure that all parties are in the same or close rounds. In our case, however, 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 to keep the clocks of parties close enough when they drift apart. The feasibility of utilizing our consensus clock functionality to achieve the UC-synchronization model [8, 20, 22] is an intriguing question. However, exploring this possibility is beyond the scope of this paper and could be considered as a potential avenue for future research.

PoS protocols: The security of some PoS blockchain protocols [21, 12, 2, 11, 5] 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 [3] 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 generic 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, 26, 32, 23] have been extensively studied in previous work as they are an important component in distributed systems. These protocols can be categorized into two types: reference-based and distributed synchronization (consensus-based) protocols. The latter aim to achieve a common global time among distributed nodes. They are more robust against large networks although they may converge slower than other methods. The key idea behind these algorithms is to use a linear iteration, where each node updates its local estimate of the global clock value based on information received from its neighbours. As a result, all nodes in the network eventually converge to the same consensus clock.

Among the consensus-based protocols, [24] was the first to use the term "consensus clock." In this protocol, nodes periodically broadcast their clock values and update them by combining received clocks using a confidence parameter. Simultaneously, nodes iteratively compare results from the current and previous synchronization rounds to improve their skew compensation parameter. However, this algorithm lacks analysis against Byzantine nodes, which could hinder consensus convergence by sending incorrect clock values. ATS [30] is another consensus-based protocol that requires nodes to send periodic clock values to estimate relative drift. Nodes then run a consensus algorithm for drift compensation. Similar to [24], this protocol also lacks analysis against Byzantine nodes and does not support probabilistic consensus mechanisms like the longest chain rule used in blockchains. SATS [16] addresses delay and message manipulation attacks on ATS, but its adversarial model has limitations. In [17], nodes differently reach a consensus on a maximum clock value for faster conversion. However, the same problems present in previous protocols, such as lack of analysis against Byzantine nodes, persist in this approach. Additionally, using a maximum clock value might not be suitable in blockchain networks, as a malicious node could push the consensus clock too far into the future, causing honest nodes to skip many rounds. To address deception attacks, [37] introduces a Byzantine node detection mechanism to establish trusted links between nodes. It deploys a specified consensus mechanism and its attack model is not general.

Despite extensive research on clock synchronization in distributed systems, no protocol has been designed to directly deploy the consensus mechanism of a blockchain protocol for clock synchronization. Beyond this, they are lack of the formal security model and security analysis. In this paper, we close this gap by providing the first formal security model for the consensus clock and constructing a secure protocol that can be integrated with minimal changes to the node's algorithm of the blockchain protocol.

1.2 Protocol Overview

In this section, we give a high level overview of our relative time protocol. Our protocol is built upon a blockchain protocol which consists of sequential rounds. Each round is assigned to some set of parties to produce blocks. In this blockchain protocol, each block has the round information in it. We assume that parties running it have the clocks which have the difference

at most δ when the blockchain protocol starts with a genesis block. Here, δ is the maximum network delay. This difference can be simply satisfied by constructing a clock initially shows 0 just after receiving the genesis block. From that point, in every T-tick of their clock, they start a new round and run the round as specified in the blockchain protocol. Then, they update their local clock with the relative time protocol in the end of each epoch which is a notion introduced by our protocol.

In more detail, epochs mark specific intervals in the blockchain protocol. Within each epoch, parties actively participate in the blockchain protocol or just listen to the network. When they receive a valid block, they construct a candidate clock for this block in which round information r is embedded. Specifically, r is interpreted as the current clock value of the block producer's clock. Accordingly, they construct a candidate clock that currently shows r. Although the candidate clocks may not precisely match the block producer's clock due to network delays, they are considered close enough. In the end of epoch, parties retrieve the clocks associated with the *finalized* (agreed) blocks generated during that epoch. Then, they sorts them and select the clock which shows the median clock value as their new clock. Then, the new epoch starts.

We show that the difference between the new clocks of honest parties is bounded within $\delta + \epsilon_{\max}$ at the beginning of each epoch (after the clock update). Here, $\epsilon_{\max}$ is the maximum frequency error occurred between candidate clocks after their construction. We achieve this bound because honest parties consider the clocks of the same finalized blocks when selecting the median clock, resulting in their clocks being close to one another (See Theorem 3.1).

We also show that our protocol ensures that the difference between the new clock which will be used in the next epoch and the original clock used during the epoch is no more than $2\delta + \epsilon_{\max}$. This is due to the selection of the *median* clock and the assumption that more than half of the finalized clocks are generated by honest parties. These facts make the new clock range between the minimum and maximum clocks constructed from honest blocks. Since the clocks of honest parties are already within a certain bound, the new clocks, which are close to one of these honest clocks, remain close to the original clock used during the epoch. This property ensures that honest parties never skip a round after the clock update because their new clocks are close the the original clock. Thus, the security of the underlying blockchain protocol is preserved.

One advantage of our protocol is that parties do not need to actively participate in the block production mechanism of the blockchain protocol to obtain new clocks that are synchronized with the rest of the parties. External parties running another protocol requiring synchronized clocks can simply listen to the blockchain network to keep their clocks close to the others.

Remark 1: Our bounds are influenced by the drift rate of clocks during an epoch, which impacts the value of $\epsilon_{\max}$. Once a candidate clock is constructed, it progresses based on the local timer's rate for each party. Consequently, it is affected by any possible drift that occurs until the end of the epoch. As a result, even if two candidate clocks belonging to different parties initially start with the same rate, they may drift apart during the epoch. This deviation increases with the length of the epoch. In our study, we measured the clock drifts of nodes running the Polkadot relay chain protocol BABE [6] in Section 5. Our clock analysis revealed that an epoch length of 12 hours can cause a maximum frequency error of 0.75 seconds between two candidate clocks. Additionally, we examined the stability of these

nodes' timers, aiming to determine if it is feasible to establish a bound $\epsilon_{\max}$ that remains valid throughout the epochs. Our analysis shows that even if the timers do not precisely follow the same rate, they remain stable over a time interval. In other words, their frequency error does not change significantly within this interval. Therefore, it is possible to fix a bound $\epsilon_{\max}$ as the network parameter δ and determine a suitable round length for the blockchain protocol.

Remark 2: The interdependence between our relative time protocol and the underlying blockchain protocol may initially appear to create a chicken and egg problem. On one hand, our protocol relies on the security properties of the blockchain protocol, such as consensus on finalized blocks and the presence of honest blocks. On the other hand, the security of the blockchain protocol depends on the synchronization provided by our protocol. However, the solution to this apparent dilemma is straightforward. The blockchain protocol maintains its security as long as the synchronization between honest parties is ensured. In other words, an honest party should not start a new round until it receives the honest block(s) generated in the previous round. This assumption is crucial for maintaining the necessary security properties of a blockchain protocol. Our relative time protocol addresses this requirement by bounding the differences between the clocks of honest parties. Once this bound is satisfied within an epoch, the security properties of the blockchain protocol satisfied during that epoch too. As a result, at the end of the epoch, the candidate clocks are obtained from the blockchain protocol have the necessary security properties for our relative time protocol. So, the same bounds carry over to the next epoch with the new clocks selected from these candidate clocks. Thus, the parties remain synchronized throughout the next epoch. We note that our protocol assumes the initial clock differences between honest parties when the genesis block is released are bounded by the network delay parameter δ. This ensures that the synchronization among honest parties is already satisfied during the first epoch, effectively bootstrapping the security for next epochs. By providing the necessary clock synchronization, our protocol enables the underlying blockchain protocol to maintain its security guarantees, establishing a mutually beneficial relationship between the two protocols.

2 Security Model

In this section, we introduce our new GUC-model for the consensus clock. In our model, we have multiple interactive Turing machines (ITM)'s and each ITM has an inbox collecting messages from other ITMs, adversary $\mathcal{S}$ 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 the (G)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. $\mathcal{S}$ is also activated whenever the ideal functionalities are invoked. $\mathcal{S}$ can corrupt parties $P_1, P_2, ..., P_n$. If a party is corrupted, then its current state is shared with $\mathcal{S}$. See Appendix A for more detail about UC and GUC model.

