FEDERAL RESERVE BANK
PHILADELPHIA

Working Papers
RESEARCH DEPARTMENT

Blockchain Economics

Joseph Abadi
Federal Reserve Bank of Philadelphia Research Department

Markus Brunnermeier
Princeton University

WP 22-15

PUBLISHED
May 2022

FRB CFI
PHILADELPHIA

ISSN: 1962-5361

DOI: https://doi.org/10.21799/frbp.wp.2022.15

Blockchain Economics*†

Joseph Abadi‡ Markus Brunnermeier§

January 2022

Abstract

The fundamental problem in digital record-keeping is establishing *consensus* on an update to a ledger, e.g., a payment. Consensus must be achieved in the presence of *faults*—situations in which some computers are offline or fail to function appropriately. Traditional centralized record-keeping systems rely on *trust* in a single entity to achieve consensus. Blockchains decentralize record-keeping, dispensing with the need for trust in a single entity, but some instead build a consensus based on the wasteful expenditure of computational resources (proof-of-work). An ideal method of consensus would be tolerant to faults, avoid the waste of computational resources, and be capable of implementing all individually rational transfers of value among agents. We prove a Blockchain Trilemma: any method of consensus, be it centralized or decentralized, must give up (i) fault-tolerance, (ii) resource-efficiency, or (iii) full transferability.

Keywords: Blockchain, Consensus, Cryptocurrency, Mechanism Design, Payments, FinTech
JEL Codes: D80, E42, G00

*We are grateful for insightful discussions by Gur Huberman and Will Cong, helpful comments from Zhiguo He, Rohit Lamba, Stephen Morris, Ulrich Müller, Arvind Narayanan, Jonathan Payne, Wolfgang Pesendorfer, Fahad Saleh, Raphael Auer, conference participants at the Cambridge University Conference on the Economics of Distributed Ledger Technology, the San Francisco Blockchain Week CESC, the CEBRA Annual Meeting, the UCLA Anderson Fink Center Conference on Financial Markets, the Harvard CMSA Blockchain Conference, the AFA Annual Meeting, the NYU Five Star Conference, the University of Chicago Cryptocurrencies and Blockchains Conference, the St. Louis Fed, and the NYU Conference on Financial Intermediation, and seminar participants at the SEC, Yale University, the Wharton School at the University of Pennsylvania, the Princeton University Department of Computer Science, the Princeton University Department of Economics, and the BIS.

‡Federal Reserve Bank of Philadelphia (joseph.abadi@phil.frb.org)
§Princeton University (markus@princeton.edu)

1 Introduction

Trust is of central importance in record-keeping. A ledger represents an agreed-upon history of events and must therefore remain free of tampering or fraud. However, record-keepers are rarely inherently trustworthy, and they often face the temptation to profit by altering the ledger. For record-keepers to be trusted, then, they must be provided with incentives to behave honestly.

Traditionally, record-keeping has been centralized: a single entity is given full power to update the ledger however it desires, and it derives some long-term benefit from its privileged position (e.g., by extracting rents). The record-keeper can be trusted to communicate honestly with the ledger's users, ensuring they remain in agreement about the ledger's contents, only if these long-run incentives are strong enough. Blockchain technology has provided a radical decentralized alternative to record information. Public blockchains seek to minimize reliance on centralized trust: there is no single authority who keeps records. For instance, Bitcoin uses a voting method known as proof-of-work (PoW), in which power to update the ledger is allocated based on the expenditure of computational resources, whereas other blockchains use proof-of-stake (PoS), which instead allocates voting power to token holders. In such systems, voting majorities must be incentivized to behave honestly in order to guarantee consistent record-keeping. Hence, the advent of blockchains raises important new questions regarding the fundamental difficulties in digital record-keeping that decentralization aims to solve.

We take the view that the fundamental problem in digital record-keeping is ensuring agents reach a consensus on updates to a ledger in the presence of faults – situations in which some communication devices, e.g., computers, are offline or do not function appropriately. We therefore study the design of *consensus algorithms*: communication protocols that permit agents with non-faulty communication devices to reach agreement on a desirable outcome. Crucially, in an economic environment, there is no *inherently* trustworthy agent who can be relied upon to truthfully relay information to others: all agents must be incentivized to follow the prescribed protocol. From a design perspective, two main questions arise: In the absence of a trusted mediator, how can record-keeping be designed to incentivize honest reporting? What types of constraints does a lack of trust impose on the types of outcomes that can be implemented?

We posit three ideal features of a consensus algorithm. First, a consensus algorithm should be *fault-tolerant*: it should allow agents who communicate honestly to reach agreement even when some fraction of agents has faulty devices that fail to communicate (or send messages erratically). Faults are fundamental to digital communication: computers may be temporarily disconnected from the network or suffer from programming errors, so any digital record-keeping system must be robust to these considerations. Second, a consensus algorithm should achieve *full transferability*: it should allow agents to reach agreement on *any* individually rational transfer of value. That is, the consensus algorithm should not limit the types of transactions that can be realized, since such limitations would impair the allocative efficiency of the mechanisms the record-keeping system can implement. Third, a consensus algorithm should ideally be *resource-efficient*, meaning it does not force agents to solve intrinsically useless computational problems and incur a deadweight loss

of resources through proof-of-work. We investigate the conditions under which these three ideal features can be achieved by a consensus algorithm that incentivizes honest behavior.

