NodeChain: Cheap Data Integrity Without Consensus

Orfeas Stefanos Thyfronitis Litos
Imperial College London

Zhaoxuan Wu
Imperial College London

Alfredo Musumeci
Imperial College London

Songyun Hu
Imperial College London

James Helsby
Imperial College London

Michael Breza
Imperial College London

William Knottenbelt
Imperial College London

Abstract

Blockchains enable decentralised applications that withstand Byzantine failures and do not need a central authority. Unfortunately, their massive replication requirements preclude their use on constrained devices.

We propose a novel blockchain-based data structure which forgoes replication without affecting the append-only nature of blockchains, making it suitable for maintaining data integrity over networks of storage-constrained devices. Our solution does not provide consensus, which is not required by our motivating application, namely securely storing sensor data of containers in cargo ships.

We elucidate the practical promise of our technique by following a multi-faceted approach: We (i) formally prove the security of our protocol in the *Universal Composition* (UC) setting, as well as (ii) provide a small-scale proof-of-concept implementation, (iii) a performance simulation for large-scale deployments which showcases a reduction in storage of more than 1000x compared to traditional blockchains, and (iv) a resilience simulation that predicts the practical effects of network jamming attacks.

1 Introduction

Since the advent of Bitcoin [39], blockchains have emerged as a potent component of multi-party protocols, enabling decentralised applications that admit Byzantine participants and eschew single points of failure. While mainly used in financial contexts [54, 57], blockchains have also seen success in supply chain applications [3, 31] where strict data integrity is paramount and no central authority can be trusted by all. While useful in current financial applications, existing blockchains are too heavy for constrained devices.

Motivation. Blockchains on constrained devices have been proposed for use in the shipping industry. In particular, containers in cargo ships are routinely fitted with sensors that log ambient data, e.g., temperature or humidity. In case of disputes over damaged cargo, this data can be used by insurance companies or courts to aid adjudication, therefore its integrity is instrumental. Previous blockchain solutions do not work on constrained devices because they impose massive storage requirements per device, as well as because consensus requires at least 3 rounds of communication which is taxing on their battery.

We address these problems by proposing a blockchain-based mechanism that efficiently stores sensor measurements while continuously guaranteeing that the sensors' logs are append-only: existing data cannot be changed after the fact, but only new data can be added (i.e., history cannot be rewritten, a.k.a. *integrity*). This setting imposes strict storage and power consumption constraints on devices. It also forces the use of a wireless network, thus *jamming attacks* (whereby a malicious antenna "drowns out" legitimate traffic around it) become a concern. Our mechanism is under active production development in an industrial context and there already exists client demand.

Core protocol & guarantees. At the core of our system (Fig. 1), each party P (a.k.a. *leader*) builds a separate *nodechain* C_P, a data structure akin to a blockchain, in which P unilaterally decides the block contents (batches of sensor data in our application) and finalises each new block on its own cheaply (at about the cost of a signed git commit) before broadcasting it. All other honest parties (a.k.a. *attestors*) send *attestations* on new C_P blocks to P in an online fashion and reject *forked* nodechains (i.e., chains with two valid, conflicting blocks at the same *block height*, indicating an attempt to rewrite history). Every party is the leader of its own nodechain, as well as an attestor to other parties' nodechains. Each nodechain is independent from the rest.

Under relaxed assumptions, honest parties are guaranteed to receive sufficient attestations on their new blocks (*liveness*). Furthermore, any external observer (a.k.a. *judge*) can verify whether a nodechain has collected sufficient attestations for its blocks (cf. Subsec 2.2), in which case the judge has *integrity* guarantees, i.e., that the data in the nodechain have not been altered after they were originally broadcast.

On consensus. Since attestors do not exchange attestations among themselves, they do not reach consensus on the contents of nodechains, making the design unsuitable for applications that require it. In our case however, consensus turns out to be superfluous as we do not need a total ordering of all parties' data, nor does a party need a consistent view of other parties' data to build its own nodechain, thanks to a lack of dependencies among nodechains.

The benefit is that each attestor only needs to store the (hash of the) latest block of others' nodechains, reducing the attestor's per-nodechain storage overhead from the entire nodechain data down to a small constant, while communication complexity is reduced to a single round per block (as opposed to at least two rounds per block needed to reach consensus). Thanks to our protocol design (Sec. 3) along with a number of optimisations (Subsec. 5.4), our simulations demonstrate a 430x storage and an almost 8x computation reduction compared to a Tendermint-based blockchain [15], one of the most lightweight, widely used, and state-of-the-art consensus protocols.

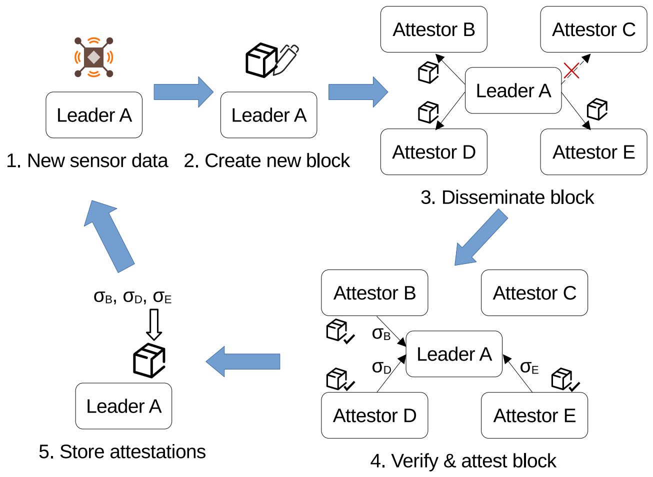

Figure 1. Creation and attestation of new blocks. Each leader creates a new block containing new sensor data and disseminates it to other parties on a best-effort basis — here sending to C fails. Attestors in turn verify that the new block correctly extends the previously known latest block of the leader's nodechain and send an attestation (i.e., a signature) to the new block back to the leader. The leader then aggregates and stores the attestations. Each leader periodically repeats this process throughout the protocol execution. The protocol is resilient to corruptions and network failures, as long as assumptions 1–4 are upheld. At a later time, a judge can deduce that the nodechain has not been tampered with (i.e., has *integrity*) by verifying block correctness and validity of sufficient attestations.

Network model. Shipping containers are not rearranged while at sea, thus the (wireless) network topology is fixed and known to all. We take advantage of this permissioned, known-topology setting to establish a fine-grained network model in which direct wireless links are represented by edges on a graph that admits jamming attacks and network partitions. This assumption is weaker than the state of the art, which normally assumes that all messages eventually reach their destination (i.e., asynchronous communication or stronger) [5, 41, 59].

Connectivity and honesty assumptions. As we formally prove (Sec. 2 and Appx. F), our protocol guarantees data integrity as long as the network connectivity and the honest party topology uphold certain reasonable assumptions. The latter generalise the usual honest majority assumptions to less rigid *trustsets* – this generalisation is well-suited in the context of diverse stakeholders with each owning possibly more than one node (container), as well as allows by design each judge to select which sets of participants they trust.

Implementation & Simulations. We have deployed our protocol as Proof of Concept (Sec. 4) using Hyperledger Fabric [6] and we simulate its storage and energy requirements at scale (Sec. 5). We also explore the resilience of our assumptions against jamming attacks (Sec. 6).

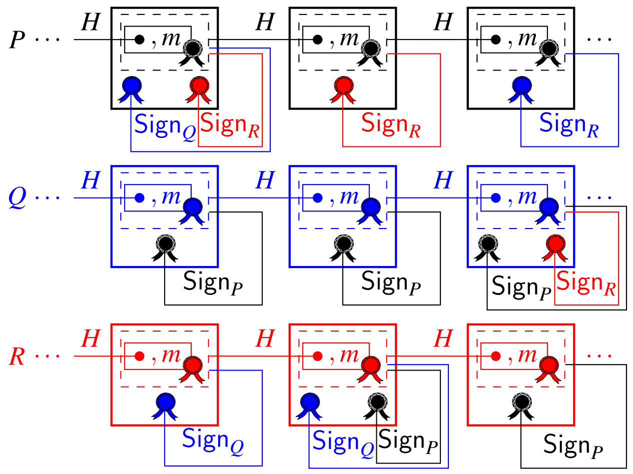

Figure 2. The nodechains of 3 parties. In P's chain, each block (thick-bordered black box) contains the previous hash, data m, P's signature, as well as attestations from (some of) the other parties. The previous hash commits to all contents of the previous block except for its attestations (dashed box). P decides data m. P's signature covers the previous hash and the block data, but not the block attestations (thin-bordered box). The attestations sign the previous hash, the data and P's signature (dashed box). Q's and R's nodechains follow the same pattern. Each nodechain evolves independently from the rest.

1.1 Our contributions

- We propose a novel, blockchain-based data structure without consensus nor data replication: Each party maintains its own *nodechain* (lightweight blockchain) and gossips new blocks to the other parties, who *attest* that new blocks are constructed correctly and do not induce forks (Fig. 2, Sec. 3). Each party stores only its own nodechain in its entirety, whereas it stores only the hash of the latest block of all other parties without replicating the contents of others' nodechains, thus minimising storage overhead.

- We formally prove our construction guarantees data integrity under reasonable assumptions in the Universal Composition (UC) [17] framework: Honest parties receive enough attestations on each block within a known delay (liveness), while malicious parties that attempt forks stop receiving attestations and thus cannot later fool a judge (integrity) (Appx. D–F).

- We model communication with an explicit graph that allows network partitions (Subsec. 2.1), as well as formalise trust and connectivity assumptions with a novel generalised framework (Subsec. 2.2). In contrast to consensus protocols, our relaxed assumptions support integrity and liveness even if some parties remain indefinitely disconnected. We further prove that the special case of two-thirds supermajority of honest, well-connected parties satisfies our assumptions and thus guarantees security, in line with Byzantine Fault Tolerant (BFT) protocols [58].

- Our construction is implemented as a Proof of Concept over Hyperledger Fabric and is executed over 12 wirelessly connected Raspberry Pis (Sec. 4). We induce a $4/8$ network partition, during which the parties continue producing blocks and exchanging attestations within the two disconnected components on a best-effort basis, contrary to some prior blockchains (e.g., BFT-based systems [15]). When we resolve the partition, the nodes automatically revert to exchanging attestations with all other nodes, thus containing the disruption within the partition period without the need to resolve forks after.

- We evaluate the performance of our system at scale via simulation (Sec. 5). Our construction requires roughly 6000x less storage and 8x less computation than a traditional blockchain in the biggest deployment of interest, in line with the theoretical storage improvement from $O(nt)$ to $O(n+t)$ (for n parties and t blocks).

- Furthermore, we assess the resilience of our assumptions under realistic attacks via a Cooja [37] simulation, resulting in actionable recommendations regarding the network configuration of practical deployments (Sec. 6). Our simulation framework is also useful for testing the resilience of candidate configurations of practical deployments.

1.2 Related Work

Explicit modelling of communication link failures has been explored in the past, with [23] providing fundamental results, extended by [51] to asynchronous communication. [55,56] investigate different sets of limitations on link failures, but without Byzantine node corruptions. [11,53] explore consensus in the presence of Byzantine nodes and per-node bounds on link failures. They also experimentally check *assumption coverage*, i.e., how likely it is for their assumptions to break under probabilistic (not adversarial) failures. [38] provides a practical binary consensus protocol resilient to Byzantine nodes and globally bound link failures. [16] provides a thorough introduction to the state of the art of distributed systems, including arbitrary node corruptions and link failures. The specific assumptions of our work consist of flexible limits on Byzantine corruptions and global conditions on link failures, and, to the best of our knowledge, have not been explored in the past.

Tendermint [15] is one of the most lightweight, widely used, and state-of-the-art consensus protocols, and an obvious contender for use in constrained devices. Our evaluation (Sec. 5) shows that our approach requires around 6000x less storage and uses around 8x less computational resources.

Prior work has explored avoiding total order of parties' messages. In Astro [19] exclusive logs are like nodechains in that each log can only be extended by one party and every party has its own log. However we do not need Astro's broadcast layer (which adds a second round of communication per block, the COMMIT message), replication of each party's chain at all nodes, or dependency (CREDIT) messages. Also Astro lacks a formal security analysis.

Narwhal [21] is a protocol related to Astro and with similar aims. At every round, each party broadcasts a block and other parties certify it if it builds on at least $2f+1$ certified blocks from the previous round (where f is the maximum number of corrupt parties the protocol can withstand). Like Astro, Narwhal needs two rounds of communication per block. Unlike Astro, these rounds cannot benefit from pipelining: the first round of a block can only happen after the second round of the previous block is completed. Similar to Astro, the security of Narwhal is not formally proven.

2 Model

Let us now dive into the model details. We here formalise the context in which our protocol runs, i.e., the network model, the trustsets, the threat model, and the connectivity assumptions, as well as the desired protocol behaviours, i.e., the security guarantees and performance goals.

2.1 Network Setting

Containers aboard a cargo ship communicate over an ad-hoc wireless mesh Local Area Network (LAN) and their location on board remains fixed while at sea. This enables us to formally model the exact network as a directed graph $G = (\mathcal{P}, E)$, where $\mathcal{P}$ is the set of parties (consisting of the containers and the ship itself) and $(u, v) \in E$ are the direct communication links between them. The graph might not be connected, i.e., there might exist nodes that cannot communicate. Our network model is thus strictly stronger than the strongest network model used in the blockchain literature [41,59], i.e., the asynchronous model, in which parties' messages can eventually reach any destination. The network is modelled by the ideal functionality $\mathcal{F}_{\text{Net}}$ (Appx. D.1).

We model time as a sequence of discrete synchronous rounds. A message takes exactly one round to propagate across an edge in E. $\mathcal{F}_{\text{Net}}$ also encompasses the modelling of time.

2.2 Trustsets

When are block attestations considered sufficient? To the best of our knowledge, we for the first time put forth the concept of *trustsets* as a fully flexible answer: A trustset T is a set of parties such that, if every party in the trustset attests to a block, then a judge will accept this block as valid. A judge can trust multiple trustsets $\mathcal{T}$ simultaneously. For a nodechain to be valid, every one of its blocks must be attested by *every party of at least one trustset* — the attesting trustsets need not coincide across blocks.

Trustsets can model the full range of trust topologies. Let us see three examples:

- Centralised trust to a single party P corresponds to a single trustset containing only P,
- Distributed trust to all parties of either committee $Ps = \{P_1, \dots, P_n\}$ or all parties of committee $Rs = \{R_1, \dots, R_n\}$ corresponds to trustsets $\mathcal{T} = \{Ps, Rs\}$,
- Decentralised trust to any quorum consisting of at least two-thirds of all protocol parties corresponds to the set of all subsets of $\mathcal{P}$ with at least $\frac{2|\mathcal{P}|}{3}$ elements: $\mathcal{T} = \{S \subset \mathcal{P} : |S| \geq \frac{2|\mathcal{P}|}{3}\}$.

In this work we assume a single, publicly known, fixed set of trustsets as a system parameter. This allows parties to know whether they have received sufficient attestations for their blocks. An alternative configuration is possible, in which different judges may choose their own trustsets and even keep their choice private. This would however come at the expense of parties not knowing if they have collected enough attestations.

2.3 Adversarial Behaviour

Our protocol is resilient to Byzantine misbehaviours by parties, including crashes and arbitrary divergence from the honest protocol. We use the *adaptive corruption* model: There is a single adversary $\mathcal{A}$ that can corrupt any party at any point in time, after which the party's actions are controlled by $\mathcal{A}$ for the remainder of the protocol execution. We denote the set of honest parties with $\mathcal{H}$and corrupted parties with $\mathcal{P} \setminus \mathcal{H}$.

Furthermore, the adversary can carry out *jamming attacks* by selecting the edges to be removed from the graph at each round. This models attacks on the physical wireless medium (e.g., the emission of high-amplitude white noise by a corrupted node over the entire range of used frequencies). Such attacks can disrupt communication between honest nodes for arbitrary stretches of time, even for the entire protocol duration. In fact, we choose modelling the network as a communication graph (rather than an abstract network interface, as in [8]) in order to explicitly include jamming attacks.