2.1 Our Framework for Consensus of Clocks

In our model, we consider a decentralized network where each party has access to its own local timer, which is akin to a computer clock in real-life scenarios. These local timers are designed to tick based on a predefined global metric time. However, due to potential adversarial

interruptions, the local timers may deviate from the expected metric time. Parties use local timers to define local clocks of a protocol. In such an environment, we model how parties synchronize their local clocks. Our model consists of the following functionalities:

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

$\mathcal{G}_{\mathsf{refTimer}}$ has a fixed rate to tick.

- ***(Registration)*** If a functionality $\mathcal{F}$ sends a message $(\mathsf{Register}, \mathsf{sid}, \mathcal{F})$, $\mathcal{G}_{\mathsf{refTimer}}$ 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, $\mathcal{G}_{\mathsf{refTimer}}$ sends $(\mathsf{Tick}, \mathsf{sid}, \mathcal{S})$ and $(\mathsf{Tick}, \mathsf{sid}, \mathcal{F})$ to every $\mathcal{F} \in \mathsf{list}_{\mathcal{F}}$.

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

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

$\mathcal{G}_{\mathsf{refTimer}}$ defines a metric time. It can be considered as a global timer that ticks with respect to the metric time (e.g., second). The functionalities, who want to be notified in every tick, register with $\mathcal{G}_{\mathsf{refTimer}}$. Whenever $\mathcal{G}_{\mathsf{refTimer}}$ ticks, it informs $\mathcal{S}$ and the registered functionalities. The difference between $\mathcal{G}_{\mathsf{refClock}}$ [10] and $\mathcal{G}_{\mathsf{refTimer}}$ is that $\mathcal{G}_{\mathsf{refTimer}}$ does not have any absolute values related to time. 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. It is only accessed by functionalities.

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

We define the global functionality $\mathcal{G}^{P}_{\mathsf{timer}}$ modelling a local timer that informs its owner P whenever $\mathcal{S}$ orders to progress the timer. Thus, P gains a perception of time with respect to $\mathcal{S}$. P uses this information to construct its own clock is defined next. See Figure 2 for the details of $\mathcal{G}^{P}_{\mathsf{timer}}$. Remark that $\mathcal{G}^{P}_{\mathsf{timer}}$ ticks with a rate that may vary in time (as crystal oscillators of clocks).

We note that we define $\mathcal{G}^{P}_{\mathsf{timer}}$ in *the GUC model* instead of the UC model to capture the fact that the real world timers interact with arbitrary protocols.

Next, we define existing notions related to clocks which are constructed with respect to local timers to label the time. Differently than timers, we define absolute values for clocks which can be used to count locally the round of a protocol which progresses with time.

Definition 2.1 (Clock)**.** *A clock $\mathcal{C}_{\ell}$ runs the algorithm (See Algorithm 1) initiated by the party P_{ℓ} with the parameters: clock initial value $\bar{c}_{\ell}$ and clock current value c_{ℓ}. The clock outputs the clock value c_{ℓ} whenever $\mathcal{G}^{P_{\ell}}_{\mathsf{timer}}$ ticks.*

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

$\mathcal{G}^{P}_{\mathsf{timer}}$ is identified with its owner $P = (\mathsf{sid}, \mathsf{pid})$.

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

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

Algorithm 1 CLOCK($\bar{c}_\ell$)

1: $c_\ell = \bar{c}_\ell$
2: **while** $\mathcal{G}^{P_\ell}_{\mathsf{timer}} \rightarrow (\mathsf{Tick}, P)$ **do**
3: $\quad c_\ell = c_\ell + 1$
4: $\quad$ **output** c_ℓ

We denote a clock as a null clock when it has not been initialized or assigned a specific value. In our protocol, a null clock is represented by the notation $\boldsymbol{\mathcal{C}} = \mathsf{null}$.

Definition 2.2 (Frequency)**.** *The frequency of a clock $\boldsymbol{\mathcal{C}}_\ell$ between $c_\ell = t$ and $c_\ell = u$ that we denote by $f_\ell(u,t)$ is the number of ticks from $c_\ell = t$ till $c_\ell = u$. So, $f_\ell(u,t) = t - u$.*

Definition 2.3 (Frequency Error)**.** *The frequency error of a clock $\boldsymbol{\mathcal{C}}_\ell$ between $c_\ell = t$ and $c_\ell = u$ 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 $\mathcal{G}_{\mathsf{refTimer}}$ between $c_\ell = u$ to $c_\ell = t$.*

When $\epsilon_\ell(u,t)$ is positive, it means that $\mathcal{G}^{P_\ell}_{\mathsf{timer}}$ is faster than $\mathcal{G}_{\mathsf{refTimer}}$. Otherwise, it means that $\mathcal{G}^{P_\ell}_{\mathsf{timer}}$ is slower than $\mathcal{G}_{\mathsf{refTimer}}$. 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 $\epsilon_{\ell \triangleright k}(u_\ell, t_\ell) = |f_\ell(u_\ell, t_\ell) - f_k(u_k, t_k)|$. Since $f_{\mathsf{ref}}(u_\ell, t_\ell) = f_{\mathsf{ref}}(u_k, t_k)$, $\epsilon_{\ell \triangleright k}(u_\ell, t_\ell) = |f_\ell(u_\ell, t_\ell) - f_k(u_k, t_k)| = |f_\ell(u_\ell, t_\ell) - f_{\mathsf{ref}}(u_\ell, t_\ell) - f(u_k, t_k) + f_{\mathsf{ref}}(u_k, t_k)| = |\epsilon(u_\ell, t_\ell) - \epsilon(u_k, t_k)|$.

Definition 2.4 (Clock Difference)**.** *Given two clocks $\boldsymbol{\mathcal{C}}_k$ and $\boldsymbol{\mathcal{C}}_\ell$, the time difference of clocks $\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_\ell$ is defined as $|c_k - c_\ell|$ where c_k and c_ℓ are the clock values of $\boldsymbol{\mathcal{C}}_k$ and $\boldsymbol{\mathcal{C}}_\ell$, respectively, when the difference is measured.*

Definition 2.5 (Clock Stability)**.** *A clock $\boldsymbol{\mathcal{C}}_k$ is T-stable if there exists ϵ_1, ϵ_2 such that $\epsilon(c_k, c_k + T) \in [\epsilon_k^1, \epsilon_k^2]$ for all $c_k \geq \bar{c}_k$.*

In real-world applications, it is often assumed that computer clocks exhibit stability [27]. Stability refers to the property of a clock's rate being consistent over time, meaning that it does not change significantly within a given period. While stable clocks are desirable, it is important to note that stability alone does not guarantee the accuracy of clocks.

We remark that the frequency error of two T-stable clocks $\boldsymbol{\mathcal{C}}_k$ and $\boldsymbol{\mathcal{C}}_k$ when $\boldsymbol{\mathcal{C}}_k = c_k$ and when $\boldsymbol{\mathcal{C}}_k = c_k + nT$ is $\epsilon_{k \triangleright \ell} \leq |n \max\{(\epsilon_k^2 - \epsilon_\ell^1), (\epsilon_\ell^2 - \epsilon_k^1)\}|$. This shows that even if clocks are stable, they drift apart in time because they are not accurate.

Functionality $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$

$\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ is parametrized with $\Theta = (\Theta_c, \Theta_p)$. It maintains a set of ITI's $P_i = (\mathsf{pid}_i, \mathsf{sid})$ in $\mathcal{P}_h$ dynamically i.e., $\mathcal{P}_h$ is initially empty and it is populated through registration.

- (***Registration:***) Upon receiving a message $(\mathsf{register}, P_i)$, $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ sends $(\mathsf{Registered}, P_i)$ to $\mathcal{S}$ and P_i. Then, $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ sends $(\mathsf{Register}, \mathcal{F}^{\Theta}_{\mathsf{Cclock}}, P_i)$ to $\mathcal{G}^{P_i}_{\mathsf{timer}}$ and receives back a message $(\mathsf{Registered}, \mathcal{F}^{\Theta}_{\mathsf{Cclock}}, P_i)$ from $\mathcal{G}^{P_i}_{\mathsf{timer}}$.
- ***(Consensus)*** Upon receiving a message $(\mathsf{Voting}, P_i, \boldsymbol{\mathcal{C}}_i)$ from each $P_i \in \mathcal{P}_h$, $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ runs ConsensusCheck$(\{\boldsymbol{\mathcal{C}}_i\}_{P_i \in \mathcal{P}_h}) \rightarrow b$ and sends $(\mathsf{Voting}, b, \{P_i, \boldsymbol{\mathcal{C}}_i\}_{P_i \in \mathcal{P}_h})$ to $\mathcal{S}$.