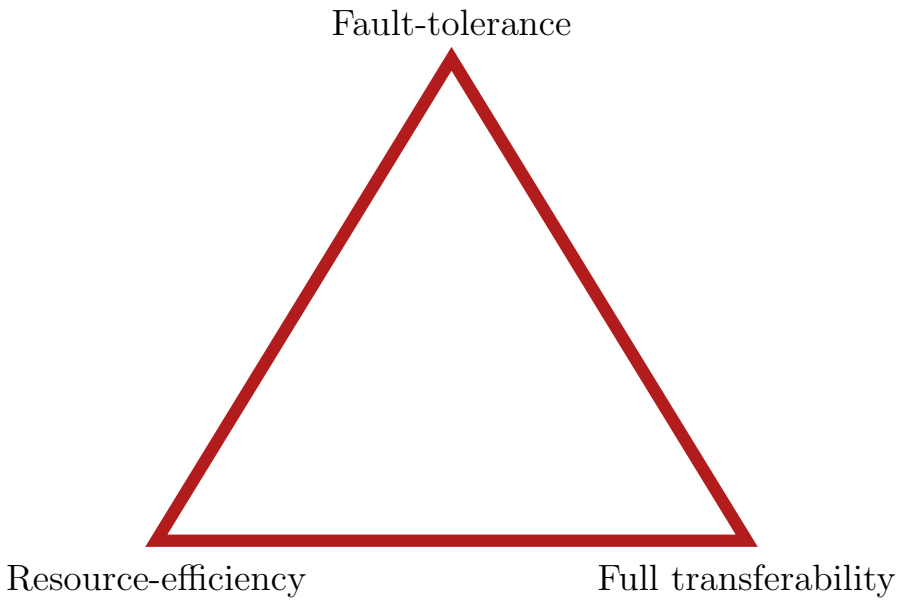

Figure 1: The Blockchain Trilemma.

We begin by adapting the theory of distributed consensus from the computer science literature (e.g., Lamport, Shostak, and Pease, 1980) to an economic setting. We impose typical assumptions on agents' communication technology and the behavior of faulty devices. Unlike previous work in computer science, though, we do not assume the existence of any agent who will automatically act honestly: the provision of incentives is key to our theory. Our consensus framework applies to a broad variety of digital record-keeping arrangements: it can accommodate centralized ledgers as well as PoW and PoS blockchains.

Our main contribution is a Blockchain Trilemma (Figure 1): we prove that no consensus algorithm can simultaneously achieve fault-tolerance, resource-efficiency, and full transferability. We also prove a converse: if there is a communication protocol with a trusted mediator that achieves all three desired properties, then any two of the desired properties are achievable by an unmediated consensus algorithm. Hence, there is a sense in which our framework allows us to tightly characterize the role of an inherently trustworthy mediator, which is one of the main questions that originally motivated the development of public blockchains (Nakamoto, 2008). When no agent can be trusted to behave honestly, it is necessary to either give up fault-tolerance (which is usually infeasible in digital systems) or sacrifice one of two types of efficiency: allocative efficiency (which is achieved under full transferability) or resource-efficiency. As we will show, giving up fault-tolerance renders dishonest behavior detectable, whereas incentives for honesty can be provided by giving up resource-efficiency or full transferability.

After proving the Trilemma, we use the insights of our model to discuss the relationship between the features of an ideal consensus algorithm and the types of digital record-keeping systems used in

practice. We elaborate on the ways in which common consensus algorithms (centralized, proof-of-work, or proof-of-stake) can be used to overcome communication frictions and identify the types of deviations that may allow record-keepers to fool other users of the ledger. The Blockchain Trilemma allows us to classify the types of incentives that ensure honesty in each case, allowing us to map each of the three main classes of record-keeping systems to a point on the Trilemma. All three of the types of ledgers we consider are fault-tolerant to a certain extent. We find that centralized ledgers and proof-of-stake blockchains both give up full transferability in order to provide incentives for a set of record-keepers. Proof-of-work blockchains, on the other hand, give up resource-efficiency.

Motivating example. We begin with a simple example to illustrate the intuition underlying the Trilemma. Consider an environment with three agents: Alice, Bob, and Carol. In the present, the three agents can produce goods for one another, and through their future interactions, they will generate a surplus V. An exchange of a unit of goods costs the producer c but yields utility $u > c$ for the consumer. The agents wish to record their transactions by using a ledger that permits them to keep track of the amount of future surplus promised to each of them. They begin with balances $\mathbf{v} = (v_A, v_B, v_C)$ on the ledger (with $v_A + v_B + v_C = V$) and can agree to a budget-feasible transfer of balances $\mathbf{t} = (t_A, t_B, t_C)$ (with $t_A + t_B + t_C = 0$) subject to the constraint that agents cannot spend more than their entire balance. For example, they may trade digital "tokens," say, agreeing to sell goods to each other at a price of p, so that one token is worth $\frac{u}{p}$ of future promised value.

Whereas monetary exchange usually takes the form of an exchange of physical tokens (e.g., cash), in a digital setting, transactions involve the exchange of messages. It is easy to determine whether a transaction is *valid* (e.g., whether a buyer owns the tokens they are trying to send) – this can typically be accomplished through cryptographic methods. However, it is far more difficult to determine whether a transaction is *final*, meaning that at all future times, users of the ledger will consider it to have occurred. If Alice can buy goods from Bob by sending a message of the form $m =$ "Sending token to Bob," what prevents her from spending that token again in the future by sending an identical message to Carol of the form $m' =$ "Sending token to Carol"? This is the key issue in digital record-keeping, known as the *double-spend problem*.

Naïvely, one might think that all agents could just agree only to finalize whichever transaction Alice sent first. In a digital setting, though, this is unrealistic. There are two key frictions in communication that impede this simple solution. First, there may be a delay in messages sent between agents, causing them to receive messages in different orders. Second, there may be *faults* in communication: agents' communication devices (i.e., computers) may be disconnected from the network, behave erratically, or shut down entirely.

Therefore, agents require a robust method of achieving *consensus* on the transactions that have been finalized in the ledger, in order to ensure that one of Alice's attempts to spend her tokens is universally recognized as final and the other is discarded. That is, agents must have some way of voting on which entry should be finalized in the ledger and reaching agreement on which votes have been cast. This voting method could be centralized, for example, in which case a single agent would have full authority to finalize entries in the ledger, or decentralized, in which case perhaps a

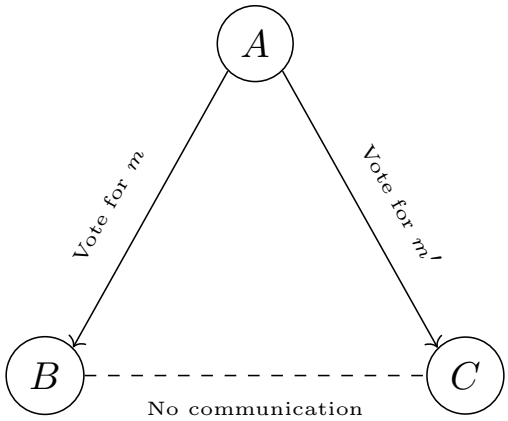

Figure 2: Alice's deviation in the motivating example. She sends conflicting votes to Bob and Carol, who are unable to communicate due to delays in message delivery.

majority would be required. No matter how the voting system is designed, though, the problem is that agents do not have access to a mediator who could be trusted to aggregate and honestly report their votes, ensuring that everyone sees the same ledger. Instead, the ledger is *distributed*: agents individually keep track of the votes that have been cast, and dishonest voting behavior could cause agents to disagree about the ledger's contents.