2.4 Honesty & Connectivity Assumptions

To achieve any security guarantees, we must of course impose some limits to the adversary's ability to corrupt parties and to sever communication links. We put forth 4 assumptions, the formal versions of which can be found in Fig. 7:

1. *Per-trustset existential honesty:* Every trustset contains at least one honest party.
2. *Malicious connectivity:* Every malicious party is connected to at least one honest party in each trustset (see assumption 1) at least once every t_{rep} rounds.
3. *Fully honest trustset:* There exists at least one trustset consisting exclusively of honest parties.
4. *Honest connectivity:* Every honest party achieves a round-trip connection with every party of at least one fully honest trustset (see assumption 3) at least once every t_{rep} rounds.

Looking ahead, t_{rep} is a system parameter that defines the initial time-to-live of network messages, as well as the number of blocks at the tip of a nodechain that judges cannot trust. Assumptions 1 and 3 impose constraints on party corruptions, whereas assumptions 2 and 4 limit what edges can be jammed simultaneously and for how long.

Let us now compare our assumptions to the standard honesty assumption found in other blockchain designs (e.g., [15, 26]), i.e., that strictly less than $1/3$ of the participants are corrupted. Observe that, under the latter assumption, a new block that is endorsed by at least $2/3$ of the parties cannot be "forked" by a different block endorsed at the same height. This is so because (by the pigeonhole principle) the two sets of endorsing parties would contain at least $1/3$ of all participants in common and thus the intersection would contain at least one honest party — but honest parties never endorse different blocks at the same height, so forks are impossible.

This assumption corresponds to the decentralised trust discussed in Subsec. 2.2 with trustsets $\mathcal{T} = \{S \subset \mathcal{P} : |S| \geq \frac{2|\mathcal{P}|}{3}\}$. It is easy to show that with these trustsets, the standard assumption is sufficient for assumptions 1 and 3 to hold: Due to its size, every trustset contains at least $|\mathcal{P}|/3$ honest parties, thus satisfying assumption 1. Furthermore, since the corrupted parties are less than $|\mathcal{P}|/3$, it is $|\mathcal{H}| \geq \frac{2|\mathcal{P}|}{3}$, thus $\mathcal{H} \in \mathcal{T}$, satisfying assumption 3. We further observe that asynchronous communication implies both assumptions 2 and 4. Taken together, the above mean that our assumptions are more general and can even be weaker than the state of the art.

2.5 Cryptography

We model the intended behaviour our protocol, including its security guarantees, as the $\mathcal{F}_{\text{Ledgers}}$ ideal functionality in the UC [17] framework (Appx. D.2). Briefly, this is a composable framework of simulation-based cryptography [33], whereby two worlds operating in the context of an arbitrary environment $\mathcal{E}$ are juxtaposed: the ideal world, in which the ideal functionality encodes the intended protocol behaviour, and the real world, in which parties run the actual protocol. Security is achieved if the two worlds are *indistinguishable*, i.e., if no efficient (i.e., *probabilistic polynomial-time*) environment $\mathcal{E}$ can distinguish the two worlds (except with negligible probability). Our real-world protocol Π_{Ledgers} can be found in Appx. D.3 and a more extensive discussion of UC in Appx. C.

Our protocol leverages two simple cryptographic primitives: hashes and digital signatures. The hash function $\mathsf{Hash}()$ is used to ensure each block of a nodechain references the previous block, as well as to minimise message size by hashing block contents before sending them. As is standard in simulation-based security proofs, the hash function is modelled as a Random Oracle.

The digital signatures are used by a leader when creating a new block to authenticate it, as well as by attestors to generate attestations. In the ideal world, they are modelled as an ideal functionality $\mathcal{F}_{\text{Sig}}$ [18]. In the real world, leaders use Schnorr signatures [46] due to their small size, whereas attestors use BLS signatures [14] due to their aggregatability (Subsec. 5.4).

2.6 Security Properties

Our protocol must provide two security properties: *integrity*, i.e., that a judge never accepts a forked nodechain, and *liveness*, i.e., that an honest party gets sufficient attestations on its every new block. See Appx. F for the formal guarantees.

Looking ahead (Sec. 3), when presented with a nodechain, a judge removes the last t_{rep} blocks and accepts it if every remaining block has been attested by all parties of at least one trustset. Honest parties stop attesting to new blocks of forked nodechains after detecting the fork. Assumptions 1 and 2 ensure honest parties are present in all trustsets and eventually learn about all forks. Together, the above guarantee integrity to the judge. Similarly, assumptions 3 and 4 ensure that at least one honest trustset will learn (and thus attest to) new honest blocks, which guarantee liveness to honest parties.

We note that the per-nodechain integrity guarantee is not sufficient for applications that need total order of transactions, e.g., complex financial protocols. Rather, it is suitable when we need to guarantee that each party's chain has not been forked. Apart from storing containers' sensor readings, another possible application of nodechains is storing public figures' statements.

2.7 Performance Properties

Our motivating application, i.e., securely storing sensor readings of containers aboard a ship, imposes strict performance requirements in terms of storage and energy consumption — most containers lack a power supply, so the devices need to be battery-powered. If every party creates t blocks in total, just storing them without integrity would require storage $O(t)$. On the other hand, storing all the data in a single traditional blockchain would impose $O(|\mathcal{P}|t)$ storage requirements. Our design goal is a protocol in which each party needs $O(|\mathcal{P}|+t)$ storage.

Looking ahead (Section 5), each party in our protocol indeed requires $O(|\mathcal{P}|+t)$ storage where the constant of $|\mathcal{P}|$ is only 368 bytes. Its energy requirements for a single container in the biggest possible deployment given today's largest container ships' capacity for a 1-month trip with 1 measurement per minute is less than 250kJ on average, i.e., around 1.75 commercial, $10,000$mAh power banks.

3 Protocol Overview

Let $\mathcal{P}$ be the set of all parties and $\mathcal{H} \subset \mathcal{P}$ its honest subset. Furthermore, fix trustsets $\mathcal{T}$; every trustset $T \in \mathcal{T}$ consists of parties in $\mathcal{P}$. Also consider an adversary $\mathcal{A}$ that obeys the corruption (1, 3) and network connectivity (2, 4) assumptions.

Each honest party $P \in \mathcal{H}$ fulfills 3 roles: *Leader* of its own nodechain C_P, *attestor* of the nodechains of others (Fig. 1), and *judge* of the validity of received nodechains accompanied by attestations — we assign all three roles to all honest parties to avoid differentiating parties, facilitating the UC modelling of the protocol (Π_{Ledgers}, Appx. D.3). We here provide a description of the protocol. We refer the reader to Appx. E for the proof of UC security and to Appx. F for the proofs of the two security properties (integrity and liveness).

Leader. Let us first see how $P \in \mathcal{H}$ fulfills its *leader* role. Initially, P's nodechain C_P consists of a single *genesis* block $(\varepsilon, 0, \varepsilon, \mathsf{Sign}(sk_P, \varepsilon))$, where sk_P is P's signing key and ε the empty string.

Let $\mathsf{prevhash}$ be the $\mathsf{Hash}()$ of the latest block in C_P and $h = |C_P|$. Upon receiving data x from its sensor (which in practice happens periodically and in Π_{Ledgers} is modelled as a (SUBMIT, x) message by the environment $\mathcal{E}$ that can arrive at any moment), P creates a new block $(x, h+1, \mathsf{prevhash}, \mathsf{Sign}(sk_P, \mathsf{prevhash}||x||h+1))$, appends it to C_P, and sends it (via $\mathcal{F}_{\text{Net}}$) to all other parties, expecting attestations. Upon receiving an attestation by $R \in \mathcal{P}$ for its block of height $h+1$, P verifies that the attestation is valid (using signature verification on the attestation with R's public key pk_R) and adds it to the set of attestations for height $h+1$ (stored alongside the block of height $h+1$, cf. Fig. 2).

Attestor. P acts as *attestor* for each party $R \in \mathcal{P}$, as long as P has not marked R as corrupt. For leader R, P is initialised with pk_R and R's signature on the genesis block of C_R. P keeps in its storage the hash $\mathsf{prevhash}$ and the height h of the latest valid block of C_R that it has learned.

Upon receiving by R (via $\mathcal{F}_{\text{Net}}$) a new block $b = (x, h', H, \sigma)$, P first verifies the block by performing three checks: (i) that $\mathsf{Verify}(pk_R, H||x||h, \sigma)$ (if this check fails, then the block is ignored as it has not been correctly authenticated), (ii) that $h' = h+1$ (if this check fails, then the block is ignored as it is not at the height P expects), and (iii) that $H = \mathsf{prevhash}$ (if this check fails, then R is marked as corrupt and ignored for the remainder of the execution, as it has provided an authenticated, but invalid, block). If the block is verified, P replaces $\mathsf{prevhash}$ with $\mathsf{Hash}(b)$, increments h by 1, and generates and sends to R (via $\mathcal{F}_{\text{Net}}$) the attestation $\mathsf{Sign}(sk_P, b)$.

Judge. As a *judge*, P expects a message (JUDGE, C_R, R) by $\mathcal{E}$. In order for parties outside the ship to also be able to act as judges, P does not use any of the data it has collected during the protocol in order to judge a nodechain, only the trustsets $\mathcal{T}$ and the parties' public keys.

P first removes the last t_{rep} blocks from C_R. It then checks the genesis block: it ensures that $C_R[0] = (\varepsilon, 0, \varepsilon, \sigma)$ and $\mathsf{Verify}(pk_R, \varepsilon, \sigma)$ — genesis blocks need no attestations. For each subsequent block in C_R, P checks that (i) its height is 1 plus the height of the previous block, (ii) its $\mathsf{prevhash}$ equals the $\mathsf{Hash}()$ of the previous block, (iii) R's signature on the block is valid, (iv) each attestation of this block is a valid signature on the block, and (v) each block has attestations by all parties of at least one trustset $T \in \mathcal{T}$. If all checks succeed, the nodechain is accepted, otherwise it is rejected.

Intuitively, our protocol achieves integrity under assumptions 1 and 2 since there exists no trustset of which all the attestors will attest to a forking block. Furthermore, as long as new blocks reach attestors in the order they are generated, our protocol as described here achieves liveness. See Appx. A for discussion on further protocol improvements.

4 Proof of Concept implementation

In order to demonstrate feasibility, we have implemented our protocol as a full-stack Proof of Concept. For the mesh network, we use the batman-adv Linux kernel module [1], which implements a decentralised routing protocol resilient to temporary network partitions. For the blockchain infrastructure (block generation, gossiping, verification) we use Hyperledger Fabric [6], a highly configurable framework for building custom permissioned blockchains. Our implementation is deployed over 12 Raspberry Pi 4B devices, each representing one container. Each device generates simulated temperature data, but Raspberry Pis, thanks to their GPIO port, can be easily connected to a variety of physical sensors (measuring, e.g., temperature, humidity, GPS location, or acceleration).

The progress of the protocol and the various nodechains can be monitored via a specially designed user interface which lists the contents of the blocks of each nodechain along with the number of attestations they have collected. Under normal operation with all nodes following honestly the prescribed protocol, all new blocks are attested by all other nodes.

Furthermore, our Proof of Concept manifests that our protocol is resilient to temporary network partitions during which some attestations are not gossiped. To that end, we temporarily partition the network by physically separating 4 of the 12 devices to sever the wireless connection. The 4 devices are subsequently reunited with the rest after some time. The UI confirms that the expected behaviour is indeed achieved: during the partition, attestations are exchanged only between the devices that remain in the same connected component, but that all nodes revert to exchanging attestations on new blocks with all other nodes after the partition is resolved.

Moreover, our implementation shows that forked blocks are not attested: Via a suitable script, one node sends two diverging blocks for the same height of its nodechain to different parties. After gossipping the conflicting blocks and realising the attempted fork, the honest nodes mark the offending node as corrupt and stop attesting to new blocks by it.

We note that network partitions that last for more than t_{rep} rounds violate assumption 2 and thus the integrity of blocks generated during the partition is not guaranteed by our formal analysis. Nevertheless, our protocol can naturally recover from partitions by simply allowing leaders to inform the previously disconnected attestors of all missed blocks. The reconnected attestors can then resume producing attestations for new blocks — still, they should not attest to blocks generated during the partition, as this would risk attesting to blocks that may have been tampered up until the moment that the partition was resolved.

If each connected component has an honest attestor, forking attempts will be detected after the partition is resolved. Furthermore, even if the partition is never resolved, attestations should still be exchanged on a best-effort basis. Assuming at least one honest attestor per connected component, a judge with access to all parties can still enjoy integrity guarantees for blocks created during the partition. For simplicity, our formal analysis does not model periods of failure of the network assumptions; extending our formal security guarantees to withstand partitions is left as future work (Sec. 7).

Unfortunately, an implementation building on an unmodified Hyperledger Fabric cannot be used at the desired scale (i.e., tens of thousands of nodes). The main reason is that the only way to model single-leader nodechains in Hyperledger Fabric is via the use of Fabric's *channels*. Each channel is a separate blockchain with different, customisable rules governing its execution and consensus. Sadly, participating in a channel (even as an observer) requires storing locally all its blocks. This means that every party must store the full history of everyone else, eliminating the main optimisation of our protocol, i.e., the reduction of per-party storage from $O(|\mathcal{P}|t)$ to $O(|\mathcal{P}|+t)$. Furthermore, Hyperledger Fabric does not remove the need to run consensus for each new block of single-leader channels and requires sensor data to be stored as transactions in blocks — both of these impose undue overheads that a bespoke implementation can avoid by design.

5 Performance Evaluation

Several studies [9, 22, 29, 30] show experimentally that traditional blockchain systems do not scale well as the number of parties grows. Our design forgoes consensus in favour of scalability. In order to demonstrate that this tradeoff is worthwhile, we implemented a discrete-event simulation framework to evaluate the performance of our protocol against traditional alternatives as the number of parties and blocks increase.

The methodology of our simulation is as follows. We first implement the core data structures and main protocol actions, i.e., block production, verification, and attestation, and calculate their storage requirements. Assuming full connectivity and no malicious parties, we then analytically calculate the per-party number of signatures produced, verified, and aggregated, the number of hash function invocations, and the number of network messages exchanged throughout an execution of the protocol. The result is a function of the number of parties and the quantity of blocks they produce. We then extract the energy requirements of each action from the literature and estimate the total energy consumption by multiplying the cost of each action by the number of times it is executed. The simulations were run on an Apple M1 processor with 16 GB RAM, programmed in C++, and compiled using Apple Clang version 15.0.0. We evaluated code performance by measuring elapsed CPU time using the C++ Standard Library function `chrono::high_resolution_clock`.

We provide a specially designed user interface through which one can explore and compare the energy consumption at various scales and with customisable configurations. Last but not least, we carry out the same storage and energy calculations for an alternative deployment that uses a single, traditional blockchain that runs Tendermint [15] for consensus. All needed parameters are chosen to favour Tendermint.

5.1 Storage Requirements

The BLAKE2 hash function [7] is used due to its high performance and sufficient security. Leaders sign new blocks with Schnorr signatures [46], chosen for their cheap generation and verification, small size, and excellent security. Attestors use BLS signatures [14, 48] so that multiple attestations for the same block can be incrementally aggregated into a single BLS signature without multiple rounds of communication — see below for more discussion on aggregation.

There are four central data structures stored by each party: The party's own fixed data, the blocks of the party's own nodechain, the fixed data of each counterparty, and the latest block of a counterparty as stored by an attestor. A party's fixed data consist of its party ID (4 bytes), its BLS ID (32 bytes, needed by the underlying BLS library), its BLS secret key (32 bytes), its BLS public key (288 bytes), its Schnorr secret key (44 bytes), and its randomness seed (32 bytes). These amount to each party storing 432 bytes for its own fixed data.

For each of its blocks, a leader stores the raw block contents (which we arbitrarily set to 12 bytes but could have any size) and the aggreegated BLS attestations (144 bytes [48]), totalling 156 bytes per own block. There is no need to store the Schnorr signature of every block, as it can be recreated by running the signing algorithm with the same randomness, so storing a single randomness seed for the entire execution is enough.