  $\mathcal{S}$ replies with $(\mathsf{NewClock}, \mathsf{sid}, \mathcal{F}^{\Theta}_{\mathsf{Cclock}}, \{\tilde{\boldsymbol{\mathcal{C}}}_j\}_{(\mathsf{pid}_j, \mathsf{sid}) \in \mathcal{P}_h})$ where:

  - $|\boldsymbol{\mathcal{C}}_j - \tilde{\boldsymbol{\mathcal{C}}}_j| \leq \Theta_c$ for all $\boldsymbol{\mathcal{C}}_j \neq \mathsf{null}$ if $b = 1$ and
  - $|\tilde{\boldsymbol{\mathcal{C}}}_i - \tilde{\boldsymbol{\mathcal{C}}}_j| \leq \Theta_p$ for all pairs $P_i, P_j \in \mathcal{P}_h$.

  In the end, $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ sends $(\mathsf{Consensus}, \mathsf{sid}, P_j, b, \tilde{\boldsymbol{\mathcal{C}}}_j)$ to each party $P_j \in \mathcal{P}_h$.

Figure 3: The functionality $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$

2.1.3 Consensus on Clock ($\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$)

We construct $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ in Figure 3 which helps parties to shorten the distance between their clocks when they drift apart. We define $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ with the parameter $\Theta = (\Theta_c, \Theta_p)$ where Θ_c is a parameter to determine if the consensus is possible and Θ_p is the parameter that defines the maximum clock difference allowed in $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$. We denote the set of honest parties by $\mathcal{P}_h$. $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ works as follows:

(**Registration:**) This phase is executed by an honest party P_i that wants to join the protocol. After $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ receives a registration message from P_i, it informs $\mathcal{S}$ about it. In addition, it registers itself to P_i's local timer functionality $\mathcal{G}^{P_i}_{\mathsf{timer}}$ to construct the clock of P_i in later phases.

(Consensus): This phase starts by receiving all clocks of honest parties as a vote. Once all the clocks have been received, the $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ functionality performs a consensus check to determine if consensus is possible with these clocks.

The consensus check is implemented through the ConsensusCheck algorithm, which takes the clocks as input and returns $b \in \{0, 1\}$ indicating whether consensus is possible. The purpose of the consensus check is to verify if it is feasible to construct a new clock $\bar{\boldsymbol{\mathcal{C}}}_j$ for each party P_j that satisfies certain criteria. Specifically, the check ensures that if there exists a new clock $\bar{\boldsymbol{\mathcal{C}}}_j$ which is sufficiently close to the new clocks of other parties (within a distance of Θ_p), and that it remains within a certain distance Θ_c of the party's original clock $\boldsymbol{\mathcal{C}}_j$ (i.e., $\boldsymbol{\mathcal{C}}_j - \bar{\boldsymbol{\mathcal{C}}}_j \leq \Theta_c$) unless $\boldsymbol{\mathcal{C}}_j = \mathsf{null}$. By performing this consensus check, $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ determines whether the received clocks allow for the construction of new clocks that align closely with each other while still maintaining a reasonable deviation from the original clocks.

After the consensus check, $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ sends all clocks of honest parties and b to $\mathcal{S}$ and $\mathcal{S}$ returns new clocks for each party. These new clocks are within the distance of Θ_p and does not deviate from the original clocks more than Θ_c if the consensus is possible i.e., $b = 1$. In the end, $\mathcal{F}^{\Theta}_{\mathsf{Cclock}}$ sends each new clock $\tilde{C}_i$ to its owner P_i and sends the value of b to indicate

whether consensus was possible or not.

In summary, the goal of this functionality is to provide honest parties with clocks that are close to their initial clocks and do not deviate significantly from them. The bound Θ_c ensures that the new clocks do not drift too far from the initial clocks, which is important for protocols relying on the output of $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$. It prevents situations where the new clocks show significantly different times compared to the initial clocks, which could potentially disrupt the normal execution of protocols. If a protocol does not have such concerns and allows for larger clock deviations from the original clock, the value of Θ_c can be considered as infinite (∞).

2.2 UC- Partially Synchronous Network Model

Our network model $\mathcal{F}_{\mathsf{com}}$ is similar to the partially synchronous network model [12, 21, 15] deployed most of the PoS-blockchain protocol. Differently, our functionality accesses to $\mathcal{G}_{\mathsf{refTimer}}$ in order to have the notion of time. Now, we define the functionality $\mathcal{F}_{\mathsf{com}}$ which models a partially synchronous network with the time delay δ. Here, δ represents number of δ-increment message by $\mathcal{G}_{\mathsf{refTimer}}$.

$\mathcal{F}_{\mathsf{com}}^{\delta}$: Each party P_i has access to its inbox populated by $\mathcal{F}_{\mathsf{com}}^{\delta}$. $\mathcal{F}_{\mathsf{com}}^{\delta}$ first registers to $\mathcal{G}_{\mathsf{refTimer}}$ and creates a local timer to measure the real time. Whenever $\mathcal{G}_{\mathsf{refTimer}}$ sends a message with Tick, it increments it. When P_i sends a message, $\mathcal{F}_{\mathsf{com}}^{\delta}$ sends it to $\mathcal{S}$ and starts to measure the time until receiving message from $\mathcal{S}$ that says 'send' or until δ ticks passed since sending the message to $\mathcal{S}$. In either case, $\mathcal{F}_{\mathsf{com}}^{\delta}$ puts it to the inbox of all parties. This model guarantees that every honest message arrives to all parties at most δ ticks later than the time it is sent.

We note that the perception δ can be different for each party P_k. In other words, if a message sent when a clock of P_k shows u and received when the clock shows t, then $f_k(u,t) \leq \delta + \epsilon_k(u,t)$ since $0 \leq f_{\mathsf{ref}}(u,t) \leq \delta$. In our analysis, we make the assumption that $\epsilon_k(u,t)$ is very close to zero for the sake of clarity and simplicity. This assumption is reasonable because the value of δ is typically small in real-world protocols, and the impact of the clock drift rate on the overall time difference is negligible.

3 Basic Clock Synchronization Protocol

In this section, we describe the Basic Clock Synchronization Protocol (BCSP) within the hybrid model $\mathcal{G}_{\mathsf{timer}}^{P}$ and $\mathcal{F}_{\mathsf{com}}^{\delta}$. BCSP is a protocol designed to synchronize clocks among multiple parties. The protocol ensures that the newly obtained clocks maintaining a limited distance from other newly acquired clocks. This distance bound depends on parameters such as network delay and the frequency error of the clocks during the protocol execution. Our relative time protocol, implemented on a blockchain, is based on the BCSP.

Consider m honest ITI's $P_1, ..., P_m$ initiated with integer parameters $n \geq 1$, $0 < a \leq n$ and an initial clock value $\bar{c}_k$ for each party P_k given by $\mathcal{Z}$. After the initiation, each party P_k constructs a clock $\boldsymbol{\mathcal{C}}_k$ that runs $\textsc{Clock}(\bar{c}_k)$. In this protocol, whenever $\mathcal{Z}$ requests to P_k to send a message, P_k obtains the current clock value $c_i \leftarrow \textsc{Clock}(\bar{c}_k)$ without any delay after the request and sends a message B_i including c_i with $\mathcal{F}_{\mathsf{com}}^{\delta}$. When an ITI P_ℓ receives a message B_i, it constructs a clock $\boldsymbol{\mathcal{C}}_{i^\ell}$ that runs $\textsc{Clock}(c_i)$ After receiving n messages, each party P_ℓ obtains the clock values $c_h^\ell \leftarrow \boldsymbol{\mathcal{C}}_{h^\ell}$ for all $h \in [1,n]$. It adds them to a list list_ℓ and sorts it. In the end, it obtains a^{th} clock value c_j^ℓ of sorted list which is the output of a clock

$\boldsymbol{\mathcal{C}}_{j^\ell}$. In the end, P_ℓ outputs its new clock $\tilde{\boldsymbol{\mathcal{C}}}_\ell = \boldsymbol{\mathcal{C}}_{j^\ell}$.