Thus, agents need a *consensus algorithm* that incentivizes them to behave honestly. Concretely, in this example, the ideal properties of a consensus algorithm are:

- **Fault-tolerance**: The consensus algorithm permits any two agents to come to agreement even when the third has a faulty communication device.
- **Resource-efficiency**: The consensus algorithm does not require agents to expend (significant) computational resources to update the ledger (e.g., electricity costs in Bitcoin mining).
- **Full transferability**: Any individually rational transfer of value among two agents can be implemented via the consensus algorithm.

The main difficulty in designing a consensus algorithm is that no matter how it is designed, Alice may communicate with Bob as if she wants to vote on message m while communicating with Carol as if she wants to vote on message m' (Figure 2). As long as consensus algorithm is fault-tolerant, this strategy may be profitable for Alice. Fault-tolerance requires that Alice be able to reach agreement with Bob in Carol's absence, and that Alice and Carol be able to reach agreement without Bob. If messages are delivered with some delay, Alice may agree on message m with Bob, while simultaneously reaching an agreement on m' with Carol, *before* Bob and Carol ever communicate with each other (and discover Alice's dishonesty). This why the double-spend problem arises.[1]

[1]Note that Alice could also try to convince Carol that she sent tokens to a different account A' that she also owns, as often observed in reality.

A *fault-tolerant* consensus algorithm must therefore provide Alice with incentives not to double-spend. How can a well-designed consensus algorithm dissuade Alice from engaging in this type of dishonest behavior? It may impose ex-ante costs on agents who cast multiple conflicting votes, or it may punish dishonest agents in some way ex-post. The proof-of-work algorithm used by Bitcoin and many other public blockchains imposes ex-ante costs by forcing agents to solve computational problems in order to create a "block," which is effectively a vote cast in favor of a set of transactions. Hence, by giving up *resource-efficiency*, it is possible to incentivize Alice to behave honestly.

Alice can also be punished ex-post if the consensus algorithm permits agents to verify her dishonest behavior after she has spent her tokens twice. Even if Bob and Carol do not communicate with each other during the initial voting process, eventually they will be able to communicate and realize that Alice lied (for example, if each of them has a signed message from Alice voting on different messages). Then, Bob and Carol can use that evidence to update the state of the ledger, reducing the value promised to Alice (to zero if necessary). However, if Alice is permitted to transfer her entire promised value to Bob (or Carol) in a transaction, it is not possible to punish her – there will be nothing more to take from her if she behaves dishonestly. In other words, if she is permitted to transfer her entire promised value to others, she can no longer be trusted to be honest. Hence, ex-post punishments are feasible only if *full transferability* is relinquished.

This simple argument illustrates the impossibility result in the Blockchain Trilemma: a consensus algorithm cannot simultaneously achieve fault-tolerance, resource-efficiency, and full transferability. In our model, we also prove a converse: any two of the desired properties can be achieved if the third is given up.

Related Literature. Our paper is mainly related to three strands of the academic literature. First, our paper follows a long tradition in economics that studies communication in games with and without trusted mediators. Forges (1986) and Myerson (1986) illustrate how a revelation principle applies to communication mechanisms in games with trusted mediators. The literature on implementation theory (e.g., Maskin, 1999, among many others) studies how games with trusted mediators can be designed to implement desirable social outcomes. Eliaz (2002) extends this work to a setting in which some agents may be "faulty" and communicate in unpredictable ways, like ours. Our paper's main point of departure from this extensive literature is the assumption that agents lack a trusted mediator.

Of course, there are other papers in economics that study unmediated communication in games. Forges (2020) provides an extensive treatment of the subject.[2] Relative to the literature on unmediated communication, we differ in our assumptions regarding communication frictions – namely, that agents may have faulty communication devices and that communication takes time. As our results show, these realistic assumptions are consequential for our conclusions.

Second, our paper is related to the literature on distributed consensus in computer science. Lamport, Shostak, and Pease (1980) is the seminal paper in the study of synchronous consensus

[2]See also Aumann and Hart (2003), Ben-Porath (1998, 2003), and Gerardi (2004), which study the correlated equilibria that may arise.

algorithms that tolerate "Byzantine" faults. We focus on a setting with asynchronous communication, studied by Bracha and Toueg (1985), Fischer, Lynch, and Paterson (1985), and Castro and Liskov (1999), among others. In the existence result of our Trilemma, we build on the consensus algorithms derived by this earlier literature.[3] Our main contribution in this area is to adapt the distributed consensus literature to an economic setting. We require that consensus algorithms be designed not only to protect against non-strategic faults in communication, but also to disincentivize coordinated deviations by strategic agents. Concurrently with our work, Halaburda, He, and Li (2021) provide an economic analysis of equilibrium multiplicity under the Byzantine Fault Tolerance (BFT) algorithm developed by Castro and Liskov (1999).

Finally, our work is related to the emergent literature on the economic properties of blockchains and cryptocurrencies. Budish (2018) and Gans and Gandal (2019) study the costs of incentivizing honesty for cryptocurrency blockchains in isolation. Biais et al. (2019) propose a game-theoretic model of Bitcoin mining and show that while the strategy of mining the longest chain proposed by Nakamoto (2008) is in fact an equilibrium, there are other equilibria in which the blockchain forks, as observed empirically. Saleh (2020) shows how proof-of-stake blockchains may guarantee security through fluctuations in token prices. Relative to the previous literature, we study a unified record-keeping framework and outline the requirements that must be satisfied by *any* record-keeping system in order to ensure security.