Each party stores the public keys and the latest block of every counterparty — there are $|\mathcal{P}|-1$ counterparties in total. The two public keys of each counterparty are its Schnorr public key (44 bytes), used to verify its signatures on the new blocks of its nodechain, and its BLS public key (288 bytes [48]), used to verify any attestations it sends. As stored, the latest block of a counterparty consists of its hash (32 bytes) and its height (4 bytes). The total per-party storage overhead for the counterparty data is thus $368(|\mathcal{P}|-1)$ bytes.

In total, a party with t blocks that joins an execution of our protocol with $|\mathcal{P}|$ participants stores a total of $156t + 368|\mathcal{P}|+64$ bytes. For comparison, simply storing locally the raw sensor readings without any integrity guarantees would cost $12t$ bytes. The overhead of our protocol is thus

$$144t + 368|\mathcal{P}| + 64 \text{ bytes} \ .$$

In practice, this overhead is acceptable. An average container ship today can carry $|\mathcal{P}| = 3,875$ containers [32] and an average container is in transit for a month [35]. Assuming that all containers participate in the protocol and that each sensor produces 1 reading (and thus 1 block) per 10 seconds ($t = 259,200$), we see that the total storage of our protocol is only $41,861,264\text{B} \approx 40\text{MiB}$ for each participant. To demonstrate future-proofness, let us furthermore examine a deployment much larger than what is possible today: All containers in a container ship of size double the currently largest one ($|\mathcal{P}| = 50,000$ parties) participate in the protocol for 6 months and each container continuously produces 1 block per 10 seconds ($t = 1,555,200$ blocks). Then the total storage of each party is only $261,011,264$B ≈ 249MiB. Due to its relatively small size, this data can likely be stored in the storage medium already used for storing the sensor readings, no additional storage media are needed.

5.2 Energy Consumption

The actions that consume energy in our protocol are mainly signature-related operations (signing, verifying, aggregating), reads from and writes to disk, and network transmissions. Cryptographic operations take 99.6% of the execution time, so we ignore the energy costs of other computations, including hashing, which takes 0.1% of the execution time. The energy consumption of basic actions is summarised in Table 1.

Operation	mJ
BLS.Sign()	50.23
BLS.Verify()	112.59
BLS.AggregateSigs() (2 sigs)	0.28
BLS.AggregatePKs() (2 PKs)	0.56
Schnorr.Sign()	0.52
Schnorr.Verify()	37.53
Read 4096 bytes from disk	$32.77 \cdot 10^{-3}$
Write 4096 bytes to disk	$32.77 \cdot 10^{-3}$
Transmit a packet (2,304B)	0.81

Table 1. Energy consumption estimates of basic actions in millijoules. "Transmit" includes the cost of both sending and receiving.

Producing a BLS signature costs roughly 50.23mJ, whereas verifying one costs approximately 112.59mJ [42,43]. To the best of our knowledge, no reliable estimates of the energy required for Schnorr operations are available in the literature, so we use a heuristic to approximate it: we scale the energy costs of the two BLS algorithms by the fraction of time their Schnorr counterparts to run — this arguably gives only a rough estimate, but which is sufficient for our purposes. BLS signing needs 97x more time than Schnorr signing, whereas BLS verification takes thrice the time of Schnorr verification, so we extrapolate that producing a single Schnorr signature needs 0.52mJ and verifying a single Schnorr signature expends 37.53mJ of energy. Using the same heuristic, we deduce that aggregating 2 BLS signatures requires 0.28mJ, whereas aggregating 2 BLS public keys requires 0.56mJ.

Moving on to energy consumption by disks, a typical SSD has an average read/write speed of 0.7GB/s and consumes 7W on average [47]. This amounts to around 1nJ per read/written bit. We assume that the entire SSD energy consumption goes to reads and writes, i.e., the idle energy is negligible.

Last but not least, we discuss the energy consumption of network operations. We assume the use of IEEE 802.11 Wi-Fi, for which highly accurate energy consumption estimations of data reception are available [27, 40, 50, 52]. According to [27], sending always costs less energy than receiving, so we conservatively set sending energy equal to receiving energy. By interpolating the energy consumption for sending a packet of sizes 100B and 1500B as calculated in [52], we get that the per-bit energy for transmitting (i.e., including both sending and receiving) x bytes is $68,571.4/x + 14.2$nJ — our result agrees with [40]. To minimise energy consumption, we use the maximum permitted packet size (2,304B [2]), so the energy consumption per transmitted bit is set to 43.96nJ.

In order to calculate the total transmissions, we arrange the nodes in a lattice, matching the onboard topology of containers. Based on the dimensions of the resulting parallelepiped, we then calculate the average number of hops needed for a single multi-hop message through our simulation and we assume that each party transmits an equal share of these hops. We find the total number of multi-hop messages by counting the number of messages needed for every leader of a fully honest ship to receive attestations for all its blocks by all attestors. Our simulator is parameterised by the number of parties and the number of blocks. Both the energy consumption and the storage requirements of each node are linear in the two parameters. As a concrete example, we simulated an 1-month-long trip of the largest container ship ($|\mathcal{P}| = 24,346$), in which every container produces one block every minute ($t = 43,200$). The average per-party energy consumption in this scenario is approximately 244,571J, or 1.79 power banks of 10,000mAh each. With today's energy prices in the UK [12], this amounts to 0.020\$ of energy-related overheads per party due to our protocol. For completeness, this scenario would need a per-party total storage of approximately 15MiB.

We remark that our methodology might result in discrepancies with energy requirements in real-world systems since the latter suffer from inefficiencies, noise from invalidated caches, branch prediction misses, packet loss, and other unpredictable parameters. Still, we opted for this approach since it gives informative order-of-magnitude estimations and because the energy consumption of real-world deployments depends strongly on the concrete hardware stack used, which is liable to change rapidly as the product evolves.

5.3 Comparison with Tendermint

In order to put the efficiency of our protocol in context, we compare our results with storing sensor data in a single Tendermint [15] blockchain instead of having one nodechain per party. We measure the overhead of Tendermint in terms of storage and energy consumption using the same heuristics leveraged in the evaluation of our protocol. We choose to compare with Tendermint because it is one of the most efficient consensus protocols and widely used in industry.

In brief, Tendermint parties (a.k.a. *validators*) need 3 communication rounds to reach consensus on each new block. In the first round (“propose”), each party disseminates one piece of signed sensor data (i.e., one transaction) to its counterparties, which the latter verify ($|\mathcal{P}|$ Schnorr signatures, $|\mathcal{P}|^2$ Schnorr verifications, and at least $|\mathcal{P}|^2$ transmissions). A single party (the block *leader*) then signs and sends the proposed block to everyone else (1 Schnorr signature and at least $|\mathcal{P}|$ transmissions).

The second Tendermint round (“pre-vote”) has every validator verify the signature on the proposed block and pre-vote it. Pre-voting involves signing the proposed block and sending it to every other validator, which amounts to a total of $|\mathcal{P}|$ new Schnorr signatures, at least $|\mathcal{P}|^2$ transmissions, and $|\mathcal{P}|^2$ verifications of Schnorr signatures.

A validator reaches the third round (“pre-commit”) upon receiving the same pre-vote from more than two-thirds of the validators. The validator then signs and disseminates its pre-commit to everyone else. Each receiver verifies each pre-commit and finally commits the block upon receiving valid pre-commits by more than two-thirds of the validators. This round requires $|\mathcal{P}|$ new Schnorr signatures, at least $|\mathcal{P}|^2$ transmissions, and $|\mathcal{P}|^2$ verifications of Schnorr signatures.

Name	Size (bytes)	Comment
Header	267	See Table 5
Txs	sensor reading size $+32(\lvert\mathcal{P}\rvert-1)$	Own data + others’ hash
LastCommit	$75+68\lvert\mathcal{P}\rvert$	See Table 3
Evidence	0	No infractions when all honest

Table 2. Tendermint block size

In order to make the comparison as fair as possible, we have applied a number of optimisations to Tendermint. First of all, nodes only store hashes of other parties’ data, not the data itself. Secondly, we omit from each Tendermint block any data that would not be used by our protocol. Moreover, we use 4-byte-long Tendermint Indexes to identify validators (instead of 20-byte-long addresses). Lastly, the height of each block is represented with 3 bytes (instead of the standard 4 bytes). We refer the reader to Tables 2–6 for more details on the Tendermint data structures used.

Name	Size (bytes)	Comment
Height	3	
Round	4	
BlockID	68	See Table 5

Table 3. Tendermint commit size

Name	Size (bytes)	Comment
Pubkey	$32\lvert\mathcal{P}\rvert$	Parties’ Ed25519 keys
ProposerPriority	8	Implementation detail
PrivateKey	32	

Table 4. Tendermint validator size

Name	Size (bytes)	Comment
Height	3	
LastBlockID	68	hash - 32B + PartsHeader - 36B
LastCommitHash	32	
DataHash	32	
ValidatorsHash	32	
NextValidatorsHash	32	
ConsensusHash	32	
LastResultHash	32	
ProposerIndex	4	Fits $\lvert\mathcal{P}\rvert = 50,000$

Table 5. Tendermint header size

In a simulation with parameters matching the ones discussed above (i.e., $|\mathcal{P}| = 24,346, t = 43,200$), the Tendermint-based protocol requires a per-party storage of 97.00GiB and energy consumption of $1,920,303$J. In other words, our protocol is 6179x more efficient in terms of storage and 7.85x more efficient in terms of energy overheads. The costs (denominated in USD) of various configurations are shown in Table 7.

Name	Size (bytes)	Comment
Type	1	prevote or precommit?
Height	3	
BlockID	68	See Table 5
ValidatorIndex	4	
Signature	64	

Table 6. Tendermint pre-vote & pre-commit sizes

5.4 Optimisations

Over the course of the performance simulation implementation, we identified a number of helpful optimisations that can be also used in a production deployment. To begin with, raw block data are never signed, transmitted, nor stored by attestors; hashes of the block data are used instead. This greatly enhances privacy and performance. In fact, since the block contents never leave the node, each party can add an unlimited

t \ $\lvert\mathcal{P}\rvert$		$3,875$	$24,346$	$50,000$
$43,200$	N	0.01	0.03	0.05
	T	11.92	82.61	169.48
$259,200$	N	0.05	0.16	0.28
	T	75.80	497.36	1,017.71
$1,555,200$	N	0.30	0.93	1.71
	T	458.19	2,984.16	6,106.29

Table 7. Per-party costs of our protocol and Tendermint in USD for various nodechain lengths t and number of parties $|\mathcal{P}|$, assuming energy costs 0.30\$/kWh and storage costs 0.85\$/GB. "N" is for "NodeChain", "T" for "Tendermint". Our protocol is more than 3 orders of magnitude more cost-efficient.

amount of data in each block (bounded only by the party's data production rate and its computational ability to hash it). As a result, our protocol imposes no limit to the per-second rate of adding data to a chain, whereas other blockchains have a severely limited rate (e.g., Bitcoin supports 7 transactions per second [20]).

As discussed earlier, leaders sign their blocks with Schnorr signatures, whereas attestations are created with BLS signatures. This keeps leader signatures cheap to produce and verify, while enabling aggregation of attestations, reducing each leader's per-block overhead from $44|\mathcal{P}|$B down to just 144B and helping achieve the $O(|\mathcal{P}|+t)$ performance requirement.

Great energy savings can be attained by transmitting large network packets. Buffering multiple smaller messages to build a single large packet will likely be needed.

A technique that greatly improves performance is *incremental signature aggregation*. When an intermediate node receives two or more attestations for the same block, it aggregates them before sending them on towards the leader. This vastly reduces the overall messages needed. In order to eliminate the cost of aggregating public keys, intermediate nodes can furthermore pre-agree on only aggregating attestations from specific neighbours. This removes the necessity of transmitting bitmaps of the parties of which the signatures are contained in a specific aggregated signature, but makes the protocol more susceptible to attacks by malicious intermediaries. Last but not least, intermediate nodes do not validate the received attestations before aggregating it — only the leader eventually validates the final aggregate attestation before storing it. This optimisation notably reduces the number of BLS validations, which is the most expensive operation, but increases the effects that malicious attestors can cause. If these aggregation-related optimisations are to be adopted, less optimised fallbacks with intermediate signature verifications, or even without any aggregation, need to be available. This will maintain high efficiency in favourable conditions, while adaptively minimising the effect of potential attacks that such optimisations enable in case of detected misbehaviour.

6 Wireless Network Resilience

Wireless communications are much less reliable than wired ones. Indeed, wireless communication can be interrupted by environmental factors like electromagnetic interference, or by malicious parties that employ radio jamming to disrupt communication. Unfortunately, our main application, i.e., the logging of sensor data by containers aboard a ship, precludes wired configurations due to the high cost of manually interconnecting them or fitting them with automatic connectivity features. In this section we investigate through simulation the resilience of our protocol against practical malicious jamming.

More concretely, we define resilience as the ability of the network to uphold assumptions 1–4. As we saw earlier, assumptions 2 and 4 depend on the exact location of jamming nodes in the network graph. It is unconventional and costly for a container owner to fix its containers' physical locations on board, as the shipowner customarily rearranges containers when docking to optimise loading times. Apart from strategically fixing the location of corrupted containers, the attacker has to intelligently decide how strong an antenna to fit each one with, how much energy each jamming node needs in total, and, crucially, exactly when, what, and how powerfully should each jamming antenna transmit — many of these choices are very sensitive to the exact configuration of the low-level communication protocol used.

In light of the above, it is exceedingly difficult to analytically classify the types of attacks that successfully break resilience. We therefore opt for simulating prominent attack scenarios and measuring their success rate at breaking resilience for a varying number of jammers and jamming power.

In particular, we explore two types of jamming attacks, namely *saturation* and *partition* attacks. The former models an attacker with a number of containers randomly positioned throughout the vessel (Fig. 3), whereas the latter randomly places adversarial nodes only on a single bay, thus maximising the probability of partitioning the network into two separate components (Fig. 4). Note that partition attacks are harder to orchestrate given the aforementioned container rearrangement. In both attacks, all jamming nodes transmit constantly at the same power. Multiple access techniques [36] are not considered, i.e., there exists a single communication channel. (Equivalently, the jammer knows perfectly when and how to jam parties' channels.) Our simulations support any container size, number, and spatial 3D configuration. We here choose to focus on 40-foot containers, which is the most popular size in the industry and results in transceivers being spatially farther apart than 20-foot containers, thus making the network less reliable. Our simulation results are the average of simulating the attacks of interest in a number of common container spatial configurations [44, Sec. 5.4.3a] with bay, row, and layer numbers ranging from 1 to 25.

Furthermore, in order to match the usual honesty assump-

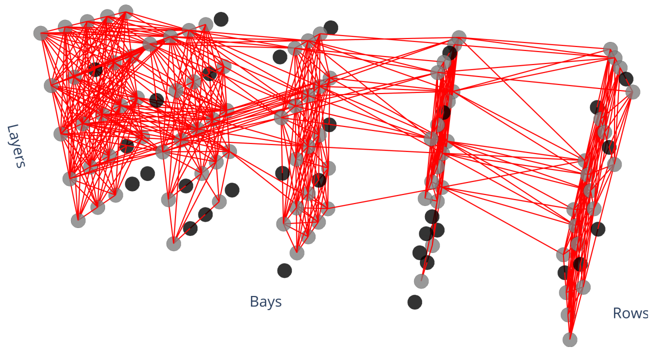

Figure 3. Network under saturation attack by the black nodes.

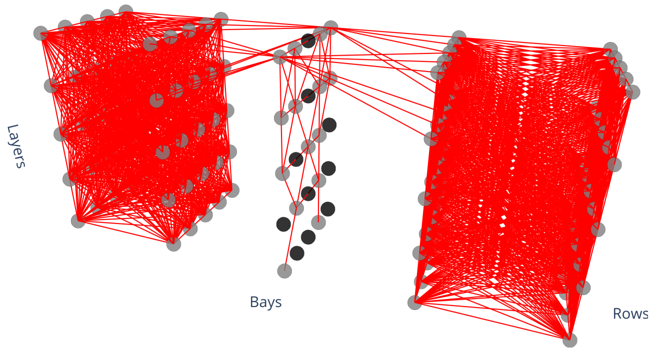

Figure 4. Network under partition attack by the black nodes.

tion in the literature and to take advantage of the inherent resilience benefits of decentralisation, our simulations focus on the *decentralised trustsets* configuration (Subsec. 2.2), which corresponds to a block being valid if any two-thirds of the participants attest to it. This makes our simulation results relevant to deployments using traditional blockchains as well.