The critical assumption in BCSP is that $\mathcal{F}_{\mathsf{com}}^{\delta}$ relay exactly n messages during the protocol. This means that all parties receive the same n messages in the end of the protocol. We have this artificial assumption to have a simpler analysis. We later realize this assumption in our protocol by relying on a consensus mechanism that agrees on n messages. The important take away from BCSP as shown in our analysis is that even if the initial clock values are very different from each other or some timestamps are not correct or arrival times of messages are different for each party, all parties obtain a clock which are close to each other.

Theorem 3.1. *Assume that $\epsilon_{h^k \rhd h^\ell}(c_h, c_h^k) \leq \epsilon_{\max}$ for all $k, \ell \in [1, m]$ and $h \in [1, n]$. Then the clock difference of new clocks $\tilde{\boldsymbol{\mathcal{C}}}_k$ and $\tilde{\boldsymbol{\mathcal{C}}}_\ell$ in the end of BCSP for all $k, \ell \in [1, m]$ is at most $\delta + \epsilon_{\max}$.*

Proof. If a party P_y received the message B_h without any delay and no frequency error occurred on P_y's timer after receiving it, then $\boldsymbol{\mathcal{C}}_{h^y}$ would show $\tilde{c}_h = c_h + f_{\mathsf{ref}}(c_h, \tilde{c}_h)$ in the end of BCSP. Considering the network delay and the frequency error after the creation of $\boldsymbol{\mathcal{C}}_{h^y}$, we can deduce that $c_h^y = \tilde{c}_h + \delta_{h \to y} + \epsilon_y(c_h, c_h^y)$. Therefore,

$$\tilde{c}_h + \epsilon_y(c_h, c_h^y) \leq c_h^y \leq \tilde{c}_h + \delta + \epsilon_y(c_h, c_h^y) \tag{1}$$

for all $y \in [1, m]$ and $h \in [1, n]$. Here, $\delta_{h \to y}$ is the network delay when forwarding B_h to P_y.

Now, let's analyse the clock differences of any two clocks $\tilde{\boldsymbol{\mathcal{C}}}_k$ and $\tilde{\boldsymbol{\mathcal{C}}}_\ell$ in the end of BCSP for all $k, \ell \in [1, m]$. Let's assume that $\mathtt{sort}(\mathsf{list}_k)[a] = c_i^k$ and $\mathtt{sort}(\mathsf{list}_\ell)[a] = c_j^\ell$. So, we need to upper bound $|c_i^k - c_j^k|$ to find out the clock differences of $\tilde{\boldsymbol{\mathcal{C}}}_k$ and $\tilde{\boldsymbol{\mathcal{C}}}_\ell$. We have the following cases:

1. $c_j^k \leq c_i^k$ and $c_i^\ell \leq c_j^\ell$: Using the the facts $c_i^\ell - c_j^\ell \leq 0, c_j^k - c_i^k \leq 0$ and Equation (1), we obtain $|c_i^k - c_j^\ell| \leq \delta + \epsilon_{\max}$.

2. $c_i^k \leq c_j^k$ and $c_j^\ell \leq c_i^\ell$: Using the the facts $c_i^k - c_j^k \leq 0, c_j^\ell - c_i^\ell \leq 0$ and Equation (1), we obtain $|c_i^k - c_j^\ell| \leq \delta + \epsilon_{\max}$.

3. $c_j^k \leq c_i^k$ and $c_j^\ell \leq c_i^\ell$: In this case there are $n - a$ more elements in $\mathtt{sort}(\mathsf{list}_k)$ after c_i^k and $n - a$ more elements in $\mathtt{sort}(\mathsf{list}_\ell)$ after c_j^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. $c_i^\ell \in \mathcal{A}_\ell$ but $c_i^k \notin \mathcal{A}_k$ and $|\mathcal{A}_k| = |\mathcal{A}_\ell|$. Therefore, there must exit x such that $c_x^k \in \mathcal{A}_k$ but $c_x^\ell \notin \mathcal{A}_\ell$. This implies that $c_j^k \leq c_i^k \leq c_x^k$ and $c_x^\ell \leq c_j^\ell \leq c_i^\ell$. Then, we obtain

   - if $c_i^k \geq c_j^\ell$, $|c_i^k - c_j^\ell| = c_i^k - c_j^\ell \leq c_x^k - c_x^\ell \leq \delta + \epsilon_{\max}$.
   - if $c_i^k \leq c_j^\ell$, $|c_i^k - c_j^\ell| = c_j^\ell - c_i^k \leq c_i^\ell - c_i^k \leq \delta + \epsilon_{\max}$.

4. $c_i^k \leq c_j^k$ and $c_i^\ell \leq c_i^\ell$: As in Case 3, we obtain $|c_i^k - c_j^\ell| \leq \delta + \epsilon_{\max}$.

□

4 Realization of Consensus on Clock

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

A blockchain protocol is an interactive protocol between parties to construct a blockchain. Garay et al. [15] define some properties to obtain a secure blockchain protocol. In these definitions, the blockchain protocol follows a clock that increments the rounds r_i in the synchronous communication model [8, 20, 22]. Below, we give the minimum properties that we need from a blockchain protocol to deploy our relative time protocol that realizes $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$.

Definition 4.1 (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 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 define an algorithm $\textsc{FinalBlocks}(\boldsymbol{B}, r_u, r_v)$ that retrieves all finalized blocks of $\boldsymbol{B}$ generated between round r_u and r_v if they exist. If a blockchain protocol has the CP property, $\textsc{FinalBlocks}(\boldsymbol{B}, r_u, r_v)$ obtains first $\boldsymbol{B}^{\lceil k}$ and returns all blocks that are generated between round r_u and r_v if the round of the last block of $\boldsymbol{B}^{\lceil k}$ is greater than or equal to r_v.

We modify the chain quality property (CQ) by Garay et al. and define the chain density (CD) property. Our definition makes sure that any set of finalized blocks which has size greater than or equal to s_{cd} includes more honest blocks.

Definition 4.2 (Chain Density (CD) Property)**.** *The CD property with parameters $s_{cd} \in \mathbb{N}$ ensures that for all $\boldsymbol{B}$ possessed by an honest party and for all r_u, r_v where $r_v \geq r_u + s_{cd}$ if $\textsc{FinalBlocks}(\boldsymbol{B}, r_u, r_v)$ outputs a set $\mathcal{B}_f \neq \emptyset$ then more than half of the blocks in $\mathcal{B}_f$ are generated by honest parties.*

The next property is a critical property to have the liveness property on the blockchain protocol. It guarantees a minimum rate of blockchain growth, which is essential for the protocol to continue making progress and adding new blocks to the chain.

Definition 4.3 (Chain Growth (CG) property [15])**.** *The CG property with parameter $\tau \in (0, 1]$ and $s_{cg} \in \mathbb{N}$ ensures that for all $\boldsymbol{B}$ possessed by an honest party at round r and for all $r_u, r_v \leq r$ where $r_v \geq r_u + s_{cg}$, the number of blocks in $\boldsymbol{B}$ generated for the rounds between r_u and r_v is greater than or equal to τs_{cg}.*

The General Structure of a Blockchain Protocol: We present a high-level blockchain protocol, denoted as $\pi_{\mathcal{H}}$, which operates in the hybrid model with the functionality $\mathcal{F}_{\mathsf{com}}^{\delta}$. This protocol serves as the underlying framework for running our relative time protocol. By doing so, we highlight the minimal requirements that a blockchain protocol must fulfill in order to accommodate our protocol.

The protocol $\pi_{\mathcal{H}}$, outlined in Algorithm 2, is structured into rounds. In each round, a set of parties is selected as block producers who generate and transmit their respective blocks

for that particular round. Each block includes the round number associated with it. Parties determine the round of the protocol from their individual local physical clocks. Each round takes T-clock ticks.