Outline. The remainder of the paper is structured as follows. We present our model of digital record-keeping in Section 2. In the model, we describe a class of communication games in which agents attempt to reach consensus on updates to a ledger. We introduce our concept of a consensus algorithm and define the fault-tolerance, resource-efficiency, and full transferability properties. Then, we outline our model's main assumptions, allowing us to prove the Trilemma in Section 3. We proceed to discuss how our model relates to digital record-keeping in practice as well as how the examples we consider (centralized, proof-of-work, and proof-of-stake) map to our three ideal properties in Section 4. Sections 5 and 6 discuss our main assumptions on the communication technology and formally extend the model to incorporate relaxations of those assumptions. Section 7 concludes.

2 The Consensus Framework

We model an environment with a digital ledger in which agents must come to a consensus on a collection of transactions. We first describe the payoff environment and then introduce the consensus problem that arises in digital record-keeping. Following the description of the communication game that agents play to reach consensus on an outcome, we define our notion of consensus algorithms (postponing some technical details to Appendix A). We conclude this section by summarizing, at a high level, the key assumptions of our model that lead to the Blockchain Trilemma.

[3]The impossibility result of Fischer, Lynch, and Paterson (1985) does not apply to our setting because we assume the lags in message delivery are random and cannot be manipulated by malicious attackers.

2.1 Payoff environment

There is a set $\mathcal{N} = \{1, \ldots, N\}$ of agents (with $N \geq 3$) whose preferences are summarized by a publicly known state of the world θ in a finite set Θ. Agents can realize a finite set of physical *transactions* $\mathcal{Y}$. A transaction $y \in \mathcal{Y}$ is a voluntary exchange of goods among some subset of agents $S(y) \subset \mathcal{N}$, who are said to be *involved* in transaction y. When a transaction y occurs, agent n receives utility $u_n(y|\theta)$, where $u_n : \mathcal{Y} \times \Theta \to \mathbb{R}$ describes n's preferences over individual transactions.

Agents can compensate each other in transactions by transferring balances they hold on a ledger. Each agent n initially holds a non-negative integer balance $v_n \in \mathbb{N}$ on the ledger.[4] To each transaction is associated a budget-feasible *transfer* of ledger balances $\mathbf{t}(y) = \{t_n(y)\}_{n \in S(y)} \in \mathbb{Z}^{S(y)}$ among the agents $S(y)$ involved in the transaction, such that $\sum_{n \in S(y)} t_n(y) = 0$ and $t_n(y) \geq -v_n$ for all $n \in S(y)$, that is, agents cannot transfer more than their entire balance.

An *outcome* $x = (\mathbf{y}(x), \mathbf{t}(x))$ in this environment is a budget-feasible collection of transactions $\mathbf{y}(x) \subset \mathcal{Y}$ and the corresponding transfer of balances $\mathbf{t}(x) = \sum_{y \in \mathbf{y}(x)} \mathbf{t}(y)$. Agents' preferences are assumed to be additively separable across transactions and quasilinear in ledger balances,[5] so an outcome x is *individually rational* in state θ if

$$\sum_{y \in \mathbf{y}(x)} u_n(y|\theta) + t_n(x) \geq 0 \ \forall \ n \in \mathcal{N}. \tag{IR}$$

2.2 Digital record-keeping

The purpose of record-keeping is to permit agents to transfer value in order to implement an individually rational (i.e., mutually beneficial) outcome x. Our environment is consistent with both physical record-keeping (e.g., via the exchange of cash) and digital record-keeping. In either case, the "ledger" is a representation of the payoffs agents will receive through their future interactions, and physical transactions are realized when the agents involved agree that the ledger has been "updated" appropriately.

When records are kept with physical tokens, an update to the ledger is just an exchange of tokens. For a transaction y to take place, the participants $S(y)$ in that transaction need only determine whether each agent has the requisite number of tokens. An agent cannot spend the same token in multiple transactions, so transactions can be agreed upon independently.

The key distinguishing feature of digital record-keeping is that the ledger is a shared database of digital transfers of value. Unlike physical record-keeping, there is no guarantee that an agent will not spend the same unit of value in multiple transactions. Agents must communicate to reach a *consensus* on an update to the ledger in order to ensure that no agent violates his budget constraint. In a digital setting, then, agents must reach agreement with others not only on their own desired transactions, but on the entire set of transactions to be added to the ledger. An effective method

[4] If we deal with a finite set of possible transactions, it is without loss of generality to assume that agents' balances have integer values. In reality, of course, balances on digital ledgers must come in discrete increments.

[5] We make this assumption for simplicity of notation, but it is inessential for our results.

of reaching consensus will ensure that no agent agrees to a transaction that has not been finalized in the eyes of all other agents.

We will present an abstract static model of consensus that adds game-theoretic elements to the typical framework used by much of the computer science literature studying blockchain consensus protocols.[6] In section 4.2, we apply our model to discuss how digital record-keeping systems in reality (centralized, proof-of-work, and proof-of-stake) deal with the problem of transaction finality.

The communication game. We model the process of reaching consensus on an outcome x as a communication game $\mathcal{G}$ played among agents. There is no mediator who can be inherently trusted to collect information about agents' desired transactions and report an outcome to them. Instead, agents communicate in rounds $k = 1, 2, \ldots$ using *nodes*, which are programmable devices that can send one another bilateral, private messages and eventually *decide* on an outcome. When agent n's node decides on outcome x, n acknowledges the update to the ledger $\mathbf{t}(x)$ and agrees to complete any transactions $y \in \mathbf{y}(x)$ in which she is involved.[7] Messages may be costly to generate, as in proof-of-work. Figure 3 illustrates the timeline of the game.

We introduce two frictions in communication that are central to the study of distributed consensus. These frictions will in general make it difficult to detect dishonest behavior.

Assumption 1. *There are two frictions in communication.*

1. *Messages are delivered with a* ***random lag*** *of at most* Δ *rounds.*[8]

2. *Some agents have* ***faulty*** *nodes that are incapable of communicating.*

Faulty nodes should be interpreted as computers that are shut down or offline for other reasons. With some abuse of terminology, we sometimes directly refer to agents with faulty nodes as faulty agents. Agents do not necessarily know who is faulty and who is not. We will use a robust equilibrium concept that will not require us to further specify agents' beliefs. We extend Assumption 1 to accommodate *arbitrary* behavior by faulty nodes (e.g., glitches or hacks) in Section 6.