Our simulation adopts the free-space path loss wireless communication model, i.e., the waves are assumed to propagate in a vacuum and they get weaker farther from their source, following an inverse-square relationship. This choice has been made as the path loss characteristics of a container-based environment are poorly researched and uniquely challenging [4]. It is thus essential to experimentally test the resilience characteristics of a practical deployment with actual hardware before commercial use. All honest parties transmit at 0dBm, while we vary the relative adversarial jamming power.

We use the Cooja simulation framework [37], a mature, robust, and highly configurable tool for simulating wireless networks in detail. Each *mote* (Cooja node) runs the Contiki operating system [24]. In order to isolate the effects of active malicious jamming from unintended jamming due to legitimate but overlapping uses of the network, a single, randomly chosen party disseminates a single block in each simulation.

The results of our simulation of the saturation and partition attacks can be seen in Fig. 5 and 6 respectively. An honest signal is not delivered to its intended recipient if its power is lower than the sum of all jamming signals at the location of the recipient. In the two figures, "saturation" means preventing all honest nodes from collecting sufficient attestations, whereas "partition" corresponds to separating honest parties into multiple disconnected components.

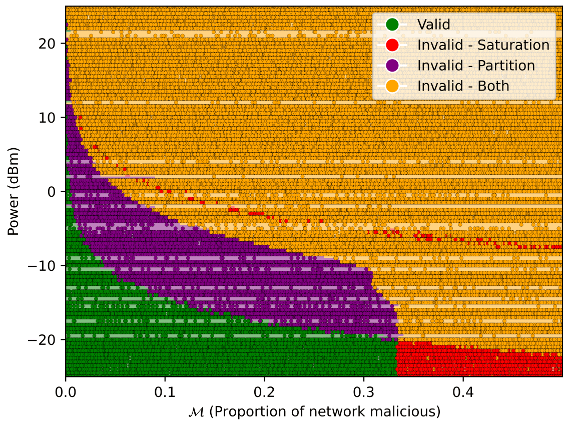

Figure 5. Saturation attack success for varying levels of jamming power and proportion of adversarial parties. Corrupted nodes are uniformly distributed throughout the vessel. All corrupted nodes jam with the same power.

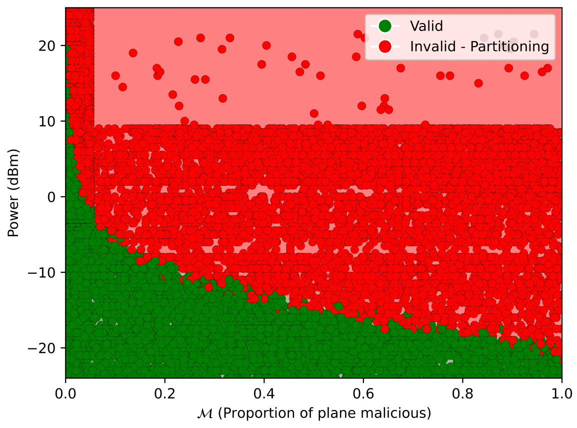

Figure 6. Partition attack success for varying levels of jamming power and proportion of adversarial parties. Corrupted nodes are uniformly distributed in the middle bay. All corrupted nodes jam with the same power.

We observe that saturation attacks can cause both saturation and partitions, whereas partition attacks cannot induce saturation. Let us now discuss specific observations drawn from our results. Saturation attacks (Fig. 5) succeed if more than 1% of the total nodes jam with power equal to that of honest signals, or if more than 10% of the total nodes jam with power equal to 5% of honest signals. In parallel, partition attacks (Fig. 6) succeed if more than 4% of the nodes in a single bay jam with power equal to that of honest signals, or if more than 20% of the nodes in a single bay jam with power equal to 10% of honest signals. This is more robust than expected, especially given the lack of multiple-access communication, as a considerable number of containers need to be malicious for a jamming attack to succeed.

We can glean some useful lessons from this simulation. Firstly, it is imperative to use a multiple-access communication protocol (i.e., a protocol with multiple channels), otherwise jamming is relatively cheap and even honest parties may cause congestion by themselves. Furthermore, in order to make the network more resilient to partition attacks, we recommend fitting two antennas at the two far ends of each container instead of only one.

7 Future Work

7.1 Extensions to Reliability and Functionality

Our protocol forms a cryptographic core which can be extended with a number of security and functionality improvements in its surrounding execution environment. First of all, our integrity guarantees do not ensure that the logged data correspond to the physical reality, only that they have not been tampered with after being logged. Since each container is physically sealed by its owner, any sensors found inside are under the full control of the container owner. As such, it is possible for a malicious container owner to, e.g., modify the logic on its heat sensor before it is installed so that, at a defined point in the journey, it will ignore its real measurements and instead log fabricated temperatures consistent with a fire. Cryptographic techniques alone cannot guard against such attacks.

A potential solution that guarantees the veracity of stored data is using sensors operated by a commonly trusted third party — this introduces an undesirable central point of failure and requires the existence of an entity that is prepared to take on this role and the potential liability it carries. An alternative solution is fitting each container with multiple redundant sensors, each controlled by a different party, and have these sensors agree on each measurement by, e.g., majority vote — this requires the owners of all sensors to be present when the container is sealed to ensure no sensor is tampered with, further increasing installation and operating costs.

Our protocol can be extended with additional mechanisms for corroborating the veracity of data produced by certain types of sensors. For example, a node that stores location data can periodically add authenticated, timestamped location data from other nodes to its own nodechain, or even signals from reputable antennas ashore when they happen to be in range. As a further example, unusually high temperature measurements can be configured to prompt nearby containers to activate their own temperature sensors if inactive and share their measurements with the affected node. The additional data points can be presented to a judge as evidence that the container's own measurements are truthful.

Beyond enhancing data security, additional practical measures can be taken to reinforce uninterrupted network connectivity. Future work can investigate communication protocols that minimise the success of jamming attempts, as well as techniques to hold jammers accountable.

In a communication protocol with multiple access [36] (i.e., multiple channels), two parties that use predictable communication channels (such as a fixed frequency) can be *selectively jammed* by $\mathcal{A}$ (e.g., $\mathcal{A}$ can emit white noise at this specific frequency). Selective jamming is less energy-intensive for $\mathcal{A}$ compared to jamming a wide range of channels. One approach that disrupts attempts at selective jamming involves designing a bespoke communication protocol that hides from the adversary the channels used for direct messages. For example, each pair of neighbouring nodes can agree on a common, secret randomness seed before the protocol starts and use it to periodically switch to a new pseudorandom channel. This approach must guard against accidental overlap of channels, for example by swiftly hopping to a new random channel if congestion is identified. Cognitive radio approaches [49] can also increase resilience.

Regarding jammer accountability, honest nodes can use triangulation techniques to identify the source of jamming and impose penalties. Any such method must of course guard against abuse and spurious accusations by malicious parties.

Moving on, differently to other blockchain protocols, our system does not require data replication across nodes. This however means that the only log of the data is inside the container itself. To increase redundancy, some protocol participants (i.e., other containers or the ship) can store the entirety of other parties' nodechains, possibly in exchange for money. To further enhance redundancy, nodechain snapshots can also be stored on a best-effort basis outside the ship. Options include transmission via satellite, to antennas in range when docking or close to coast, or to other ships in proximity.

The wireless network that our protocol runs on can be exploited for other useful functions. For example, it can be used to alert the ship's crew in case of actionable emergencies, e.g., power delivery failure to a *reefer* (i.e., a refrigerator container), or fire.

7.2 Security Refinements and Generalisations

Our formalisation does not currently model changes to the set of containers and their topology, which however do take place whenever a container ship docks. We leave as future work adapting our protocol to support changes to the network graph without needing initialisation from scratch.

Our formal model does not have provision for connectivity gaps, i.e., for periods during which parties cannot collect sufficient attestations. This can happen in practice if the connectivity assumptions (2 and 4) are temporarily broken. In this case, an attestor might receive multiple blocks by a leader after a potentially long delay. In such an event (and given that it has not detected any forks), the attestor should only attest to the last block, thus showing that the leader might have tampered with the contents of these blocks. However, since the leader cannot tamper with data across attestations, the attestor can confidently continue providing attestations after the network is restored (cf. Appx. A). An extension of our model can generalise our results to these weaker assumptions and provide robust liveness guarantees even under temporary network splits.

A single constant, t_{rep}, parametrises our security model. Practical deployments must tune this parameter to a concrete value, which should be done via practical experimentation in the field and reasonable predictions on the extent of jamming attacks. Lower t_{rep} values lessen the per-leader storage overhead of each attestor and reduce the number of blocks that the judge has to trim from the end of a to-be-verified nodechain (i.e., a lower t_{rep} reduces wasted blocks), but strengthen assumptions 2 and 4; higher values reverse these trends.

7.3 Alternative Building Blocks

Employing STARKs [10], SNARKs [28] or similar cryptographic succinct proof primitives can help reduce asymptotic storage requirements from $O(|\mathcal{P}|+t)$ down to $O(\log(|\mathcal{P}|)+t)$ or even to $O(t)$. Further investigation in this direction is needed, as it is possible that any asymptotic gains could be eliminated in practice by the high constant cost of such approaches.

The size of a BLS signature in the used library [48] is 144B, but it can be reduced to 96B [25, Sec. 2.9.1]. BLS aggregation can be leveraged even more aggresively to further reduce storage requirements for attestations, ideally down to a single BLS signature per nodechain that attests to all blocks by all attestors. In particular, it is possible to aggregate signatures of different messages, albeit more communication is needed [13]. We leave this investigation as future work.

There exists a middle ground between our protocol and the vanilla single-blockchain protocol with which we compared our work in Sec. 5. In particular, we believe it is possible to design a single-blockchain system that contains all sensor measurements, in which each party stores only its own data, along with block headers and (e.g., Merkle) proofs of inclusion of its data in each block. Still, we believe that our design is more efficient in all fronts, provides sufficient security for an array of relevant applications, as well as offers increased resilience against network issues and the flexibility of trustsets instead of requiring asynchronous communication and rigid consensus quorums. We however leave a more in-depth specification of and comparison with such a system as future work.

We have considered using Trusted Execution Environments (TEEs [45]) as an alternative approach to securely providing the desired functionality. We have however rejected this approach because of the need to trust the TEE manufacturer, as well as due to the lack of TEEs in commercial low-powered devices such as Raspberry Pis. We leave an in-depth exploration of TEE-based solutions as future work.

8 Conclusion

In this work we propose NodeChain, a novel blockchain-based mechanism that guarantees the integrity of data produced by a distributed set of participants in an online fashion, i.e., that data producers cannot tamper with the data after they are stored (Sec. 3). We provide a formal specification and analysis (Appx. D–F), a Proof-of-Concept implementation (Sec. 4), a simulation of performance at scale (Sec. 5), and a simulation of the resilience of the protocol under network jamming attacks (Sec. 6).

The motivating application is the continuous storage of sensor data produced by containers aboard a cargo ship. Such data are instrumental in adjudicating insurance claims in case of damaged cargo. Guaranteeing data integrity has the potential to drastically reduce the scope of deceitful claims by container owners as well as the ability of rogue shipowners to dispute valid claims, thus lessening the oversight required by the insurance industry, which can in turn lead to substantially lower insurance premiums. Gradual adoption is possible, unlike past attempts at integrating blockchains in container shipping [34], which failed due to a lack of network effect.

We formalise and prove the security of our protocol in the UC framework [17] (Appx. C) under concrete assumptions (Appx. B), as well as its crucial security properties: *integrity* and *liveness*. Our network setting allows for more disruptions than the state-of-the-art asynchronous communication model, while our corruption setting is based on the novel concept of *trustsets*, which is more flexible than the usual two-thirds honesty assumption.

Our design sacrifices consensus, which is not required in our application, in favour of efficiency. This allows our protocol to outperform off-the-shelf, single-blockchain solutions by more than 3 orders of magnitude in terms of storage requirements and by approximately a factor of 8 in terms of required computations (Sec. 5), as well as more than 1000x cost reduction. Indeed, our protocol is efficient enough to be deployed on storage- and energy-constrained devices.

Our resilience simulation (Sec. 6) shows that our protocol can withstand a non-negligible level of attacks. Still, multiple access techniques [36] are required for robustness.

Ethics Considerations

We here list ethical considerations of our protocol. We roughly order implications in order of the magnitude of their effects.

Our protocol aspires to simplify and automate part of the insurance claim procedure for owners of shipping containers. As such, and similarly to many other novel technologies, it can cause loss of existing job positions. Gradual, planned, openly communicated deployment of this technology can help mitigate such effects. Furthermore, the deployment and maintenance of a practical implementation of our system will likely also create new job opportunities, albeit not necessarily of commensurate volume.

Compared to the local, non-integrity-protected storage of sensor readings, our protocol requires additional energy expenditure for network messages and cryptographic operations. Nevertheless, the automation of tasks can cause a commensurate or even potentially greater reduction in energy and other environmentally relevant expenses (e.g., office space overheads, paper) due to less need for manual oversight and intervention.

Practical applications of our protocol must ensure compliance with international shipping laws and safety regulations, as well as data privacy concerns (e.g., GDPR) of sharing sensor data with other containers. Only gossiping hashes of block data instead of their contents (as discussed in Subsec. 5.4) aids data privacy. Our industrial partner is in the process of gaining safety approval for operating the devices aboard container ships.

Our research does not involve any private data, experiments involving live subjects, or vulnerability discovery. All software developed over the course of our research is open source and complies with the licences of all libraries and tools used.

Open Science

Apart from the present document, our research consists of the following:

- Code for the Proof of Concept implementation (Sec. 4), accompanied by a video that demonstrates its successful deployment on Raspberry Pis along with the nodechains' resilience to temporary network splits,
- Code for the performance simulation (Sec. 5),
- Code for the resilience simulation (Sec. 6).

In the interest of Open Science, all aforementioned artifacts will be made available.

References

[1] B.a.t.m.a.n. advanced. `https://www.open-mesh.org/projects/batman-adv/wiki/Wiki`. Accessed: 2025-11-1.

[2] Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pages 1–4379, 2021.

[3] Omar Alkhoori, Abduraouf Hassan, Omar Almansoori, Mazin Debe, Khaled Salah, Raja Jayaraman, Junaid Arshad, and Muhammad Habib Ur Rehman. Design and implementation of cryptocargo: A blockchain-powered smart shipping container for vaccine distribution. *IEEE Access*, 9:53786–53803, 2021.

[4] Slawomir J. Ambroziak and Ryszard J. Katulski. An empirical propagation model for mobile radio links in container terminal environment. *IEEE Transactions on Vehicular Technology*, 62(9):4276–4287, 2013.

[5] Ignacio Amores-Sesar, Christian Cachin, and Enrico Tedeschi. When is spring coming? A security analysis of avalanche consensus. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems, OPODIS 2022, December 13-15, 2022, Brussels, Belgium*, volume 253 of *LIPIcs*, pages 10:1–10:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery.

[7] Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan, and Luca Henzen. *The Hash Function BLAKE*. Information Security and Cryptography. Springer, 2014.

[8] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. *J. Cryptol.*, 37(2):18, 2024.

[9] Arati Baliga, I Subhod, Pandurang Kamat, and Siddhartha Chatterjee. Performance evaluation of the quorum blockchain platform, 2018.

[10] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018.

[11] Martin Biely, Ulrich Schmid, and Bettina Weiss. Synchronous consensus under hybrid process and link failures. *Theor. Comput. Sci.*, 412(40):5602–5630, 2011.

[12] Paul Bolton. Gas and electricity prices during the 'energy crisis' and beyond. `https://commonslibrary.parliament.uk/research-briefings/cbp-9714/`, 2024.

[13] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, 2003.

[14] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[15] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. Master's thesis, University of Guelph, Guelph, Ontario, Canada, June 2016.