When a new party P_i joins the protocol, it first executes an initialization algorithm, Initialization, to acquire the current protocol state. The blockchain state encompasses all the necessary information (e.g., existing blocks) required for active participation in $\pi_{\mathcal{H}}$. If the state corresponds to the genesis block B_0, P_i constructs its initial clock $\boldsymbol{\mathcal{C}}_i$ by executing the function Clock(0) upon receiving the genesis block from $\mathcal{F}_{\mathsf{com}}^{\delta}$. Otherwise, P_i constructs its clock based on the state information specific to the underlying blockchain protocol. After joining the protocol, P_i periodically checks whether it is its turn to produce a block using the algorithm IsMyRound in every T ticks of its local clock. Simultaneously, if P_i receives a block, it updates its state using the algorithm UpdateState. It should be noted that the specific algorithms Initialization, IsMyRound, UpdateState, and GenerateBlock are tailored to the respective blockchain protocol. The inner workings of these algorithms are unrelated to our relative time protocol and, thus, not elaborated upon in this context.

Clocks play a critical role in PoS blockchain protocols, as their security analysis relies on the synchronization assumption that all honest parties receive the honest blocks generated in the previous round when they start a new round. This assumption is crucial to prevent network partitioning and ensure that honest parties eventually reach consensus. To satisfy this assumption, it is important to ensure that the length of the round T in $\pi_{\mathcal{H}}$ is longer than δ. This ensures that even if all clocks progress synchronously, meaning $\boldsymbol{\mathcal{C}}_i - \boldsymbol{\mathcal{C}}_j = 0$ for all parties P_i and P_j, an honest party has enough time to receive all the honest blocks from the previous rounds before starting a new round.

It is evident that if parties use their clocks as described in Algorithm 3 throughout the protocol without modifying it, they may develop vastly different perceptions of time, thereby compromising the security of the blockchain protocol. Hence, we show next how parties update their clocks using $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ to prevent excessive divergence among their clocks.

Algorithm 2 $P_k(\mathsf{sid}, \mathsf{pid})$ in $\pi_{\mathcal{H}}$

1: **run** Initialization(P_k)
2: state $\leftarrow \mathcal{F}_{\mathsf{com}}^{\delta}$
3: **if** state $= B_0$ **then**
4: **let** $\boldsymbol{\mathcal{C}}_k$ run Clock(0)
5: registered $=$ true
6: **while** P_k.registered $=$ true **do**
7: $c \leftarrow \boldsymbol{\mathcal{C}}_k, r \leftarrow \frac{c}{T}$
8: **if** $c \neq$ null **and** IsMyRound(state) **then**
9: $B \leftarrow$GenerateBlock(state, r)
10: **send** (B, P_k) to $\mathcal{F}_{\mathsf{com}}^{\delta}$
11: **if** valid B_i **received from** $\mathcal{F}_{\mathsf{com}}^{\delta}$ **then**
12: state $\leftarrow$ Update(state, B_i)

$\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-Hybrid Model: The deployment of the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ functionality in the $\pi_{\mathcal{H}}$ blockchain protocol, as described in Algorithm 3, is a straightforward. If the current state of the protocol is in the genesis state, parties construct their clocks as specified in $\pi_{\mathcal{H}}$. Otherwise, they initialize their clocks to a null value. Once they have learned the state of the protocol, parties register with the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ functionality to obtain a new clock if they currently have a null clock and to ensure their clocks remain synchronized with the rest of the network later on. We introduce a new parameter, denoted as ℓ_e, which represents the length of the epoch during which the clock remains unchanged and without any update. Whenever an epoch e ends i.e., the IsNewEpoch algorithm returns true, parties send their current clocks to the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ functionality. Then, $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ sends new clocks indicating that the consensus on

clock achieved in return. After receiving them, parties update their current clocks with the provided values if the consensus on clocks achieved i.e., $b = 1$. Otherwise, they abort and do not continue to run $\pi_{\mathcal{H}}$ because the new clocks does not maintain the security of $\pi_{\mathcal{H}}$.

One critical issue in this process is determining the appropriate time to update the clocks. If the interval between clock updates is too long, the clocks of honest parties may already have deviated significantly, making it difficult to achieve consensus on the new clock values. Therefore, it is crucial to select an appropriate interval that balances the need for clock synchronization without introducing excessive delays that could compromise the security of the protocol.

Algorithm 3 $P_k(\mathsf{sid}, \mathsf{pid})$ in $\pi_{\mathcal{H}}$ in $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model

1: **run** INITIALIZATION(P_k)
2: state $\leftarrow \mathcal{F}_{\mathsf{com}}^{\delta}$
3: **if** state $= B_0$ **then**
4: **let** $\boldsymbol{\mathcal{C}}_k$ run CLOCK(0)
5: $e = 0, r_e = 0$
6: **else**
7: $\boldsymbol{\mathcal{C}}_k = \mathsf{null}$
8: $e, r_e \leftarrow$ GETEPOCH(state)
9: registered $=$ true
10: **send** $(\mathsf{Register}, P_k)$ to $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$
11: init $\leftarrow$ state
12: **while** P_k.registered $=$ true **do**
13: $c \leftarrow \boldsymbol{\mathcal{C}}_k, r \leftarrow \frac{c}{T}$
14: **if** $c \neq \mathsf{null}$ **and** ISMYROUND(state) **then**
15: $B \leftarrow$GENERATEBLOCK(state, r)
16: **send** (B, P_k) to $\mathcal{F}_{\mathsf{com}}^{\delta}$
17: **if** valid B_i **received from** $\mathcal{F}_{\mathsf{com}}^{\delta}$ **then**
18: state $\leftarrow$ UPDATE(state, B)
19: **if** ISNEWEPOCH(state) **then**
20: **send** $(\mathsf{Voting}, P_k, \boldsymbol{\mathcal{C}}_k)$ **to** $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$
21: **receive** $(\mathsf{Consensus}, \mathsf{sid}, b, \tilde{\boldsymbol{\mathcal{C}}}_j)$ **from** $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$
22: **if** $b = 0$ **then**
23: ABORT
24: $\boldsymbol{\mathcal{C}}_k \leftarrow \tilde{\boldsymbol{\mathcal{C}}}_k$
25: $e \leftarrow e + 1, r_e \leftarrow r$

Lemma 4.1. *Assume that $\pi_{\mathcal{H}}$ is a secure blockchain protocol under the following synchronization assumption: all honest parties receive the honest blocks generated in the previous round when they start a new round. Then, $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model is secure given that $T \geq \delta + \Theta_p + \epsilon_{\max}$ and $\Theta_c < T$ and clocks of honest parties T-stable (Definition 2.5). Here, $\epsilon_{k \triangleright \ell}(c, c + T) \leq \epsilon_T$ for all P_k, P_ℓ and $\epsilon_{\max} = \ell_e \epsilon_T$ where ℓ_e is the epoch length in terms of round.*

Proof. $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model does not change the general structure of $\pi_{\mathcal{H}}$. The only change is updating the local clocks time to time through clocks provided by $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ and aborting if consensus on clock is not possible. Therefore, we need to make sure that these changes do not change the behaviour of honest parties and preserves the synchronization between honest parties so that $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model still preserves the security of $\pi_{\mathcal{H}}$.

We show that the possible behaviour changes of an honest party in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model different than the honest party running $\pi_{\mathcal{H}}$. One possible change of behaviour of an honest party in $\pi_{\mathcal{H}}$ in $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model is that either not receiving the blocks of previous rounds before starting a new round because of a different perception of rounds between parties or skipping some rounds. We now analyse they are not possible as long as $T > \delta + \Theta_p + \epsilon_{\max}$ and $\Theta_c < T$.