Payoffs. Agents' payoffs depend on their nodes' decisions as follows:

- A physical transaction y is agreed upon (and payoffs $u_n(y|\theta)$ are realized) if at any point the nodes of all agents involved, $n \in S(y)$, have decided on an outcome x such that $y \in \mathbf{y}(x)$.

- *Consensus* is achieved once all non-faulty nodes have decided on a common outcome x^*. Agents receive the payoffs $v_n + t_n(x^*)$ and the game ends.

Note that transactions are realized before the final transfer of ledger balances $\mathbf{t}(x^*)$ takes place. This formulation allows agents to violate their budget constraints by acting dishonestly: they can

[6] Narayanan et al. (2016) articulate the link between blockchain consensus algorithms used in practice and the theory of distributed consensus developed by to Lamport, Shostak, and Pease (1982). Garay and Kiayias (2020) provide a comprehensive framework for the study of consensus algorithms that is compatible with our model.

[7] After deciding on an outcome, a node can revise its decision (e.g., if it initially decides on the wrong value due to dishonest behavior by others).

[8] Formally, the lag length is drawn independently from a distribution G with full support on $\{0, 1, \ldots, \Delta\}$. The IID assumption allows us to circumvent the impossibility result of Fischer, Lynch, and Paterson (1985).

convince others to agree to transactions y that are not finalized in the ledger as part of the eventual consensus outcome x^*.

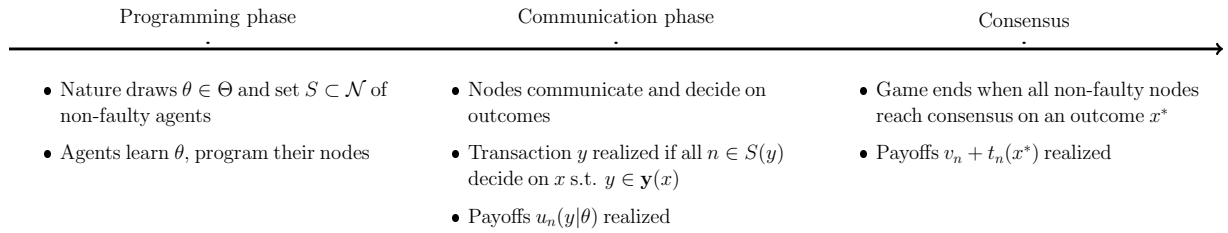

Figure 3: Timeline of the communication game played by agents using their nodes.

Agents program their nodes to send messages and decide on an outcome based on the history of messages they have previously received, in order to maximize

$$U_n = \mathbb{E}\left[\sum_{y \in \mathbf{y}} u_n(y|\theta) + v_n + t_n(x^*) - \sum_{m \in \hat{M}_n} \kappa(m)\right], \tag{1}$$

where $\mathbf{y}$ is the set of realized transactions, $\kappa(m) \geq 0$ is the disutility cost of sending a message m, and $\hat{M}_n$ is the set of messages sent by n over the course of the game. We provide a formal description of this class of communication games in the Appendix (Definition A.1).

2.3 The design objective

In this section, we introduce the key concepts in our theory of consensus algorithms at a high level. In order to streamline the exposition and avoid burdensome notation, our discussion is not entirely formal. Precise definitions for the concepts in this section can be found in Appendix A.2.

There is a *designer* who would like to devise a method by which the set of agents with non-faulty nodes can reach consensus on an appropriate outcome, called a *consensus algorithm*.

Definition 1. *A **consensus algorithm** consists of a communication game $\mathcal{G}$ played by agents and a communication protocol $\mathcal{C}$ that their nodes should be programmed to follow.*

The designer specifies the rules of the game, including the set of messages a node is permitted to send in each round and the cost of sending each message. The communication protocol is a profile of strategies dictating how nodes should communicate with each other and when they should decide on an outcome (i.e., conclude that a set of transactions has been finalized in the ledger). A node n's prescribed strategy should depend only on its information, i.e., the history of messages it has previously received. Section 4 will discuss the different types of consensus algorithms used by centralized systems as well as proof-of-work and proof-of-stake blockchains.

If a consensus algorithm enables a subset $S \subset \mathcal{N}$ of agents to reach consensus on their own, even when agents $\mathcal{N} - S$ are faulty, it must satisfy two conditions. These ensure that agents in S have incentives to follow the protocol and that, when they do, consensus is achieved on an individually rational outcome.

Definition 2. *A subset of agents $S \subset \mathcal{N}$* ***can achieve consensus*** *under the consensus algorithm $(\mathcal{G}, \mathcal{C})$ if, whenever agents in S have non-faulty nodes, the following two conditions hold:*[9]

- *(**Condition 1**) Agents in S are guaranteed to reach consensus on an individually rational outcome if they follow the communication protocol $\mathcal{C}$.*
- *(**Condition 2**) No coalition $S' \subset S$ has an incentive to jointly deviate from protocol $\mathcal{C}$.*[10]

It might be the case that some (but not all) $S \subset \mathcal{N}$ can achieve consensus. As an example, some blockchain consensus protocols guarantee consensus only when a majority of nodes are non-faulty, so $S \subset \mathcal{N}$ achieves consensus if and only if $|S| > \frac{1}{2}N$. By contrast, users of centralized record-keeping systems can achieve consensus only when the node belonging to the ledger's owner is non-faulty, so if agent n^* owns the ledger, S can achieve consensus whenever $n^* \in S$.

Condition 1 is commonplace in the study of consensus algorithms in computer science. However, in an economic environment, the designer faces an additional challenge. Agents cannot be forced to program their nodes in accordance with the communication protocol: they must have incentives to do so (Condition 2). There is a natural motive to lie: by doing so, agents can fool others into accepting payments that are later nullified (as in our motivating example). This challenge is unique to a setting without a trusted mediator: if there were a mediator who could be trusted to report an outcome, there would be no possibility of disagreement on updates to the ledger.

Importantly, we require that the consensus algorithm be robust to *coalitional* deviations. A common concern, especially in the context of decentralized record-keeping systems, is that one entity might be in control of several nodes simultaneously. Blockchains are therefore typically designed with coalitional deviations in mind.[11] As we argue in Section 4.2, absent such concerns, it would be trivially easy to design a fully secure consensus algorithm, so the coalition-proofness requirement is essential for our results.