[16] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[17] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14–17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

[18] Ran Cohen, Jack Doerner, Eysa Lee, Anna Lysyanskaya, and Lawrence Roy. An unstoppable ideal functionality for signatures and a modular analysis of the dolev-strong broadcast. Cryptology ePrint Archive, Paper 2024/1807, 2024.

[19] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 26–38, 2020.

[20] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[21] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 34–50, New York, NY, USA, 2022. Association for Computing Machinery.

[22] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1085–1100, New York, NY, USA, 2017. Association for Computing Machinery.

[23] Danny Dolev. The byzantine generals strike again. *J. Algorithms*, 3(1):14–30, 1982.

[24] Adam Dunkels. Contiki: Bringing ip to sensor networks. `https://ercim-news.ercim.eu/en76/rd/contiki-bringing-ip-to-sensor-networks`, 2009.

[25] Ben Edgington. Upgrading ethereum. `https://eth2book.info/capella/part2/building_blocks/signatures/#signing`, 2023.

[26] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 51–68, New York, NY, USA, 2017. Association for Computing Machinery.

[27] Karina Gomez, Roberto Riggio, Tinku Rasheed, and Fabrizio Granelli. Analysing the energy consumption behaviour of wifi networks. In *2011 IEEE Online Conference on Green Communications*, pages 98–104, 2011.

[28] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[29] Runchao Han, Vincent Gramoli, and Xiwei Xu. Evaluating blockchains for iot. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2018.

[30] Runchao Han, Gary Shapiro, Vincent Gramoli, and Xiwei Xu. On the performance of distributed ledgers for internet of things. *Internet of Things*, 10:100087, 2020. Special Issue of the Elsevier IoT Journal on Blockchain Applications in IoT Environments.

[31] Ling Li and Honggeng Zhou. A survey of blockchain with applications in maritime and shipping industry. *Information Systems and e-Business Management*, 19(3):789–807, September 2021.

[32] Feng Lian, Jiaru Jin, and zhong Yang. Optimal container ship size: a global cost minimization approach. *Maritime Policy & Management*, 46:1–16, 06 2019.

[33] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:112, 2017.

[34] Maersk. A.p. moller - maersk and ibm to discontinue tradelens, a blockchain-enabled global trade platform. https://www.maersk.com/news/articles/2022/11/29/maersk-and-ibm-to-discontinue-tradelens, 2022.

[35] Maersk. A short guide on ocean freight transit times. https://www.maersk.com/logistics-explained/transportation-and-freight/2023/09/27/sea-freight-guide, 2023.

[36] Harshita Mathur and T. Deepa. A survey on advanced multiple access techniques for 5g and beyond wireless communications. *Wirel. Pers. Commun.*, 118(2):1775–1792, May 2021.

[37] Tayyab Mehmood. Cooja network simulator: Exploring the infinite possible ways to compute the performance metrics of iot based smart devices to understand the working of iot based compression & routing protocols, 2017.

[38] Henrique Moniz, Nuno Ferreira Neves, and Miguel Correia. Turquois: Byzantine consensus in wireless ad hoc networks. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 537–546, 2010.

[39] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[40] Claro Noda, Shashi Prabh, Mário Alves, and Thiemo Voigt. On packet size and error correction optimisations in low-power wireless networks. In *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, pages 212–220, 2013.

[41] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 643–673, Cham, 2017. Springer International Publishing.

[42] Helena Rifà-Pous and Jordi Herrera-Joancomartí. Cryptographic energy costs are assumable in ad hoc networks. *IEICE Trans. Inf. Syst.*, 92-D(5):1194–1196, 2009.

[43] Helena Rifà-Pous and Jordi Herrera-Joancomartí. Computational and energy costs of cryptographic algorithms on handheld devices. *Future Internet*, 3(1):31–48, 2011.

[44] Jean-Paul Rodrigue. *The Geography of Transport Systems*. Routledge, New York, 6 edition, 2024.

[45] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64, 2015.

[46] Claus P. Schnorr. Method for identifying subscribers and for generating and verifying electronic signatures in a data exchange system. US Patent 4995082, 1989.

[47] Arman Shehabi, Sarah Josephine Smith, Dale Sartor, Richard Brown, Magnus K. Herrlin, Jonathan Koomey, Eric Masanet, Nathaniel Horner, Inês M. L. Azevedo, and William Lintner. United states data center energy usage report. 2016.

[48] Mitsunari Shigeo. Bls threshold signature. https://github.com/herumi/bls, 2017.

[49] Carl R. Stevenson, Gerald Chouinard, Zhongding Lei, Wendong Hu, Stephen J. Shellhammer, and Winston Caldwell. Ieee 802.22: The first cognitive radio wireless regional area network standard. *IEEE Communications Magazine*, 47(1):130–138, 2009.

[50] Li Sun, Ramanujan K. Sheshadri, Wei Zheng, and Dimitrios Koutsonikolas. Modeling wifi active power/energy consumption in smartphones. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 41–51, 2014.

[51] Ye Wang and Roger Wattenhofer. Asynchronous byzantine agreement in incomplete networks. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 178–188. ACM, 2020.

[52] Ninad Warty, Ramanujan K. Sheshadri, Wei Zheng, and Dimitrios Koutsonikolas. A first look at 802.11n power consumption in smartphones. In *Proceedings of the First ACM International Workshop on Practical Issues and Applications in next Generation Wireless Networks*, PINGEN '12, page 27–32, New York, NY, USA, 2012. Association for Computing Machinery.

[53] B. Weiss and U. Schmid. Consensus with written messages under link faults. In *Proceedings 20th IEEE Symposium on Reliable Distributed Systems*, pages 194–197, 2001.

[54] Sam Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William Knottenbelt. Sok: Decentralized finance (defi). In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, page 30–46, New York, NY, USA, 2023. Association for Computing Machinery.

[55] Kyrill Winkler. *Characterization of Consensus Solvability under Message Adversaries*. PhD thesis, TU Wien, Karlsplatz 13, 1040 Vienna, August 2019.

[56] Kyrill Winkler, Ulrich Schmid, and Yoram Moses. A characterization of consensus solvability for closed message adversaries. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, *23rd International Conference on Principles of Distributed Systems, OPODIS 2019, December 17-19, 2019, Neuchâtel, Switzerland*, volume 153 of *LIPIcs*, pages 17:1–17:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[57] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.

[58] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 347–356, New York, NY, USA, 2019. Association for Computing Machinery.

[59] Jun Zhao, Jing Tang, Zengxiang Li, Huaxiong Wang, Kwok-Yan Lam, and Kaiping Xue. An analysis of blockchain consistency in asynchronous networks: Deriving a neat bound. In *40th International Conference on Distributed Computing Systems*, ICDCS, pages 179–189, New York, NY, 2020. IEEE.

A Handling Connectivity Gaps

In practice, each attestor stores a received block b of height larger than expected for at most t_{rep} rounds. Each leader only sends each new block when asking for attestations (as described in Sec. 3 and unlike Π_{Ledgers} of Appx. D.3 in which the whole chain is sent). If the attestor receives all blocks between the latest it has attested and b within t_{rep} rounds, then it attests to all blocks and stores b. Otherwise, in case of a connectivity gap (discussed in Sec. 7), the attestor deletes unconnected blocks (to avoid Denial-of-Service attacks against its storage space), asks for their retransmission if the network recovers, and only attests to the most recent block (thus refraining from attesting to blocks that might have been tampered with during the connectivity gap). This practical approach achieves the same level of security under exactly the same assumptions as Π_{Ledgers}, but we decided to prove security with the approach in Π_{Ledgers} to simplify the exposition.

B Honesty & Connectivity Assumptions

Definition 1 (Communication graphs)**.** *Let $G = (\mathcal{P}, E)$ be the* non-jammed graph*, i.e., the graph that has the parties as nodes and a directed edge from a node to another for each direct channel in the communication network when no jamming occurs. Let $(G_i = (\mathcal{P}, E_i))_{i \in \mathbb{N}}$ be the sequence of subgraphs of G (the* communication graphs*) such that for all i, the edges of G_i match the communication links that are not jammed by $\mathcal{A}$ at round i.*

Definition 2 ($A \to B$ timewise path)**.** *Let G be a non-jammed graph, G_i be communication graphs, $A, B \in \mathcal{P}$, and $t_1 < t_2 \in \mathbb{N}$. A $A \to B$* timewise path *is a sequence of edges $(e_i)_{i \in \{t_1, \ldots, t_2\}}$ such that $\forall i \in \{t_1, \ldots, t_2\}, e_i \in E_i$ and $\bigcup_{i=t_1}^{t_2} e_i \subset E$ is a path from A to B in E.*

Assumptions

1. *Per-trustset existential honesty:* Every trustset contains at least one honest party. ($\forall T \in \mathcal{T}, \exists c \in T : c \in \mathcal{H}$)
2. *Malicious connectivity:* Every malicious party is connected to at least one honest party in each trustset (see assumption 1) at least once every t_{rep} rounds. ($\forall P \in \mathcal{P} \setminus \mathcal{H} \forall t \in \mathbb{N}, \exists t' \in \{t, \ldots, t + t_{\text{rep}}\} : \forall T \in \mathcal{T} \exists c \in T \cap \mathcal{H} : \exists$ timewise path $P \to c = (e_i)_{i \in \{t, \ldots, \cdot\}}$ with $|P \to c| \leq t_{\max}$)
3. *Fully honest trustset:* There exists at least one trustset consisting exclusively of honest parties. ($\exists T \in \mathcal{T} : \forall c \in T, c \in \mathcal{H}$)
4. *Honest connectivity:* Every honest party is roundtrip-connected to at least one fully honest trustset (see assumption 3) at least once every t_{rep} rounds. ($\forall P \in \mathcal{H} \forall t \in \mathbb{N}, \exists t' \in \{t, \ldots, t + t_{\text{rep}}\} : \exists T \in \mathcal{T} : \forall c \in T, c \in \mathcal{H} \wedge \exists$ timewise path $P \to c \to P = (e_i)_{i \in \{t', \ldots, \cdot\}}$ with $|P \to c \to P| \leq t_{\max}$)

Figure 7. Honesty & Connectivity Assumptions

C Universal Composition

According to the UC framework [17], which is based on the simulation-based cryptographic paradigm [33], there are two executions of interest: the *real-world* and the *ideal-world* executions. Both executions consist of a number of *instances* of *interactive Turing machines*.

In the real world execution, there is one environment machine $\mathcal{E}$, one adversary machine $\mathcal{A}$ and one machine per party P; the set of parties $\mathcal{P}$ is a parameter of the execution and is static (i.e., does not change in the course of the execution). All parties $P \in \mathcal{P}$ begin their execution as honest parties, executing Π_{Ledgers}. At any point during one of its activations, $\mathcal{A}$ can corrupt any $P \in \mathcal{P}$ by sending a single "corruption" message to P.

In the ideal world execution, there is one environment machine $\mathcal{E}$, one simulator machine $\mathcal{S}$ and one functionality machine $\mathcal{F}_{\text{Ledgers}}$ (Fig. 9).

In both worlds the execution is sequential, i.e., only one machine is executing at any instance. It starts from $\mathcal{E}$, which gives control to other machines by sending input messages. Parties can communicate among themselves only via the network functionality $\mathcal{F}_{\text{Net}}$ (Fig. 8), which models the point-to-point, jammable network aboard a container ship: On every round, $\mathcal{A}$ specifies which channels are being jammed. A message that uses a jammed channel is dropped. $\mathcal{F}_{\text{Net}}$ also models the synchronous time.

$\mathcal{E}$ can at any point output a binary value, at which point the execution completes. Let $\text{EXEC}^{\mathcal{F}_{\text{Net}}}_{\Pi_{\text{Ledgers}},\mathcal{A},\mathcal{E}}$ be the binary random variable (over the local random coins of all involved machines) that describes the output of $\mathcal{E}$ with adversary $\mathcal{A}$ and honest parties executing Π_{Ledgers} and having access to the ideal functionality $\mathcal{F}_{\text{Net}}$. We say that the two worlds are *indistinguishable* if

$$\exists n_0 \in \mathbb{N} : \forall n \geq n_0, \forall \text{ PPT } \mathcal{A}, \exists \text{ PPT } \mathcal{S} : \forall \text{ PPT } \mathcal{E},$$
$$\Pr[\text{EXEC}^{\mathcal{F}_{\text{Net}}}_{\Pi_{\text{Ledgers}},\mathcal{A},\mathcal{E}} = 1] - \Pr[\text{EXEC}^{\mathcal{F}_{\text{Net}}}_{\mathcal{F}_{\text{Ledgers}},\mathcal{S},\mathcal{E}} = 1] < \text{negl}(n)$$

where $\text{negl}(n)$ is a negligible function in n, i.e., $\forall d \in \mathbb{N}, \exists n_0 \in \mathbb{N} : \forall n \geq n_0,\ \text{negl}\ (n) < n^{-d}$. For simplicity, the above is also written as $\text{EXEC}^{\mathcal{F}_{\text{Net}}}_{\Pi_{\text{Ledgers}},\mathcal{A},\mathcal{E}} \approx \text{EXEC}^{\mathcal{F}_{\text{Net}}}_{\mathcal{F}_{\text{Ledgers}},\mathcal{S},\mathcal{E}}$. Observe that (due to the order of quantification) $\mathcal{S}$ may depend on $\mathcal{A}$.

D Ideal-World Functionalities & Real-World Protocols

D.1 $\mathcal{F}_{\text{Net}}$

Functionality $\mathcal{F}_{\text{Net}}$

```
Initially delivered = $\emptyset$, $t = \bot$ // The latter shows that $\mathcal{F}_{\text{Net}}$ is uninitialised
1:  On first (INIT, $(G = (\mathcal{P}, E), \mathcal{T}, t_{\text{rep}})$) by $\mathcal{A}$:
2:      for all $P \in \mathcal{P}$ do ignored$(P) \leftarrow \emptyset$
3:      $t \leftarrow 0$
4:      send (INIT-DONE) to $\mathcal{A}$
5:  Ignore any other message received before the above

6:  On (CORRUPT, $P \in \mathcal{H}$) by $\mathcal{A}$:
7:      $\mathcal{H} \leftarrow \mathcal{H} \setminus \{P\}$ // remove $P$ from honest party set
8:      if assumption 1 or 3 of Fig. 7 is violated then halt
9:      send (CORRUPT-DONE) to $\mathcal{A}$

10: On (JAM, $e \in E$) by $\mathcal{A}$:
11:     $E \leftarrow E \setminus \{e\}$ // remove jammed edge
12:     On reaching the next round, restore $e$ ($E \leftarrow E \cup \{e\}$)
13:     send (JAM-DONE) to $\mathcal{A}$

14: On (IGNORE, $R$) by $P$:
15:     add $R$ to ignored$(P)$

16: On (MULTICAST, $x$) by $P \in \mathcal{P}$: // flood message for $t_{\text{rep}}$ rounds
17:     ensure we have not received another multicast by $P$ at round $t$
18:     generate unique message id mid // uniformly random, retry on collision
19:     $K_{\text{mid}} \leftarrow \{P\}$ // those who have received the message
20:     repeat on the first activation of the next $t_{\text{rep}}$ rounds: // Runs after handling the activation message. Other messages can be served normally while awaiting in ll. 27 and 31.
21:         $K'_{\text{mid}} \leftarrow \emptyset$ // those who just received the message
22:         for all $R \in K_{\text{mid}}$ do
23:             if $P \in$ ignored$(R)$ then continue with next iteration of loop of l. 22
24:             for all $S \in N(R)$ do // $R$'s neighbours at current round
25:                 $b \leftarrow$ False
26:                 if $R \notin \mathcal{H}$ then // if $R$ corrupted, ask $\mathcal{A}$ whether to relay
27:                     send (RELAY, mid, $R$, $S$) to $\mathcal{A}$, await reply (RELAY, mid, $b$)
28:                 end if
29:                 if $R \in \mathcal{H} \lor b$ then // if $R$ is honest or $\mathcal{A}$ agreed,
30:                     if $S \notin K_{\text{mid}}$ then // relay to $S$ if not relayed before
31:                         await (STEP) by $\mathcal{A}$, then add $(\text{mid}, R)$ to delivered and output $(x, P)$ to $R$ // if many multicast messages are awaiting a (STEP) by $\mathcal{A}$, serve the lexicographically first
32:                         $K'_{\text{mid}} \leftarrow K'_{\text{mid}} \cup \{S\}$
33:                     end if
34:                 end if
35:             end for
36:         end for
37:         $K_{\text{mid}} \leftarrow K_{\text{mid}} \cup K'_{\text{mid}}$
38:     send (MULTICAST, $x$, $P$, mid) to $\mathcal{A}$

39: On (NEXT-ROUND) by $\mathcal{A}$:
40:     if we have executed all loops of l. 22 that have been installed by l. 20 during the past $t_{\text{rep}}$ rounds then // $\mathcal{A}$ has prompted with (STEP) the relaying of every message that each honest party should relay during the current round
41:         $t \leftarrow t + 1$
42:         send (NEXT-ROUND-YES) to $\mathcal{A}$
43:     else
```

```
44:         send (NEXT-ROUND-NO) to $\mathcal{A}$
45:     end if

46: On (GET-DELIVERED) by $\mathcal{A}$:
47:     send (DELIVERED, delivered) to $\mathcal{A}$

48: On (GET-ROUND) by $P$:
49:     return $t$

50: On (GET-TRUSTSETS) by $P$:
51:     return $\mathcal{T}$
```