Assume that a clock difference of two parties P_i and P_j at a round r is Δ and $\boldsymbol{\mathcal{C}}_i \geq \boldsymbol{\mathcal{C}}_j$ in any epoch. This means that P_i starts a new round $r + 1$ before P_j. When P_j sends a block when starting the round $r = \frac{c}{T}$, the block arrives P_i at latest when P_j's clock shows $rT + \delta$. Since the difference between P_i's and P_j's clocks is Δ, when P_i receives P_j's block, $\boldsymbol{\mathcal{C}}_i$ shows

$rT + \delta + \Delta$. We need to show that $\delta + \Delta < T$ in any round of an epoch to guarantee that P_i receives the block before starting the round $r + 1$.

After receiving the genesis block in $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model, each party constructs their clocks starting from 0. Because of the network delay, the difference between their clocks is at most δ when they construct their initial clocks. Then, they do not touch their clocks till the next epoch. Given that the clocks are stable, the clock difference of a clock $\boldsymbol{\mathcal{C}}_k$ and $\boldsymbol{\mathcal{C}}_\ell$ at a round r in epoch $e = 0$ is at most $\Delta_0 = \delta + \epsilon_{k \triangleright \ell}(0, rT)$ where $\epsilon_{k \triangleright \ell}(0, rT) \leq r\epsilon_T \leq n_e \epsilon_T \leq \epsilon_{\max}$. Since $\delta + \Delta_0 < T$, even if clocks drift apart during the epoch, each honest block arrives to other honest parties before they move to the next round.

Now we look at the epochs after the first epoch $e \geq 1$ which starts at round $\frac{c_e}{T}$ and ends at round $\frac{c_{e+1}}{T}$. In the end of each epoch, parties update their clocks with the clocks provided by $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$. The clocks given by $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ are within the distance Θ_p. Therefore, during the other epochs, the difference of clocks $\boldsymbol{\mathcal{C}}_k$ and $\boldsymbol{\mathcal{C}}_j$ at any round r where $\frac{c_e}{T} \leq r < \frac{c_{e+1}}{T}$ is at most $\Delta_e \leq \Theta_p + \epsilon_{k \triangleright \ell}(c_e, rT) \leq \Theta_p + \epsilon_{\max}$. Therefore, parties maintain the fact that $\delta + \Delta_e < T$ during an epoch.

We have shown that maintaining the clock with $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ does not break the assumption that parties starts a new round after receiving all blocks of previous rounds. Now, we need to show that an honest party never skips a round due to the clock change i.e., it checks for every round whether it is its round via IsMyRound. In more detail, when starting a new epoch, since each honest party deploys a new clock, there is a possibility that a party misses a round because the new clock indicates a round in future according to the old clock. However, $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ ensures that the difference between the old and new clocks is not more than Θ_c. Therefore, if a party P_k's old clock shows c_k, where $rT \leq c_k < (r+1)T$, when an epoch changes, the new clock that P_k obtains from $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ shows $\tilde{c}_k$, where $rT - \Theta_c \leq \tilde{c}_k < (r+1)T + \Theta_c$. Since $\Theta_c < T$, it follows that $(r-1)T < \tilde{c}_k < (r+2)T$. Consequently, the new clock indicates a round r', where $r - 1 \leq r' \leq r + 1$. As a result, honest parties do not skip any rounds after the clock update, ensuring that they consistently check for each round using the IsMyRound algorithm [1].

As it can be seen above analysis, the consensus on clock is always possible because the new epoch starts before the clocks are not drifted apart a lot. Therefore, there exists a new clock for each party which is close to the given clock and close to other new clocks of parties. Specifically, $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ guarantees that the difference between new clocks of parties is at most Θ_p in the beginning of each epoch. Therefore, parties never aborts. □ □

We next replace $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ with our relative time protocol and show that $\pi_{\mathcal{H}}$ running the relative time protocol to update the clocks realizes $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model. Thus, it inherits the security of $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model.

Relative Time Protocol Algorithm 4 demonstrates the integration of our relative time protocol into ϕ. The gray lines represent the existing structure of the blockchain protocol (Algorithm 3), while the black lines represent the additional steps introduced by our relative time protocol. Therefore, parties do not need to modify the generic structure of the blockchain protocol; they only need to execute extra steps to update their clocks. In more detail, after running Initialization, party P_k follows these steps: If the current state is not the genesis state, P_k sets its clock $\boldsymbol{\mathcal{C}}_k$ to null. This null clock converges later to the consensus clock.

[1] If $r' = r$ or $r' = r - 1$, P_k may recheck if r' is its round and may equivocate in r'. Some PoS-blockchain does not allow this. If this is an issue in the blockchain protocol design, then this can be prevented by checking whether a block already is published for this round after the clock update.

Algorithm 4 $P_k(\mathsf{sid}, \mathsf{pid})$ in $\pi_{\mathcal{H}}$ running the relative time protocol

1: **run** INITIALIZATION(P_k)
2: state $\leftarrow \mathcal{F}_{\mathsf{com}}^{\delta}$
3: **if** state $= B_0$ **then**
4: **let** $\boldsymbol{\mathcal{C}}_k$ run CLOCK(0)
5: $e, r_e = 0$
6: **else**
7: $\boldsymbol{\mathcal{C}}_k = \mathsf{null}$
8: $e, r_e \leftarrow$ GETEPOCH(state)
9: registered $=$ true
10: init $\leftarrow$ state
11: **while** P_k.registered $=$ true **do**
12: $c \leftarrow \boldsymbol{\mathcal{C}}_k, r \leftarrow \frac{c}{T}$
13: **if** $c \neq \mathsf{null}$ **and** ISMYROUND(state) **then**
14: $B \leftarrow$GENERATEBLOCK(state, $\frac{c}{T}$)
15: **send** (B, P_k) to $\mathcal{F}_{\mathsf{com}}^{\delta}$
16: **if** valid B_i **received from** $\mathcal{F}_{\mathsf{com}}^{\delta}$ **then**
17: $r' \leftarrow$ ROUND(B_i)
18: let $\boldsymbol{\mathcal{C}}_{i^k}$ run CLOCK($r'T$)
19: state $\leftarrow$ UPDATE(state, B)
20: **if** ISNEWEPOCH(r_e, state) **then**
21: FINALBLOCKS($\boldsymbol{B}, r_e, r$) $\rightarrow \mathcal{B}_f$
22: **for all** $B_i \in \mathcal{B}_f$ **do**
23: list $= \{c_{i^k} \leftarrow \boldsymbol{\mathcal{C}}_{i^k}\}$
24: $c_{j^k} \leftarrow$ MEDIAN(list)
25: $\boldsymbol{\mathcal{C}}_k \leftarrow \boldsymbol{\mathcal{C}}_{j^k}$
26: $e = e + 1, r_e = r$

Otherwise, P_k constructs a clock $\boldsymbol{\mathcal{C}}_k$ that runs CLOCK(0) and sets epoch $= 0$. Regardless of the state, P_k continues with the protocol as follows:

When P_k receives a valid block B with round r' from $\mathcal{F}_{\mathsf{com}}^{\delta}$, it constructs a clock $\boldsymbol{\mathcal{C}}_{i^k}$ that runs CLOCK(c_i) where $c_i = r'T$. We note that $\boldsymbol{\mathcal{C}}_{i^k}$ is a very close approximation of the clock of the block producer of B if the block producer is honest i.e, the block producer adds the correct round information to B.

At the end of each round, P_k checks if a new epoch starts by using the ISNEWEPOCH algorithm. ISNEWEPOCH returns true in every ℓ_e rounds if P_k joined to the protocol at the first round r_e of the current epoch e. If ISNEWEPOCH returns true, P_k obtains all candidate clocks constructed between rounds r_e and $r_e + s_{cd}$ where $s_{cd} \leq \ell_e$. Then, similar to the BCSP in Section 3, P_k obtains the clock values of these clocks and sorts them. Finally, P_k selects the median clock value $c_{j\rightarrow k}$ from the sorted clock values and sets its new clock to $\boldsymbol{\mathcal{C}}_{j\rightarrow k}$, which outputs $c_{j\rightarrow k}$.