Having formulated our concept of a consensus algorithm, we can describe the three ideal properties in the Trilemma.

Definition 3. *We define the following three properties of a consensus algorithm:*

- *(**Fault-tolerance**) Any majority $S \subset \mathcal{N}$ can achieve consensus.*
- *(**Resource-efficiency**) The consensus algorithm does not make use of costly messages.*
- *(**Full transferability**) If $S \subset \mathcal{N}$ can achieve consensus, then agents in S can reach agreement on any individually rational outcome x such that $\mathbf{t}(x)$ transfers value among them.*[12]

[9] Specifically, we require that Conditions 1 and 2 hold when agents in S *know* that agents in $\mathcal{N} - S$ are faulty.

[10] Formally, we impose the Strong Nash Equilibrium condition of Aumann (1959) (Definition A.3 in Appendix A). We use this coalition-proofness requirement rather than that of Bernheim, Peleg, and Whinston (1987) because we envision a situation in which deviating nodes are controlled by one entity (see below). We further discuss our equilibrium notion in Online Appendix D.

[11] In Bitcoin, for instance, the most relevant concern is a "51% attack" in which an entity controls the majority of the network's computing power. Section 4.2 provides other examples.

[12] More specifically, let S be a set of agents who can achieve consensus and $x = (\mathbf{y}(x), \mathbf{t}(x))$ be an outcome that is individually rational in some state θ such that $\mathbf{t}(x)$ is a transfer among agents in S. Then when the state is θ, if agents in S are non-faulty and follow the protocol, they reach consensus on x with positive probability.

Fault-tolerance is desirable because it is typically not practical to guarantee that all users of a network will always be online, so the possibility that some agents go offline or face computer crashes should not prevent the ledger from being updated. A fault-tolerant consensus algorithm guarantees that agents will have incentives to follow the protocol and will actually reach consensus, as long as faulty nodes are known to be in the minority.[13] The designer would like the consensus algorithm to be resource-efficient because communication is not intrinsically costly, so it is undesirable for the designer to impose communication costs that generate a deadweight loss of utility (unlike Bitcoin's proof-of-work). The beneficial implications of full transferability are clear: the consensus algorithm should not arbitrarily prevent a subset of agents S from reaching agreement on any mutually beneficial (i.e., individually rational) transfer of value among themselves.

Remark. Fault-tolerance imposes a notion of robustness not only to the *presence* of faulty nodes, but also to agents' *beliefs* about the faultiness of others. As long as an agent believes that *some* majority of nodes are non-faulty, Condition 2 implies that the agent has incentives to follow the protocol. It does not matter how that agent assigns probabilities to the faultiness of *individual* nodes. Hence, our equilibrium concept permits agents to entertain heterogeneous beliefs about the faultiness of others (as in Eliaz, 2002). We discuss this point further in Online Appendix D.

2.4 The model assumptions

We will study the conditions under which it is possible to design a consensus algorithm that is fault-tolerant, resource-efficient, and achieves full transferability. We make three additional assumptions that lead to our main result, the Blockchain Trilemma, which states that in the absence of a trusted mediator, it is only possible to simultaneously achieve two of those objectives. Lemma 1 guarantees that, at least, it is possible for a communication protocol with a trusted mediator to achieve fault-tolerance, resource-efficiency, and full transferability.

Lemma 1. *Under Assumption 1, when a trusted mediator is available,*[14] *there exists a consensus algorithm that achieves fault-tolerance, resource-efficiency, and full transferability.*

We provide a proof in Appendix B.

The consensus algorithm with a trusted mediator is extremely simple. First, the mediator asks agents to report the state of the world θ. Then, if the mediator receives the same report θ from a subset S of agents, the mediator reports back some outcome x^* consisting of transactions among agents in S that are individually rational in state θ.[15] Agents have prior knowledge of the state of the world θ but not of an outcome x^* because they do not know the set of non-faulty agents S, who are the only ones capable of participating in transactions. A trustworthy mediator permits non-faulty agents to attain common knowledge of a suitable outcome. A lack of such common knowledge

[13] This is the *maximal* achievable degree of fault-tolerance, but, of course, it is possible to have partial fault-tolerance as well. For example, it could be that a two-thirds supermajority is required for consensus.

[14] A trusted mediator is a special node n^* (owned by no agent) that is assumed to automatically follow the communication protocol chosen by the designer.

[15] If the mediator receives conflicting reports, she does not report an outcome to agents. This is similar to the implementation procedure derived by Maskin (1978) in a setting with the possibility of coalitional deviations.

in an unmediated setting is what allows for the possibility of profitable dishonest behavior, causing different agents to decide on different outcomes.

Our key assumption on transactions and preferences implies an incentive for dishonesty that conflicts with the goal of full transferability. It guarantees that agents would like a record-keeping system that permits them to implement *any* budget-feasible transfer of value.

Assumption 2. *The set of outcomes and agents' preferences have the following two properties:*

- *For each budget-feasible transfer $\tilde{\mathbf{t}}$ among agents in S,*[16] *there exists an outcome $x = (\boldsymbol{y}(x), \boldsymbol{t}(x))$ such that $\boldsymbol{t}(x) = \tilde{\mathbf{t}}$ and each transaction $y \in \boldsymbol{y}(x)$ involves only agents in S, $S(y) \subset S$.*
- *For each set of transactions $\boldsymbol{y} \subset \mathcal{Y}$, there exists a state $\theta \in \Theta$ in which the set of individually rational transactions is $\boldsymbol{y}$.*

That is, every budget-feasible transfer $\mathbf{t}$ is part of some outcome x, and each outcome x is individually rational in some state θ. Under Assumption 2, the incentives for dishonesty parallel those in our motivating example. In particular, once an agent n's balance goes to zero in an outcome x (i.e., $t_n(x) = -v_n$), that agent would like to fool others into agreeing to another outcome x' that will benefit her. Assumption 2 guarantees that there in fact exists an outcome with $t_n(x) = -v_n$.

Whether it is possible to design an ideal consensus algorithm depends critically on the designer's capacity to limit dishonest behavior, so we must make assumptions about what the designer can do. Assumption 3 dictates the types of proofs that the designer can require agents to include in their messages, in order to disincentivize or limit the scope for this type of dishonest behavior.