Figure 8. Graph-based network functionality

D.2 $\mathcal{F}_{\text{Ledgers}}$

```
Functionality $\mathcal{F}_{\text{Ledgers}}$
$L'$ and $\mathcal{R}'$ are temporary variables, i.e., they are destroyed at the end of their scope.
1: On (INIT, $P$) by $\mathcal{A}$:
2:     $\text{INIT}_P \leftarrow T$

3: On first (INIT, $(G = (\mathcal{P}, E), \mathcal{T}, t_{\text{rep}})$) by $\mathcal{A}$:
4:     $t \leftarrow 0$
5:     $\mathcal{H} \leftarrow \mathcal{P}$ // All parties are initially honest
6:     for all $P \in \mathcal{H}$ do $\mathcal{L}_P \leftarrow [(\varepsilon, \varepsilon), \{\}]$ // Parties' ledgers are initially empty
7: Ignore any of the below messages received before the above

8: On (CORRUPT, $P \in \mathcal{H}$) by $\mathcal{A}$:
9:     $\mathcal{H} \leftarrow \mathcal{H} \setminus \{P\}$ // remove $P$ from honest party set
10:    if assumption 1 or 3 of Fig. 7 is violated then halt
11:    relay messages between $P$ and $\mathcal{A}$ for the remaining execution

12: On (JAM, $e \in E$) by $\mathcal{A}$:
13:    $E \leftarrow E \setminus \{e\}$ // remove jammed edge
14:    On reaching the next round, restore $e$ ($E \leftarrow E \cup \{e\}$)

15: On (NEXT-ROUND) by $\mathcal{A}$:
16:    $t \leftarrow t + 1$
17:    send (NEXT-ROUND-OK) to $\mathcal{A}$

18: On (GET-ROUND) to $\mathcal{F}_{\text{Net}}$ by $P \in \mathcal{P} \setminus \mathcal{H}$:
19:    return $t$

20: On (SUBMIT, $x$) by $P \in \mathcal{H}$:
21:    ensure $\text{INIT}_P = T$
22:    ensure we have not received another submission by $P$ at round $t$
23:    append $((x,t), \{\})$ to $P$'s chain // $\mathcal{L}_P \leftarrow \mathcal{L}_P || ((x,t), \{\})$
24:    after installing the next two reactions, send (SUBMITTED, $x$, $P$) to $\mathcal{A}$
25:    On the first (ATTESTED, $(x,t)$, $P$, $R \in \mathcal{P} \setminus \{P\}$) by $\mathcal{A}$: // reaction that triggers at most once for each distinct set of parameters
26:        append $R$ to entry $x$ in $P$'s chain // $((x,t), S) \leftarrow ((x,t), S \cup \{R\})$
27:    On reaching round $t + t_{\text{rep}}$:
28:        if $\nexists T \in \mathcal{T} : \forall c \in T, c \in S$, where $((x,t), S) \in \mathcal{L}_P$ then
29:            // No trustset attested $x$, assumption 4 of Fig. 7 failed
30:            mark $x$ entry in $P$'s chain as unattested
31:        end if

32: On (SUBMITTED, $L$, $P \in \mathcal{P} \setminus \mathcal{H}$) by $\mathcal{A}$:
33:    if $\mathcal{L}_P.\mathsf{map}(((x,t), \_) \mapsto (x,t)) \preccurlyeq L$ then
34:        append $L[|\mathcal{L}_P| + 1 : -1].\mathsf{map}((x,t) \mapsto ((x,t), \{\}))$ to $\mathcal{L}_P$
35:    end if
36:    send (NOTED) to $\mathcal{A}$

37: On $(x, R, P \in \mathcal{P} \setminus \mathcal{H})$ by $\mathcal{A}$:
38:    output $(x, R)$ to $P$ as $\mathcal{F}_{\text{Net}}$

39: On (READ) by $P \in \mathcal{H}$:
40:    ensure $\text{INIT}_P = T$
41:    send (READ, $P$) to $\mathcal{A}$ and assign reply to $(L, \mathcal{R})$
42:    $L' \leftarrow L.\mathsf{map}(((x,t), (\_, \_)) \mapsto (x,t))$
43:    $\mathcal{R}' \leftarrow \mathcal{R}.\mathsf{map}(S \mapsto S.\mathsf{map}((R, \_) \mapsto R))$
44:    ensure $L' = \mathcal{L}_P.\mathsf{map}(((x,t), \_) \mapsto (x,t)) \wedge \mathcal{R}' = \mathcal{L}_P.\mathsf{map}((\_, S) \mapsto S)$
45:    return $\mathsf{zip}(L, \mathcal{R})$

46: On (JUDGE, $L$, $R$) by $P \in \mathcal{H}$:
47:    ensure $\text{INIT}_P = T$
48:    $L' \leftarrow L.\mathsf{map}((((x,t), (\_, \_)), \_) \mapsto (x,t))$
49:    if $L'[:-t_{\text{rep}}] \not\preccurlyeq \mathcal{L}_R.\mathsf{map}(((x,t), \_) \mapsto (x,t))$ then return (BAD, $L$, $R$)
50:    if any $\mathcal{L}_R[:|L|]$ entry is unattested (l. 30) then return (BAD, $L$, $R$)
51:    send (JUDGE, $L$, $R$, $P$) to $\mathcal{A}$ and return its response
```

Figure 9. Ledgers functionality

D.3 Π_{Ledgers}

Remark. Differently to the description of Sec. 3, in Fig. 10 each leader sends its whole nodechain on every new block, while each attestor stores every leader's entire nodechain and compares it with each received, to-be-attested nodechain. This is inefficient because the whole nodechain is sent on each block, but it trivially avoids the issue of blocks being lost due to network failures (an issue from which the description of Sec. 3 is not protected, and thus needs stronger network assumptions to achieve liveness). Still, both approaches guarantee integrity under the same assumptions.

Protocol Π_{Ledgers}

P (self) keeps track of one nodechain $\mathcal{L}_R$ for each honest party $R \in \mathcal{H}$. A nodechain is a list of blocks, which are (data, previous hash, signature) triples. Additionally, P keeps track of other parties' attestations to its own blocks in a list of sets of signatures $\mathcal{R}$. Lists are 0-indexed. The genesis block is excluded from the attestation procedure. $\mathcal{T}$ is a temporary variable, i.e., it is destroyed at the end of its scope. The initialisation and key generation with $\mathcal{F}_{\text{Sig}}$ is implicit.
"On x by $R \in \mathcal{H}/\mathcal{P}$" is a shorthand for "On output $(x, R \in \mathcal{H}/\mathcal{P})$ by $\mathcal{F}_{\text{Net}}$". In the context of such messages, **return** passes the execution token to $\mathcal{F}_{\text{Net}}$ via input.

```
1: On first (INIT, $(G = (\mathcal{P}, E), ((pk_R, \sigma_R^0))_{R \in \mathcal{P}})$) by $\mathcal{A}$:
2:     $\mathcal{H} \leftarrow \mathcal{P}$ // All parties are initially honest
3:     for all $R \in \mathcal{P}$ do if $\neg\mathsf{Verify}(pk_R, \varepsilon, \sigma_R^0)$ then $\mathcal{H} \leftarrow \mathcal{H} \setminus \{R\}$
4:     for all $R \in \mathcal{H}$ do $\mathcal{L}_R \leftarrow [((\varepsilon, \varepsilon), (\varepsilon, \sigma_R^0))]$ // At first ledgers only have genesis block
5:     $\mathcal{R} \leftarrow [\{\}, \{\}, \dots]$ // Infinite list of empty sets: no attestations received yet

6: On (SUBMIT, $x$) by $\mathcal{E}$:
7:     ensure we have received (INIT, _) by $\mathcal{A}$ // Fig. 10, l. 1
8:     send (GET-ROUND) to $\mathcal{F}_{\text{Net}}$ and assign reply to $t$
9:     ensure $t \neq \bot$
10:    ensure we have not received another submission by $\mathcal{E}$ at round $t$
11:    $\mathsf{prevhash} \leftarrow \mathsf{Hash}(\mathcal{L}_P[-1])$
12:    $\mathsf{block} \leftarrow ((x, t), (\mathsf{prevhash}, \mathsf{Sign}(sk_P, \mathsf{prevhash}||x||t)))$
13:    $\mathcal{L}_P \leftarrow \mathcal{L}_P||\mathsf{block}$
14:    input (MULTICAST, (ATTEST, $\mathcal{L}_P$)) to $\mathcal{F}_{\text{Net}}$

15: On (ATTEST, $L$) by $R \in \mathcal{H}$:
16:    for $i \in \{1, \dots, |L|\}$ do // Check for chain validity
17:        $((x, t), (\mathsf{prevhash}, \sigma)) \leftarrow L[i]$
18:        ensure $\mathsf{prevhash} = \mathsf{Hash}(L[i-1]) \land \mathsf{Verify}(pk_R, \mathsf{prevhash}||x||t, \sigma)$
19:        // $\mathcal{A}$ may fabricate invalid chains, $R$ not necessarily corrupt
20:    end for
21:    if $\mathcal{L}_R \not\preccurlyeq L \land L \not\preccurlyeq \mathcal{L}_R$ then // $R$ equivocated
22:        $\mathcal{H} \leftarrow \mathcal{H} \setminus \{R\}$ // mark $R$ as corrupt
23:        input (IGNORE, $R$) to $\mathcal{F}_{\text{Net}}$
24:    end if
25:    else // $L$ is a valid chain
26:        if $|L| > |\mathcal{L}_R|$ then $\mathcal{L}_R \leftarrow L$
27:        input (MULTICAST, (ATTESTED, $R$, $|L|, \mathsf{Sign}(sk_P, L[-1])$)) to $\mathcal{F}_{\text{Net}}$

28: On (ATTESTED, $P$, $h$, $\sigma$) by $R \in \mathcal{P}$:
29:    ensure $0 < h \leq |\mathcal{L}_P| \land \mathsf{Verify}(pk_R, \mathcal{L}_P[h], \sigma)$
30:    $\mathcal{R}[h] \leftarrow \mathcal{R}[h] \cup \{(R, \sigma)\}$ // new attestation valid, store it at right height

31: On (READ) by $\mathcal{E}$:
32:    ensure we have received (INIT, _) by $\mathcal{A}$ // Fig. 10, l. 1
33:    send (GET-ROUND) to $\mathcal{F}_{\text{Net}}$ and ensure reply is not $\bot$
34:    return $\mathsf{zip}(\mathcal{L}_P, \mathcal{R})$ // return list of (block, attestation set) pairs

35: On (JUDGE, $L$, $R$) by $\mathcal{E}$:
36:    ensure we have received (INIT, _) by $\mathcal{A}$ // Fig. 10, l. 1
37:    send (GET-ROUND) to $\mathcal{F}_{\text{Net}}$ and ensure reply is not $\bot$
38:    if $L[0][0] \neq ((\varepsilon, \varepsilon), (\varepsilon, \sigma_R^0)) \lor \neg\mathsf{Verify}(pk_R, \varepsilon, \sigma_R^0) \lor L[0][1] \neq \{\}$ then
39:        return (BAD, $L$, $R$)
40:    end if
41:    for $i$ from 1 to $|L| - 1$ (inclusive) do
42:        parse $L[i]$ as $(((x, t), (\mathsf{prevhash}, \sigma_R)), A)$, return (BAD, $L$, $R$) on failure
43:        if $\mathsf{Hash}(L[i-1][0]) \neq \mathsf{prevhash} \lor \neg\mathsf{Verify}(pk_R, \mathsf{prevhash}||x||t, \sigma_R)$ then
44:            return (BAD, $L$, $R$)
45:        end if
46:        for $(Q, \sigma_Q) \in A$ do
47:            if $\neg\mathsf{Verify}(pk_Q, ((x, t), (\mathsf{prevhash}, \sigma_R)), \sigma_Q)$ then return (BAD, $L$, $R$)
48:        end for
49:        send (TRUSTSETS) to $\mathcal{F}_{\text{Net}}$ and assign reply to $\mathcal{T}$
50:        if $\nexists T \in \mathcal{T} : T \subset A.\mathsf{map}((Q, \_) \mapsto Q)$ then
51:            return (BAD, $L$, $R$) // No trustset attested
52:        end if
53:    end for
54:    return (GOOD, $L$, $R$)
```

Figure 10. Ledgers protocol

E Indistinguishability Theorem

Theorem 1. *If assumptions 1, 2, 3, and 4 hold, then the protocol Π_{Ledgers} (Fig. 10) UC-realises the ideal functionality $\mathcal{F}_{\text{Ledgers}}$ (Fig. 9) in the presence of $\mathcal{F}_{\text{Net}}$ (Fig. 8):*

$$\forall \text{ PPT } \mathcal{A}, \exists \text{ PPT } \mathcal{S} : \forall \text{ PPT } \mathcal{E} \text{ it is}$$

$$\text{EXEC}^{\mathcal{F}_{\text{Net}}}_{\Pi_{\text{Ledgers}}, \mathcal{A}, \mathcal{E}} \approx \text{EXEC}^{\mathcal{F}_{\text{Ledgers}}}_{\mathcal{S}, \mathcal{E}} \quad .$$

Theorem 1. We first define in Fig. 11 the simulator $\mathcal{S}$, parametrised by an arbitrary PPT real-world adversary $\mathcal{A}$.