We next show that $\pi_{\mathcal{H}}$ with the relative time protocol (Algorithm 4) and $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ hybrid model are indistinguishable by $\mathcal{Z}$ where $\Theta_p = \delta + \epsilon_{\max}$ and $\Theta_c = \Theta_p + \delta$. This means that a blockchain protocol $\pi_{\mathcal{H}}$ can deploy the relative time protocol, maintaining the same security properties as shown in Lemma 4.1. Thus, any blockchain protocol having the similar structure of $\pi_{\mathcal{H}}$ can modify the nodes' algorithm as in Algorithm 4 and set $T \geq \delta + \Theta_p + \epsilon_{\max} = 2\delta + 2\epsilon_{\max}$, then achieve the synchronization between nodes without relying on centralized clock adjustment mechanisms.

Theorem 4.2. *Assuming that $\pi_{\mathcal{H}}$ has the common prefix property with the parameter k, the chain density property with the parameter s_{cd}, the chain growth property with the parameters τ and $s_{cg} \leq \ell_e - s_{cd}$, $\tau(\ell_e - s_{cd}) \geq k$ and clocks are T-stable, $\pi_{\mathcal{H}}$ with the relative time protocol realizes $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model where $\Theta_p = \delta + \epsilon_{\max}$ and $\Theta_c = \Theta_p + \delta$ except with the probability $p_{cp} + p_{cd} + p_{cg}$ which are the probability of breaking CP, CD and CG properties in $\pi_{\mathcal{H}}$, respectively. Here, $\epsilon_{k \triangleright \ell}(c, c + T) \leq \epsilon_T$ for all P_k, P_ℓ and $\epsilon_{\max} = \ell_e \epsilon_T$.*

Proof Sketch: The proof of Theorem 4.2 is in Appendix B. Here, we provide a high level idea of our security proof. We show that if the blockchain preserves CD, CP and CG properties during an epoch of $\pi_{\mathcal{H}}$ with the relative time algorithm, the relative time algorithm generates clocks for honest parties which are within the bound Θ_p and the difference between the new

clock and the clock used during the epoch is not more than Θ_c as $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model.

During the first epoch ($e = 0$), parties in $\pi_{\mathcal{H}}$ with the relative time protocol and $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model are identical. So, CP, CD and CG properties are satisfied in $\pi_{\mathcal{H}}$ with the relative time protocol during the initial epoch. Given this fact, we can ensure that all blocks that are constructed within the first s_{cd} rounds of the epoch are finalized because when the new epoch starts the blockchain has already extended $\tau(\ell_e - s_{cd}) \geq k$ blocks after the round s_{cd}. In this way, we make sure that all honest parties agree on the same set of finalized blocks in the end of the first epoch thanks to the CP and CG property. This allows all honest parties to determine the median clock value by examining the clocks associated with these *same blocks*. Therefore, we can leverage the result of Theorem 3.1 which guarantees bounded clock differences between honest parties. The CD property ensures that more than half of the blocks used in the median algorithm are contributed by honest parties. So, the output of the median clock value falls between the minimum and maximum values of the clock values of honest clocks. Given that an honest party's clock has been already δ-close to other honest parties' clock in the beginning of the first epoch 0, the resulted new clock in the end of $e = 0$ does not drift apart from the original clock used during $e = 0$. Therefore, in the end, all honest parties obtain clocks as if $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ provides them. Now, when $e = 1$, parties in $\pi_{\mathcal{H}}$ with the relative time protocol and $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model are identical. In next epochs $e \geq 1$, parties in both $\pi_{\mathcal{H}}$ with the relative time protocol and $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model remain identical. Using the same reasoning as in $\mathsf{epoch} = 0$, we can show that honest parties obtain clocks that closely resemble those provided by $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ at the end of each epoch.

5 Conclusion

In this paper, we formally designed a security framework in the GUC model to capture physical clock synchronization in a distributed system without any reference clock. Our model is the first GUC model that captures the notion of consensus on clock. To achieve clock synchronization, we introduced the relative time protocol, a synchronization mechanism designed to work on top of a blockchain protocol. Leveraging the regular messaging process in the blockchain protocol, our protocol ensures consensus among honest parties' clocks, preventing them from diverging significantly and maintaining the security of the underlying blockchain protocol. It guarantees that new clocks are not significantly different from the old clocks, ensuring that all parties participate in every round as required by the blockchain protocol. Furthermore, our synchronization mechanism is not limited to block producers within the blockchain protocol. External parties running different protocols can utilize our protocol to synchronize their clocks without actively participating in the blockchain. This flexibility allows for broader applications of our protocol in various distributed systems.

We describe shortly how a blockchain protocol following the general structure in Algorithm 2 deploy our relative time protocol. First, we need a formal analysis for CP, CD and CG properties of the respective blockchain protocol and obtain the parameters k for CP, s_{cd} for CD and s_{cg} and τ for CG properties. Then set the epoch length $\ell_e \geq s_{cd} + \frac{k}{\tau}$. Assuming that the maximum networks delay δ assumption is set for it, we next need to set the parameter $\epsilon_{\max}$ to determine the round duration of the blockchain protocol that will deploy the relative time protocol. Once $\epsilon_{\max}$ is set, then the round duration should be set $T \geq 2\delta + 2\epsilon_{\max}$. After this, the blockchain protocol can deploy the relative time protocol as described in Algorithm

4 securely. We next analyse whether determining $\epsilon_{\max}$ is feasible in real world clocks.

Frequency error of real-world clocks: The frequency error of real world clocks is typically measured in terms of part-per-million (PPM), which represents the ratio of the difference in frequency to the ideal frequency multiplied by one million. For example, a clock with a frequency error of 10^{-8} PPM means that its tick rate deviates from the ideal tick rate by one part in ten million. This corresponds to an error of approximately 1.5 nanoseconds per day. This level of accuracy is achieved by high-precision clocks. On the other hand, more common clocks may have frequency errors in the range of 1-20 ppm. This means that their actual tick rates can deviate by up to 20 parts in one million from the ideal tick rate. Consequently, the error in these clocks can range from tens of microseconds to hundreds of microseconds per day. Even though the frequency error in all clocks is inevitable, they are mostly stable [34, 27]. It means that their frequency error ranges between min- ppm to max-ppm.

To analyze the frequency error and stability of computer clocks, we conducted an extensive experiment involving 21 different computer clocks over a period of five months. During this experiment, these computers were actively running the Polkadot relay chain protocol BABE, allowing us to observe the potential impact of CPU load on clock performance. To collect the necessary data, we deployed the NTP daemon (npdp), which periodically records the drift data of the system clock in terms of parts per million (PPM) and stores it in the file `/var/lib/ntp/ntp.drift`. The analysis of the data, as shown in Figure 4, revealed that the frequency error of the system clocks exhibited stable behavior over time. However, the frequency error was not close to zero, indicating that the system clocks were stable but not accurate. The maximum frequency error observed between these clocks was $\epsilon_{19 \triangleright 12} = 20.22 - 2.82 = 17.4$ ppm. Considering that one epoch consists of at most ℓ_e rounds, which corresponds to $\ell_e T$ seconds, we can set the maximum allowable frequency error as $\ell_e T \times 17.4 \times 10^{-6}$ for the clocks used in our experiment. For instance, if $\ell_e T$ is 12 hours, the maximum frequency error in an epoch would be $(12 \times 60 \times 60) \times 20 \times 10^{-6} \approx 0.75$ seconds.

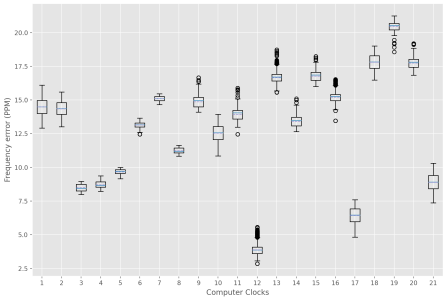

Figure 4: Clock frequency error data. The dashed line is the standard deviation and the line is the median. Each box shows the distribution of the data based on five numbers: maximum, third quartile, median, first quartile and minimum in order. The circles are outliers.

References

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

[2] 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.

[3] 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.

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

[5] J. Burdges, H. K. Alper, A. Stewart, and S. Vasilyev. Sassafras and semi-anonymous single leader election. *Cryptology ePrint Archive*, 2023.

[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] J. He, P. Cheng, L. Shi, and J. Chen. Sats: Secure average-consensus-based time synchronization in wireless sensor networks. *IEEE Transactions on Signal Processing*, 61(24):6387–6400, 2013.