Assumption 3. *The designer can require a node n to include the following types of proofs in a message m.*

1. *(**Proof-of-identity**) A verifiable, unforgeable signature, proving that the owner of node n sent the message.*
2. *(**Proof-of-receipt**) A proof that another message m' was previously received.*
3. *(**Proof-of-work**) A proof that agent n incurred a computational cost $\kappa(m)$ in the production of message m.*

Assumption 3 (formalized as Assumption A.1) concerns the types of deviations that are possible in the communication game $\mathcal{G}$ chosen by the designer. Proof-of-identity and proof-of-receipt prevent nodes from lying about certain statements, restricting the set of messages each node is permitted to send in each round. One agent's node cannot pretend to send messages on behalf of another agent, and a node cannot pretend to have received messages that were never sent to it. Proof-of-work, by contrast, disincentivizes dishonesty: it makes it costly for a node to send conflicting messages to different groups. Importantly, though, the designer cannot prevent nodes from pretending that they *did not* receive certain messages, which is crucial for our impossibility result.

[16] Recall that a budget-feasible transfer is $\tilde{\mathbf{t}} \in \mathbb{Z}^{\mathcal{N}}$ s.t. $\sum_{n \in S} \tilde{t}_n = 0,\ \tilde{t}_n \geq -v_n\ \forall\ n \in S,\ \tilde{t}_n = 0\ \forall\ n \notin S$.

Assumption 3 will permit the designer to implement a wide variety of record-keeping arrangements (e.g., centralized, proof-of-work, and proof-of-stake), as we discuss in Section 4. Additionally, it admits both ex-ante costs of dishonesty (through proof-of-work) as well as ex-post punishments: proof-of-receipt allows nodes to prove that others have previously sent dishonest messages, allowing honest agents to eventually reach consensus on a state in which dishonest agents are stripped of their balances on the ledger.

The designer faces a key limitation, however. As specified in Assumption 4, he does not know Δ, the maximum possible delay in the delivery of messages.

Assumption 4. *The maximum delay Δ in the delivery of messages is unknown to the designer.*

Assumption 4, formalized as Assumption A.2, is a restriction on the types of the communication protocols $\mathcal{C}$ the designer can choose: nodes' prescribed behaviors cannot depend on Δ. This assumption, known as *asynchronous* communication, is common in the computer science literature. It is well-known in other contexts that asynchronous communication introduces significant difficulties in the design of consensus algorithms.[17] Asynchronicity is the most appropriate assumption in the context of large-scale networks with ex-ante unknown users, e.g., the internet.

Section 4 relates our model to digital record-keeping in practice. We discuss our model's assumptions further in Sections 5 and 6. We now proceed to our main result: the Blockchain Trilemma.

3 The Blockchain Trilemma

The Blockchain Trilemma follows from Assumptions 1-4. In the absence of a trusted mediator, no consensus algorithm can simultaneously achieve fault-tolerance, resource-efficiency, and full transferability. Only two of these goals are simultaneously achievable. Hence, in light of Lemma 1, the Trilemma characterizes the costs of a lack of trust.

Blockchain Trilemma. *Under Assumptions 1-4, the following hold:*

- ***(Impossibility)*** *There does not exist a consensus algorithm that is fault-tolerant, resource-efficient, and achieves full transferability (for all Δ).*
- ***(Existence)*** *For any two of the desired properties in Definition 3, there exists a consensus algorithm satisfying both properties (for all Δ).*

We begin with a discussion of the ideas behind the Trilemma, providing an overview of the tension between the designer's three objectives. Specifically, we show how the designer faces a tradeoff between fault-tolerance, which provides opportunities for dishonest behavior, and incentives for honesty, which can be provided by relinquishing resource-efficiency or full transferability. Then we outline a proof sketch of the impossibility result, but we postpone some technical details to the full

[17] For example, compare the result of Lamport, Shostak, and Pease (1982), who show that consensus with signed messages is always possible under synchronous communication, to the impossibility result of Bracha and Toueg (1985), who prove consensus cannot be achieved in an asynchronous context when the majority of nodes are faulty.

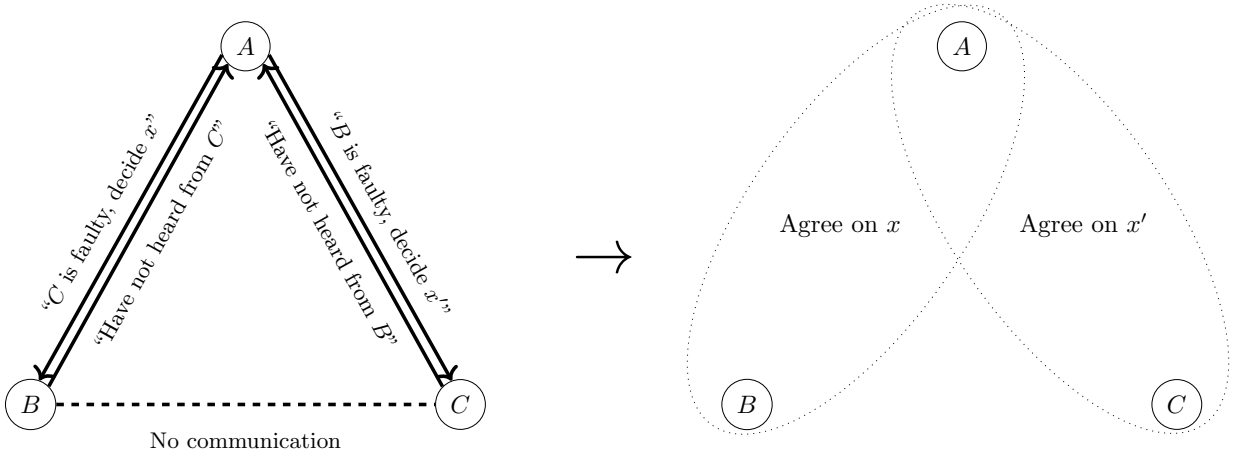

Figure 4: The deviation pursued by coalition A in the double-spend lemma. Agents in A communicate with agents in B as if they have never received a message from C, and they communicate in an analogous way with agents in C.

proof in Appendix C. The constructions of the consensus algorithms required for the proof of the existence result are involved (but well-known in the computer science literature). In the Online Appendix, we describe these consensus algorithms and prove the existence result.