Simulator $\mathcal{S}_{\mathcal{A}}$

Relay all $\mathcal{A} \leftrightarrow \mathcal{E}$ communication throughout the execution. Simulate an $\mathcal{F}_{\text{Net}}$ and an $\mathcal{F}_{\text{Sig}}$ instance internally and relay all (STEP) and (GET-DELIVERED) messages by $\mathcal{A}$ or by a corrupted party to $\mathcal{F}_{\text{Net}}$, as well as all DELIVERED messages by $\mathcal{F}_{\text{Net}}$ to $\mathcal{A}$

or to a corrupted party. "send M to $\mathcal{F}_{\text{Net}}$ as $\mathcal{A}$" is a shorthand for "simulate with internal $\mathcal{F}_{\text{Net}}$ receipt of M on backdoor tape", while "input M to $\mathcal{F}_{\text{Net}}$ as P" is a shorthand for "simulate with internal $\mathcal{F}_{\text{Net}}$ receipt of M on P's input tape". Initially $\mathcal{H} \leftarrow \mathcal{P}$.

```
1:  On $(x, m)$ by $\mathcal{A}$ to $\mathcal{F}_{\text{Net}}$ for $x \in \{\text{INIT}, \text{CORRUPT}, \text{JAM}\}$:
2:      send $(x, m)$ to $\mathcal{F}_{\text{Net}}$ as $\mathcal{A}$ and expect reply ($x$-DONE)
3:      if $x = \text{CORRUPT}$ then
4:          $\mathcal{H} \leftarrow \mathcal{H} \setminus \{P\}$ // remove $P$ from honest party set
5:          relay $P$'s messages between $\mathcal{F}_{\text{Ledgers}}$ and $\mathcal{A}$ for the remaining execution
6:      end if
7:      send $(x, m)$ to $\mathcal{F}_{\text{Ledgers}}$

8:  On (NEXT-ROUND) by $\mathcal{A}$ to $\mathcal{F}_{\text{Net}}$:
9:      send (NEXT-ROUND) to simulated $\mathcal{F}_{\text{Net}}$ as $\mathcal{A}$ and assign reply to $r$
10:     if $r = (\text{NEXT-ROUND-YES})$ then
11:         send (NEXT-ROUND) as $\mathcal{A}$ to $\mathcal{F}_{\text{Ledgers}}$, expect reply (NEXT-ROUND-OK)
12:         send (NEXT-ROUND-YES) to $\mathcal{A}$ as $\mathcal{F}_{\text{Net}}$
13:     else // $r = (\text{NEXT-ROUND-NO})$
14:         send (NEXT-ROUND-NO) to $\mathcal{A}$ as $\mathcal{F}_{\text{Net}}$
15:     end if

16: On first (INIT, $(G = (\mathcal{P}, E), ((pk_R, \sigma^0_R))_{R \in \mathcal{P}})$) by $\mathcal{A}$ to $P$:
17:     start a simulated $\Pi_{\text{Ledgers}}$ instance named $P$ & run ll. 2–5 of Fig. 10 with it, sending (SIG-INIT-OK) to it upon receiving (INIT) by the simulated $\mathcal{F}_{\text{Sig}}$
18:     send (INIT, $P$) to $\mathcal{F}_{\text{Ledgers}}$

19: On (SUBMITTED, $x$, $P$) by $\mathcal{F}_{\text{Ledgers}}$:
20:     // on SUBMIT by $P$, $\mathcal{F}_{\text{Ledgers}}$ sends SUBMITTED to $\mathcal{A}$ only if $P$ is active (Fig. 9, l. 21, as well as Fig. 11, ll. 16–18 and Fig. 9, ll. 1–2)
21:     simulate $P$ receiving (SUBMIT, $x$) by $\mathcal{E}$

22: On (MULTICAST, $x$, $P$, mid) by $\mathcal{F}_{\text{Net}}$:
23:     if $P \in \mathcal{P} \setminus \mathcal{H} \wedge x = (\text{ATTEST}, L)$ then
24:         $L' \leftarrow L.\mathsf{map}(((x,t),(\_,\_)) \mapsto (x,t))$ // Remove hashes and signatures
25:         send (SUBMITTED, $L'$, $P$) to $\mathcal{F}_{\text{Ledgers}}$ and expect reply (NOTED)
26:     end if
27:     send (MULTICAST, $x$, $P$, mid) to $\mathcal{A}$

28: On output $(x, R \in \mathcal{P})$ to $P \in \mathcal{P}$ by $\mathcal{F}_{\text{Net}}$:
29:     if $P \in \mathcal{H}$ then
30:         simulate $P$ receiving output $(x, R)$ by $\mathcal{F}_{\text{Net}}$
31:         if $x = (\text{ATTESTED}, P, i, \_) \wedge$ l. 29, Fig. 10 during l. 30 succeeded then
32:             parse $\mathcal{L}_P[i]$ of simulated $P$ as $((x,t),\_)$
33:             send (ATTESTED, $(x,t)$, $P$, $R$) to $\mathcal{F}_{\text{Ledgers}}$
34:         end if
35:     else // $P \in \mathcal{P} \setminus \mathcal{H}$
36:         send $(x, R, P)$ to $\mathcal{F}_{\text{Ledgers}}$
37:     end if

38: On (READ, $P$) by $\mathcal{F}_{\text{Ledgers}}$:
39:     return $(\mathcal{L}_P, \mathcal{R})$ as stored in the simulated $P$

40: On (JUDGE, $L$, $R$, $P \in \mathcal{H}$) by $\mathcal{F}_{\text{Ledgers}}$:
41:     simulate $P$ receiving (JUDGE, $L$, $R$) and send output to $\mathcal{F}_{\text{Ledgers}}$
```

Figure 11. Simulator for the proof of Theorem 1

We have to show that, under any sequence of messages by $\mathcal{E}$, the outputs to $\mathcal{A}$ and $\mathcal{E}$ from $\mathcal{S}$ and $\mathcal{F}_{\text{Ledgers}}$ respectively in the ideal world on the one hand and from the parties executing Π_{Ledgers} and the hybrid $\mathcal{F}_{\text{Net}}$ on the other are computationally indistinguishable. We will prove this by induction in the number of SUBMIT messages sent by $\mathcal{E}$ to some honest party and not ignored due to lack of initialisation (l. 5 in Fig. 8, ll. 7 and 21 in Fig. 9, and ll. 7, 9, 33, and 37 in Fig. 10).

We first observe that both $\mathcal{F}_{\text{Net}}$ in the real and $\mathcal{F}_{\text{Ledgers}}$ in the ideal world are initialised as a result of the same message due to ll. 1–7 of $\mathcal{S}$ and that all other messages by $\mathcal{E}$ to any machine are ignored before the initialisation (l. 5 in Fig. 8, l. 7 in Fig. 9, and ll. 9, 33, and 37 in Fig. 10). Also messages of ll. 15 and 28 in Fig. 10 cannot be sent by $\mathcal{F}_{\text{Net}}$ before initialisation due to l. 5 in Fig. 8. Then, neither $\mathcal{F}_{\text{Net}}$ nor $\mathcal{F}_{\text{Ledgers}}$ halt on initialisation, thus the initialisation procedure do not give any opportunity for distinguishing.

Next, the initialisation of each party happens in exactly the same way in both worlds with respect to the party (Fig. 10, ll. 2–5 in the real world and Fig. 11, l. 17 with the party being simulated by $\mathcal{S}$ in the ideal world). Fig. 11, l. 18 and Fig. 9, l. 2 further ensure that $\mathcal{F}_{\text{Ledgers}}$ is informed of the party initialisation immediately in the ideal world. What is more, a (READ) message by $\mathcal{E}$ to P before P has received any SUBMIT message (which is the only message that modifies $\mathcal{L}_P$) either does not produce a response if any of the party, $\mathcal{F}_{\text{Net}}$, or $\mathcal{F}_{\text{Ledgers}}$ is not initialised, or else returns $[((\varepsilon,\varepsilon),(\varepsilon,\sigma^0_P)),\{\}]$ in both worlds (Fig. 10, ll. 32–33 in the real and Fig. 9, l. 45 in the ideal world). The above serve as the induction base of the overall proof.

For the induction hypothesis, assume that $\mathcal{E}$ has sent $k \in \mathbb{N}$ SUBMIT messages to honest parties that have not been ignored due to lack of initialisation, as well as any number of other messages in any order, along with any number of messages by $\mathcal{A}$, and that the two worlds are still indistinguishable. Observe that, due to the logic of ll. 9–11 of Fig. 11, ll. 39–42 of Fig. 8, and ll. 15–17 of Fig. 9, the round number t of $\mathcal{F}_{\text{Net}}$ and $\mathcal{F}_{\text{Ledgers}}$ are always the same in the ideal world. Since $\mathcal{E}$ can learn the round via a (GET-ROUND) message to $\mathcal{F}_{\text{Net}}$ (which is handled by $\mathcal{F}_{\text{Ledgers}}$ in the ideal world as per l. 18 of Fig. 9 and by $\mathcal{F}_{\text{Net}}$ in the real world as per l. 48 of Fig. 8), the round in the ideal world must match that of the $\mathcal{F}_{\text{Net}}$ round in the real world to ensure indistinguishability.

We will perform the inductive step by first treating a SUBMIT message by $\mathcal{E}$ to an initialised honest party and then considering any message from $\mathcal{A}$ or $\mathcal{E}$ apart from SUBMIT. In both steps, we will prove that the respective message does not introduce an opportunity to distinguish the two worlds. We treat SUBMIT separately because it is the only message by $\mathcal{E}$ (besides initialisation, which has already been treated in the induction base) that causes a change to the state.

When $\mathcal{E}$ sends (SUBMIT, x) to an honest, initialised party P, as we discussed earlier either $\mathcal{F}_{\text{Net}}$ is initialized in both worlds or uninitialized in both worlds, therefore the real world will proceed to l. 10 of Fig. 10, whereas the ideal world will proceed to l. 22 of Fig. 9. Both checks will either succeed or fail consistently: If one failed and the other succeeded and since the exact same SUBMIT messages are being received by the same party in both worlds, that would imply that the round had advanced in the latter but not in the former before the SUBMIT message. This however would in turn imply a distinguishability opportunity if $\mathcal{E}$ had sent (GET-ROUND) to $\mathcal{F}_{\text{Net}}$ via a corrupted party before the SUBMIT message, contradicting the induction hypothesis.

Then, in the real world, P's $\mathcal{L}_P$ is extended by $((x,\, t),\, (\mathsf{Hash}(\mathcal{L}_P[-1]),\, \mathsf{Sign}(sk_P,\, \mathsf{Hash}(\mathcal{L}_P)[-1] \| x \| t)))$ and (MULTICAST, (ATTEST, $\mathcal{L}_P$)) is input to $\mathcal{F}_{\text{Net}}$ (ll. 11–14, Fig. 10). In the ideal world, $\mathcal{L}_P$ is extended by $((x,t),\{\})$ and (SUBMITTED, x, P) is sent to $\mathcal{S}$, who simulates receiving (SUBMIT, x) with P by $\mathcal{E}$, thus bringing the simulated P at the same state as P in the real world as per ll. 19–21 of Fig. 11, except for $\mathsf{prevhash}$ and the signature, which however follow the same distribution as in the real world due to the hash function being represented by a random oracle and the security of the signature scheme respectively — this will be useful to argue indistinguishability later. This simulation causes P to input (MULTICAST, (ATTEST, $\mathcal{L}_P$)) to $\mathcal{F}_{\text{Net}}$ as in the real world.

Subsequently, in both worlds $\mathcal{F}_{\text{Net}}$ prepares to flood the network with the multicast ATTEST message for t_{rep} rounds (ll. 18–37, Fig. 8) and then sends a receipt of the message to $\mathcal{A}$ (l. 38, Fig. 8) which in the ideal world is expected by $\mathcal{S}$ and forwarded to $\mathcal{A}$ (header, Fig. 11). By observing that the receipt contains in both cases the ledger which, compared to before the induction step, has been extended with the new submission and only differs in the hash and signature fields in an indistinguishable manner, and that the mid generated by $\mathcal{F}_{\text{Net}}$ is uniformly random, we conclude that this last message does not create a distingushability opportunity. We also observe that the multicast message will not start being delivered until the next round in both worlds, ensuring that there is no $\mathcal{E}$-$\mathcal{A}$ activation during which $\mathcal{F}_{\text{Net}}$ is ready to deliver the message in one world while it is not ready for delivery of the message in the other world — this observation will be useful later.

When $\mathcal{F}_{\text{Net}}$ outputs any message to a corrupted party P, in the real world it arrives immediately to P (who in this case is under the control of the real-world $\mathcal{A}$). Likewise, in the ideal world $\mathcal{S}$ sends it to $\mathcal{F}_{\text{Ledgers}}$ (ll. 28 & 35–36, Fig. 11), who in turn outputs it directly to P (ll. 37–38, Fig. 9). Note that the corrupted parties stored in $\mathcal{S}$ and $\mathcal{F}_{\text{Ledgers}}$ coincide (ll. 2–4 of Fig. 11 and ll. 8–9 of Fig. 9) therefore the $P \in \mathcal{P} \setminus \mathcal{H}$ condition of l. 35, Fig. 11 and l. 37, Fig. 9 always agree.

We treat now the two messages that an honest party P may receive by another party (via $\mathcal{F}_{\text{Net}}$), namely ATTEST and ATTESTED. Consider first P receiving (ATTEST, L) by R in the real world (l. 15, Fig. 10). If the chain is invalid, then the message is ignored (l. 19, Fig. 10). If it is valid but forks the previously stored chain of R, then R is marked as corrupt (l. 22, Fig. 10) and its messages are not relayed anymore (l. 23, Fig. 10). If all checks pass, then the R's chain is updated to L if the latter is longer and P multicasts an attestation to R (ll. 25–27, Fig. 10). In the ideal world, $\mathcal{F}_{\text{Net}}$'s output is intercepted by $\mathcal{S}$ (ll. 28–29, Fig. 11) and the latter proceeds to immediately simulate P receiving the output (ll. 30, Fig. 11), therefore the effects of the real world are precisely simulated in $\mathcal{S}$, including the attestation multicast. The only difference is the exact signature, which however, due to the security of the signature scheme, cannot introduce a distinguishing opportunity, which, since as per the header of Fig. 11, the ideal-world $\mathcal{A}$ immediately receives a receipt of all messages when they are multicast, could be directly exploited. Note also that the output of this message did not alter the state of $\mathcal{F}_{\text{Ledgers}}$ or $\mathcal{S}$ (apart from the simulated parties) in any way.

We now move on to P receiving (ATTESTED, P, h, σ) by R (via $\mathcal{F}_{\text{Net}}$) in the real world (l. 28, Fig. 10). First, the validity of the attestation is verified (l. 29, Fig. 10) and if the check succeeds, it is then stored (l. 30, Fig. 10). On the other hand, in the ideal world, after $\mathcal{S}$ simulates P receiving the message (l. 30, Fig. 11), it informs $\mathcal{F}_{\text{Ledgers}}$ (ll. 31–33, Fig. 11), who in turn stores the attestation (l. 26, Fig. 9). Observe that the only change in state (apart from the simulated parties) is the storage of the attestation in $\mathcal{F}_{\text{Ledgers}}$.

When $\mathcal{A}$ sends (STEP) to $\mathcal{F}_{\text{Net}}$ in either world, it may trigger the latter outputting a message to a party (l. 31, Fig. 8). We have already treated how specific output messages are handled before. We now discuss whether the same message is output in both worlds. Before the induction step, due to the induction hypothesis, the fact that $\mathcal{A}$ can view the delivered messages using the (GET-DELIVERED) message (ll. 46–47, Fig. 8), and the fact that $\mathcal{A}$ has been informed on every message multicast (l. 38, Fig. 8), the messages awaiting delivery must match in the two worlds (otherwise there would exist a distinguishing opportunity). Furthermore, as we saw above, before the induction step the rounds match in the $\mathcal{F}_{\text{Net}}$ of the real world with the $\mathcal{F}_{\text{Net}}$ of the ideal world and $\mathcal{F}_{\text{Ledgers}}$. Due to these and the fact that all multicast messages are handled at the same time in the two worlds and the delivery order is deterministic, we deduce that every (STEP) message results in delivering the same message in both worlds, therefore the delivered set of $\mathcal{F}_{\text{Net}}$ is always identical in the two worlds, thus giving rise to no distinguishing opportunity.

An INIT message by $\mathcal{A}$ to an uninitialised party does not introduce a distinguishing opportunity as discussed earlier in the context of the induction base, while if the party is initialized then the message is ignored in the real world (l. 1, Fig. 10) and does not change the state of the ideal world (l. 2 of Fig. 9 is idempotent). No response is produced either. Therefore an INIT message does not introduce a distinguishing opportunity in any case.

Regarding READ and JUDGE messages, we see by inspection of the relevant code (ll. 39–51 of Fig. 9 and ll. 40–41 of Fig. 11 for the ideal and ll. 31–54 of Fig. 10 for the real world) that the response to either keeps the state of all machines invariant. Therefore it is sufficient to prove that a single READ or a single JUDGE message does not introduce a distinguishing opportunity — we do not need to consider the number or the ordering of such messages between themselves. It is however important to distinguish whether the message was received before or after every other state-changing message.

We first focus on a (READ) message by $\mathcal{E}$ to P. If the SUBMIT message of the induction step was not addressed to P and P has not received an ATTEST or ATTESTED message, then P's state is exactly as before the induction step (both in the real world and in the simulation of P by $\mathcal{S}$ in the ideal world) and thus the response cannot induce a distinguishing opportunity due to the induction hypothesis.