[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] 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.

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] H. Marouani, M. R. Dagenais, et al. Internal clock drift estimation in computer clusters. *Journal of Computer Networks and Communications*, 2008, 2008.

[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] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] Y. Wu and X. He. Finite-time consensus-based clock synchronization under deception attacks. *IEEE Access*, 8:110748–110758, 2020.

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 $\mathcal{G}$.

B Security Analysis

Theorem B.1. *Assuming that underlying blockchain protocol $\pi_{\mathcal{H}}$ has the common prefix property with the parameter k, the chain density property with the parameter s_{cd}, the chain growth property with the parameters τ and $s_{cg} \leq \ell_e - s_{cd}$, $\tau(\ell_e - s_{cd}) \geq k$ and clocks are T-stable, $\pi_{\mathcal{H}}$ with the relative time protocol realizes $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model where $\Theta_p = \delta + \epsilon_{\max}$ and $\Theta_c = \Theta_p + \delta$ except with the probability $p_{cp} + p_{cd} + p_{cg}$ which are the probability of breaking CP, CD and CG properties in $\pi_{\mathcal{H}}$, respectively.*

Here, $\epsilon_{k \rhd \ell}(c, c+T) \leq \epsilon_T$ for all P_k, P_ℓ and $\epsilon_{\max} = \ell_e \epsilon_T$.

Proof. In order to prove the theorem, we construct a simulator $\mathcal{S}$ where $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{com}}^{\delta}$ in $\pi_{\mathcal{H}}$ with the relative time protocol and simulates the adversary in $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ hybrid model. It first registers $\mathcal{G}_{\mathsf{refTimer}}$ to emulate $\mathcal{F}_{\mathsf{com}}^{\delta}$. The simulation is straightforward. As soon as, $\mathcal{Z}$ initiates a party P_i, $\mathcal{S}$ simulates P_i in $\pi_{\mathcal{H}}$ with the relative time protocol. Whenever P_i in $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model sends message with $\mathcal{F}_{\mathsf{com}}^{\delta}$, $\mathcal{F}_{\mathsf{com}}^{\delta}$ contacts with $\mathcal{S}$ and $\mathcal{S}$ relays it to $\mathcal{A}$. Then $\mathcal{S}$ 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 $\mathcal{G}_{\mathsf{refTimer}}$, $\mathcal{S}$ moves the block to the inbox of honest parties in $\pi_{\mathcal{H}}$ with relative time protocol as $\mathcal{F}_{\mathsf{com}}^{\delta}$ in the $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model. If permission is received $\mathcal{S}$ informs $\mathcal{F}_{\mathsf{com}}^{\delta}$ in $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model. In either case, it creates a candidate clock as described in line 18 according to the arrival time of a block. When $\mathcal{S}$ receives clocks $\boldsymbol{\mathcal{C}}_j$'s from $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$, $\mathcal{S}$ finds the median clocks as described in the relative time protocol and updates the clocks of honest parties accordingly. $\mathcal{S}$ checks if the new clocks satisfies the conditions expected by the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$. If they do not satisfy, it aborts. Otherwise, it sends the updated clocks of honest parties to $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ if . Then, $\mathcal{S}$ outputs the clocks of honest parties in $\pi_{\mathcal{H}}$ with the relative time protocol. The output of an honest party in the real world and the honest party in the ideal world are not the same if $\mathcal{S}$ aborts. $\mathcal{S}$ aborts 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 analyse the probability of having such bad events in our simulation in any epoch.

(1. Case): If the CG property satisfied in an epoch, the chain grows $\tau(\ell_e - s_{cd})$ blocks after the first s_{cd} rounds of the epoch. If CP property is satisfied too, we can guarantee that the blocks generated during the first s_{cd} rounds in an epoch are finalized because $\tau(\ell_e - s_{cd}) \geq k$. Thus, all honest parties obtain clocks of the same blocks during this epoch before updating their clocks for the new epoch. This means that parties satisfy the assumption of BCSP protocol i.e., all clocks built during BCSP constructed based on the same message for each

honest party. Thus, we can use the result of Theorem 3.1 and conclude that the difference between the new clocks of honest parties in $\pi_{\mathcal{H}}$ with the relative time protocol is at most $\delta + \epsilon_{\max}$ i.e., for all $P_i, P_j \in \mathcal{P}_h$, $\boldsymbol{\mathcal{C}}_j - \boldsymbol{\mathcal{C}}_i \leq \delta + \epsilon_{\max}$ just after running the relative time protocol.

(2. Case) If parties preserve the synchronization during an epoch meaning that the first case is satisfied, $\pi_{\mathcal{H}}$ with the relative time protocol preserves the security in that epoch so it has the CD-property. It means that majority of the blocks (at least $\lfloor \frac{n}{2} \rfloor + 1$ finalized blocks in the epoch) used in Line 22 are honest ones except with the probability p_{cd}.

We now show the difference between the new clock $\boldsymbol{\mathcal{C}}_{i \to k}$ and the old clock $\boldsymbol{\mathcal{C}}_k$ is at most Θ_c assuming that $\lfloor \frac{n}{2} \rfloor + 1$ of the finalized blocks during the simulation of an epoch were sent by honest parties. We first assume that the block producer of the block B_i is an honest party P_ℓ. If it is the case, $0 \leq \boldsymbol{\mathcal{C}}_\ell - \boldsymbol{\mathcal{C}}_{i \to k} \leq \delta$. Since we know that synchronization between clocks of honest parties satisfied, $\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_\ell \leq \Theta_p$ during the epoch. Therefore, $\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_{i \to k} \leq \boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_\ell + \delta \leq \Theta_p + \delta \leq \Theta_c$. Now, we assume that B_i is generated by a malicious party. Since the clock value of $\boldsymbol{\mathcal{C}}_{i \to k}$ is the median value and majority of the clocks constructed in an epoch based on hones blocks, then there exist $\boldsymbol{\mathcal{C}}_{x \to k}, \boldsymbol{\mathcal{C}}_{y \to k}$ where B_x and B_y are generated by honest parties and $\boldsymbol{\mathcal{C}}_{x \to k} \leq \boldsymbol{\mathcal{C}}_{i \to k} \leq \boldsymbol{\mathcal{C}}_{y \to k}$. Therefore, because of the same reasoning in the case of B_i is an honest block, we can conclude that $\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_{x \to k} \leq \Theta_p + \delta \leq \Theta_c$ and $\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_{y \to j} \leq \Theta_p + \delta \leq \Theta_c$. We know that $\boldsymbol{\mathcal{C}}_{x \to k} \leq \boldsymbol{\mathcal{C}}_{i \to k} \leq \boldsymbol{\mathcal{C}}_{y \to k}$, so $\boldsymbol{\mathcal{C}}_k - \boldsymbol{\mathcal{C}}_{i \to k} \leq \Theta_p + \delta$.

(3. Case): IsNewEpoch always returns true if all clocks of the blocks are constructed during the epoch. Therefore, the parties who have a null clock obtain their new clocks from the same candidate clocks as others and they cannot be null.

We have shown that clocks sent by $\mathcal{S}$ is as expected by $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$, if CD, CP, CG properties are satisfied during an epoch. Now, we need to show that it is the case for all epochs. We know that the security properties of $\pi_{\mathcal{H}}$ maintain in $\pi_{\mathcal{H}}$ in the $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$-hybrid model as long as $T \geq \delta + \Theta_p + 2\epsilon_{\max}$ and $\Theta_c < T$. During $\mathsf{epoch} = 0$, parties are identical in both protocols. Therefore, $\pi_{\mathcal{H}}$ with the relative time protocol maintains the same security properties as in $\pi_{\mathcal{H}}$ during $\mathsf{epoch} = 0$. Given this, the clocks sent by $\mathcal{S}$ preserves the conditions defined in $\mathcal{F}_{\mathsf{Cclock}}^{\Theta}$ in the end of $\mathsf{epoch} = 0$. Now, parties during $\mathsf{epoch} = 1$ are identical in both protocols. From the same reasoning, the CD and CP properties are preserved next epochs given that they are preserved in the previous epoch. □ □