If P has received an ATTEST message instead and no matter whether it has received any SUBMIT or ATTESTED message, in the real world the only change in its state may be removing parties from the set of honest parties $\mathcal{H}$, which by inspection of Fig. 10 can be seen to be completely independent from the data returned after a (READ) message — independence here meaning that removing parties from $\mathcal{H}$ only changes P's behaviour when receiving a further ATTEST message and in no way influences any of its other actions. As we have discussed before, in the ideal world no state change occurs apart from that of the simulated party. Therefore the data returned in either world coincides with that from before the receipt of the ATTEST message and thus cannot induce a distinguishing opportunity.

If P has received one or no SUBMIT message followed by any number of ATTESTED messages when receiving (READ), then in the real world its state will have changed by the addition of the new block if it has received a SUBMIT message (l. 13, Fig. 10) and by the addition of a single attestor-signature pair per valid ATTESTED message (ll. 28–30, Fig. 10). Likewise in the ideal world, the same block (with potentially different but indistinguishable hash and signature as discussed before) will have been added to the state of the simulated P (l. 21, Fig. 11), as well as the same data without the hash and signature will have been added to $\mathcal{F}_{\text{Ledgers}}$ if $\mathcal{E}$ has sent a SUBMIT message to P (l. 23, Fig. 9), additionally to the same (up to a different, indistinguishable signature) attestor-signature pair in the simulated P (ll. 28–33, Fig. 11) and the attestor in $\mathcal{F}_{\text{Ledgers}}$ (ll. 25–26, Fig. 9) for each valid attestation.

In the real world, the (READ) message will cause P to return the same ledger and attestations it had before handling the SUBMIT message with $\mathcal{L}_P$, potentially augmented by the additional block and any newly received valid attestations as discussed above. In the ideal world, the (READ) message will cause $\mathcal{F}_{\text{Ledgers}}$ to ask $\mathcal{S}$ to provide the latest ledger and attestations (l. 41, Fig. 9), which $\mathcal{S}$ provides as found in the simulated P (ll. 38–39, Fig. 11). $\mathcal{F}_{\text{Ledgers}}$ then makes sure that the provided data match the ones it has locally stored (ll. 42–44, Fig. 9) and, given that these checks succeed, returns exactly the same data as P does in the real world (up to different but indistinguishable hashes and signatures). These checks indeed succeed always: if a (SUBMIT, x) message has been received at round t, the pair (x,t) is added to both the $\mathcal{L}_P$ maintained by $\mathcal{F}_{\text{Ledgers}}$ (l. 23, Fig. 9) and the $\mathcal{L}_P$ maintained by the P simulated by $\mathcal{S}$ (l. 13, Fig. 10) before $\mathcal{E}$ or $\mathcal{A}$ is activated again (thus ensuring that the round t does not change between the two additions). Similarly, for every (ATTESTED, P, h, σ) message sent to P by R that causes P to add an R-signature pair to its attestations, $\mathcal{F}_{\text{Ledgers}}$ adds R to its local attestations store at the appropriate location (ll. 28–33, Fig. 11 and ll. 25–26, Fig. 9). We have thus proven that no (READ) message at any time can induce a distinguishability opportunity.

We now move on to analysing a (JUDGE, L, R) message by $\mathcal{E}$ to P. Once again, we are only concerned with the case of full initialisation. In the real world, the chain L is parsed, the validity of R's initialisation signature on the genesis block and the genesis block structure itself is checked (l. 38, Fig. 10) and each block is checked for a valid signature and inclusion of the correct previous hash (l. 43, Fig. 10). The validity of each attestation is also checked (l. 47, Fig. 10). Lastly, P checks that the attestations for each block amount to attestors from an entire trustset (l. 50, Fig. 10). If any of the aforementioned checks fails, the judgement is negative and BAD is returned. If all checks succeed, the judgement is positive and GOOD is returned (l. 54, Fig. 10).

In the ideal world, $\mathcal{F}_{\text{Ledgers}}$ checks whether the queried chain with the last t_{rep} blocks removed is a prefix of the stored chain and that no block that is common between the queried and the stored chain has been marked as unattested (l. 30, Fig. 9) and returns BAD if either check fails (ll. 48–50, Fig. 9). Otherwise, it asks $\mathcal{S}$ for its judgement and returns it (l. 51, Fig. 9). In turn, $\mathcal{S}$ simulates P and returns its output (ll. 40–41, Fig. 11). Since (as we saw earlier) $\mathcal{F}_{\text{Ledgers}}$ and $\mathcal{S}$ share the same view of corrupted parties, if the query to $\mathcal{S}$ is sent, its response will coincide with that of P in the real world, therefore no distinguishing opportunity arises in this case.

We will now show that whenever $\mathcal{F}_{\text{Ledgers}}$ returns BAD in l. 49 of Fig. 9, then P in the real world returns BAD as well. If $R \in \mathcal{H}$, then $\mathcal{F}_{\text{Ledgers}}$ appends new data to $\mathcal{L}_R$ in l. 23 of Fig. 9. As we saw before, this always leads to the simulated R

multicasting the new $\mathcal{L}_R$. In case $R \in \mathcal{P} \setminus \mathcal{H}$ and due to ll. 22–25 of Fig. 11 and ll. 32–36 of Fig. 9, $\mathcal{F}_{\text{Ledgers}}$ appends new data to $\mathcal{L}_R$ when it is multicast. Therefore in both cases $\mathcal{F}_{\text{Ledgers}}$ updates its view of $\mathcal{L}_R$ between two adversarial activations during which the new data are first multicast.

Let $\hat{L} := L[:-t_{\text{rep}}]$. In order for $\mathcal{F}_{\text{Ledgers}}$ to pass a negative judgement by itself in the aforementioned lines, $\hat{L}$ must either be longer than or a fork of $\mathcal{L}_R$. In either case, consider the entry e_L of minimum index in $\hat{L}$ which does not appear in $\mathcal{L}_R$. Furthermore let $e_{\mathcal{L}}$ be the entry in $\mathcal{L}_R$ that has the same index as e has in L — note that $e_{\mathcal{L}}$ is likewise the entry of minimum index in $\mathcal{L}_R$ which does not appear in L. We will now prove that at least one entry of L has not been attested by any trustset and thus P would also pass negative judgement.

In the first case, $\hat{L}$ is longer than $\mathcal{L}_R$. Since, as we saw, $\mathcal{F}_{\text{Ledgers}}$ is informed of every new entry when it is multicast and adds it to the stored ledger, we deduce that the e_L was never multicast. Due to assumption 1 of Fig. 7, e_L has thus not received attestations from all members of any trustset and therefore L will also receive a negative judgement by P.

In the second case, $\hat{L}$ is a fork of $\mathcal{L}_R$. If e_L was never multicast and given assumption 1 of Fig. 7, no complete trustset can have attested to e_L. In case e_L was multicast, we note that $e_{\mathcal{L}}$ has been multicast, and that honest parties stop attesting for R if faced with individually valid but mutually conflicting entries in $\mathcal{L}_R$ (l. 22, Fig. 10).

Thanks to assumption 2 of Fig. 7 we deduce that at most t_{rep} rounds after e_L is multicast no more attestations for R will be produced in the real world by all the members of any trustset. Furthermore, given that only one multicast per round per party is permitted (l. 17, Fig. 8), we deduce that the longest possible attested ledger belonging to R is at most t_{rep} entries longer than the common prefix of L and $\mathcal{L}_R$. Therefore, if L pruned by t_{rep} entries is not a prefix of $\mathcal{L}_R$, L cannot have collected attestations from any trustset and thus will also receive a negative judgement by P.

We will now show that whenever $\mathcal{F}_{\text{Ledgers}}$ returns BAD in l. 50 of Fig. 9, then P in the real world returns BAD as well. As we have previously seen when discussing the handling of (READ) messages, $\mathcal{F}_{\text{Ledgers}}$ stores exactly the same attestors for each entry as R does in the real world while the latter is honest. If R has become malicious at any moment during the execution, $\mathcal{F}_{\text{Ledgers}}$ does not mark any more of its entries as unattested (as it only marks entries as unattested in l. 30 of Fig. 9, which only runs for honest parties), therefore l. 50 of Fig. 9 will not trigger for such entries and its response will coincide with that of P in the real world. For any of the remaining ledger entries, $\mathcal{F}_{\text{Ledgers}}$ will only mark it as unattested if no trustset has attested it for t_{rep} rounds (ll. 27–30, Fig. 9). We however observe that such an event is impossible under assumption 4 of Fig. 7, as the latter posits that R can communicate with every party of at least one fully honest trustset every t_{rep} rounds, therefore the honest R receives enough attestations by the honest trustset within these rounds. Thus l. 30 of Fig. 9 never runs and thus the response of $\mathcal{F}_{\text{Ledgers}}$ will coincide with that of P in the real world.

We will lastly prove that the remaining messages handled by $\mathcal{F}_{\text{Net}}$ do not introduce an opportunity for distinguishability. By inspection of $\mathcal{S}$ (Fig. 11, header, ll. 1–2, 8–9), we see that all messages by $\mathcal{A}$ to $\mathcal{F}_{\text{Net}}$ (Fig. 8, ll. 1, 6, 10, 39, and 46) are relayed immediately to the simulated $\mathcal{F}_{\text{Net}}$ and its only possible reply (i.e., DELIVERED) is relayed back. Regarding (NEXT-ROUND) by $\mathcal{A}$, as we saw before, $\mathcal{F}_{\text{Net}}$ handles the same STEP messages in the two worlds and thus delivers the same messages in both worlds, thus a (NEXT-ROUND) message succeeds or fails consistently the check of l. 40, Fig. 8 in the two worlds. Therefore a (GET-ROUND) message by $\mathcal{E}$ returns the same value in the ideal (Fig. 9, l. 18) and real (Fig. 8, l. 48) worlds at any time during the induction step. The proof is now complete. □

F Formal Security Properties & Proofs

Theorem 2 (Integrity). *If assumptions 1 and 2 hold, the* Hash *function is one-way and collision-resistant, and the signature scheme is sEUF-CMA secure, then there exists no PPT algorithm $\mathcal{A}$ that, when given access to the history and signing keys of all malicious parties, as well as a transcript of all messages of the execution, can output (L_1, L_2, R) with non-negligible probability such that all the following hold:*

- $|L_1| = |L_2|$,
- *Inputs* (JUDGE, L_1, R) *and* (JUDGE, L_2, R) *by $\mathcal{E}$ to an honest party running* Π_{Ledgers} *produce output* (GOOD, L_1, R) *and* (GOOD, L_2, R) *respectively,*
- $L'_1 \neq L'_2$, *where* $\forall b \in \{1,2\}, L'_b = L_b[:-t_{rep}].\mathsf{map}((((x,t),(\_,\_)),\_) \mapsto (x,t))$.

Theorem 2. We will prove the theorem via contradiction. Assume the existence of a PPT adversary $\mathcal{A}$ that receives the mentioned inputs and then outputs (L_1, L_2, R) which satisfy the mentioned requirements with non-negligible probability. Let e_1 (e_2) be the entry of minimum index i in L'_1 (L'_2) such that $e_1 \neq e_2$. Observe that this is the same index for both L'_1 and L'_2. It cannot be $i = 0$, since, according to l. 38 of Fig. 10, for a GOOD ledger L it must be $L[0][0][0] = (\varepsilon, \varepsilon)$, thus our assumption dictates that $L'_1[0] = L_1[0][0][0] = (\varepsilon, \varepsilon) = L_2[0][0][0] = L'_2[0]$. Thus $i \geq 1$, for which the checks of ll. 46–52 of Fig. 10 must all pass. In particular, this means that two trustsets (that might or might not be different) have attested e_1 and e_2. Since an honest party never attests to two different entries (as can be seen in ll. 19 and 23 of Fig. 10, whereby l. 27 of Fig. 10 is not run for the entry received second) and at least one party per trustset is honest (as per assumption 1), two trustsets with disjoint honest parties have attested to e_1 and e_2. Let T_1 (T_2) $\in \mathcal{T}$ be the trustset that has attested to e_1 (e_2) and $H_1 \subset T_1$ ($H_2 \subset T_2$) its honest members. Due to assumption 1, $H_1, H_2 \neq \emptyset$.

We now observe that an honest party can only attest to entries that have been previously MULTICAST (ll. 15 and 27 of Fig. 10, ll. 16 and 31 of Fig. 8) and that each party (honest or otherwise) can only MULTICAST one entry per round (l. 17 of Fig. 8). Since L_1 and L_2 are GOOD, their last entry has received attestations by honest parties and has thus been MULTICAST by R. Since L'_1 and L'_2 do not contain the last t_{rep} entries of L_1 and L_2 respectively and due to assumption 2, at least one honest party in every trustset has learned both e_1 and e_2 and thus started ignoring R before learning the latest entry of L_1 or L_2. Therefore no trustset has attested to the latest entry of L_1 or L_2, therefore neither of these ledgers is GOOD. We have once again reached a contradiction. □

Theorem 3 (Liveness)**.** *Let assumptions 3 and 4 hold, the* Hash *function be one-way and collision-resistant, and the signature scheme be sEUF-CMA secure. In an execution where $\mathcal{F}_{\text{Net}}$ has been initialised (via an $(\text{INIT}, \_)$ message by $\mathcal{A}$ – l. 8 of Fig. 8), consider an honest party P running Π_{Ledgers} that been initialised (via an $(\text{INIT}, \_)$ message by $\mathcal{A}$ – l. 1 of Fig. 10). P receives input (SUBMIT, x_0) by $\mathcal{E}$ at round t_0 without having received any other input $(\text{SUBMIT}, \_)$ by $\mathcal{E}$ at the same round. Let L be the output of a (READ) message by $\mathcal{E}$ to P at any round $r \geq t_0 + t_{rep}$. Then the following hold:*

- $(x_0, t_0) \in L'$, *where* $L' = L.\mathsf{map}((((x,t),(\_,\_)),\_) \mapsto (x,t))$,
- *Any honest party R running Π_{Ledgers} that has been initialised and receives (JUDGE, L, P) by $\mathcal{E}$ produces output (GOOD, L, P).*

Theorem 3. We first prove that $(x_0, t_0) \in L'$. We observe that P returns its stored $\mathcal{L}_P$ when receiving READ (l. 34, Fig. 10) and that $\mathcal{L}_P$ is only ever extended with new entries (l. 13, Fig. 10) — existing entries are never removed or modified. More specifically, $\mathcal{L}_P$ is extended with $((x_0, t_0), \_)$ when P receives (SUBMIT, x_0) at round t_0 (ll. 12–13, Fig. 10). This proves that $(x_0, t_0) \in L'$.

We subsequently prove that R JUDGEs L as GOOD. In order for this to happen, all checks of ll. 36–52, Fig. 10 must succeed. The check of l. 36 succeeds because of the prerequisite that the judge R is initialised. The check of l. 37 succeeds because of the prerequisite that $\mathcal{F}_{\text{Net}}$ is initialised. The check of l. 38 succeeds since the first entry of $\mathcal{L}_P$ is $((\varepsilon, \varepsilon), (\varepsilon, \sigma_P^0))$ where σ_P^0 is a valid signature on ε (ll. 3–4, Fig. 10) and the first entry of $\mathcal{R}$ is $\{\}$ (ll. 5 and 29–30, Fig. 10). The checks of ll. 41–53 succeed for all entries of L, as P only adds to $\mathcal{L}_P$ (l. 12) and $\mathcal{R}$ (l. 30) entries of the expected (l. 42) format, it makes sure that the hash chain (l. 11) and its own signature (l. 12) are as expected (l. 43), and it has collected attestations by at least one trustset as expected (ll. 46–50). The latter fact holds because P MULTICASTs its new valid ledger entry on submission (l. 14) and, due to assumption 4, a message with the new entry reaches all members of a fully honest trustset within t_{rep} rounds, them being honest they will successfully verify and attest to the new ledger entry (ll. 15–27), and the attestations will reach P (l. 28), who will in turn verify (l. 29) and store them in the appropriate location in $\mathcal{R}$ (l. 30), which forms part of the output of (READ) (l. 34). □