The first step in our argument is a key intermediate result, the "double-spend lemma," showing how agents can profitably deviate from honesty under asynchronous communication. The lemma takes as given a consensus algorithm $(\mathcal{G}, \mathcal{C})$ that achieves consensus for two overlapping sets of agents, S and S', and a pair of individually rational outcomes x, x' such that x (resp. x') can be implemented by the consensus algorithm in state θ when agents in S (resp. S') are non-faulty.[18] It demonstrates that the coalition $A = S \cap S'$ can deviate from the protocol $\mathcal{C}$ so that after transactions $\mathbf{y}(x)$ and transfers $\mathbf{t}(x)$ are agreed upon by members of $B = S \backslash S'$, then a different set of transactions $\mathbf{y}(x')$ and transfers $\mathbf{t}(x')$ are agreed upon by members of $C = S' \backslash S$ as well (with positive probability p). Such a deviation permits members of A to violate their budget constraints by spending their balances on the ledger twice, once in transactions $\mathbf{y}(x)$ and again in transactions $\mathbf{y}(x')$.

The idea underlying the deviation is exactly as in our motivating example: after coalition A has reached agreement on outcome x with coalition B (the analogue of Bob), then nodes in A communicate with C (the analogue of Carol) as if they have never heard from B, convincing C that consensus should be reached on outcome x' instead. By Assumption 3, this deviation from honesty is feasible. Assumption 4 implies that this deviation cannot always be detected, e.g., when Alice told Bob she had never heard from Carol, Bob did not know if Carol's node was faulty or if Carol was simply taking a long time to communicate. Figure 4 illustrates this deviation. The deviation we consider is slightly different from how some double-spends actually occur in practice,

[18] Formally, we mean that with positive probability, when agents in S (resp. S') are non-faulty and follow the protocol, then in state θ, consensus is achieved on x (resp. x') with positive probability.

in which the second recipient ("Carol") is usually another account owned by the deviating agent. We discuss in Section 4.2 how the deviation we derive can be viewed as representative of any kind of double-spend in reality.

The double-spend deviation highlights the tension between fault-tolerance, resource-efficiency, and full transferability, captured by a simple incentive condition (Inequality 2). When the double-spend deviation succeeds (with probability p), agents in A receive utility $\tilde{u}_n(x'|\theta) \equiv \sum_{y \in \mathbf{y}(x')} u_n(y|\theta)$ from the transactions that are realized. If this deviation is eventually discovered, the maximum punishment that can be inflicted on these agents is to strip them of their remaining ledger balances $v_n - t_n(x)$.[19] Furthermore, each agent n in the deviating coalition may have to incur an ex-ante proof-of-work cost $\kappa_n(x')$ to send the messages required to engage in the deviation. Therefore, if honest behavior is optimal, it must be that

$$\underbrace{\tilde{u}_n(x'|\theta) \cdot p}_{\text{Deviation benefit}} \;<\; \underbrace{v_n - t_n(x) + \kappa_n(x')}_{\text{Deviation cost}} \text{ for some } n \in S \cap S'. \tag{2}$$

With a fault-tolerant consensus algorithm, S and S' in Inequality 2 can be taken to be any two distinct majorities of agents, since any majority must be able to achieve consensus on its own. If the consensus algorithm achieves full transferability, then there exists a state θ and an individually rational outcome x such that agents in A spend their entire balance, $v_n - t_n(x) = 0$ for all $n \in A$. Resource-efficiency implies that the cost of engaging in any deviation is zero, $\kappa_n(x) = 0$. Hence, agents in A face no cost for deviating and attempting to convince C to decide on the outcome x', allowing them to spend their balances again. This argument leads to the impossibility result.

The existence result can also be understood through Inequality 2. We have argued that under asynchronous communication, a fault-tolerant consensus algorithm permits some deviating coalition A to spend their balances twice (in one set of transactions with B and another with C). Critically, for the deviation to go undetected, $A \cup B$ must be able to reach consensus without input from C and, likewise, $A \cup C$ must be able to reach consensus without B. Therefore, the designer can render double-spend deviations detectable by giving up fault-tolerance. If input from all agents is required to achieve consensus, there is no possibility that different agents will decide on different outcomes, and Inequality 2 becomes irrelevant. More generally, the designer can partially relax the constraints in Inequality 2 by increasing the fraction of agents required for consensus (e.g., from a majority to two-thirds). This reduces the probability p that the double-spend deviation will succeed and increases the size of the coalition $S \cap S'$ that must deviate in order to double-spend.[20]

On the other hand, if the designer wishes to maintain fault-tolerance, he has two options: he can design a communication protocol that dissuades double-spending by imposing ex-post punishments on those who deviate from honesty, or he can impose resource costs that make deviation expensive

[19] After agents discover that coalition A deviated, they can communicate evidence of this deviation to each other and reach consensus on an outcome in which those agents lose their balances.

[20] For example, when a simple majority is required for consensus, the deviating coalition $S \cap S'$ might consist of only one agent. When a fraction q is required for consensus, then the deviating coalition must consist of at least $\lceil 2q - 1 \rceil N$ agents.

from an ex-ante perspective. The term $v_n - t_n(x)$ captures the maximum punishment that can be imposed on an agent ex-post. By relinquishing full transferability and restricting transfers of value among agents, the designer can ensure that ex-post punishments are always sufficient to dissuade dishonesty. The designer can instead maintain full transferability and give up resource-efficiency. If the cost of updating the ledger is large enough (captured by $\kappa_n(x')$), then agents in the deviating coalition will not find it worthwhile to double-spend, even if ex-post punishments are not possible.

3.1 The double-spend lemma

The double-spend lemma illustrates the primary difficulty with the design of unmediated consensus algorithms in asynchronous settings, and it is the key intermediate step in our impossibility result. We provide a statement of the Lemma and a proof sketch below.

Lemma 2 (Double-spend lemma)**.** *Suppose that Assumptions 1, 3, and 4 hold. Fix a state $\theta \in \Theta$ and distinct majorities of agents $S, S' \subset \mathcal{N}$.*[21]*, and consider a fault-tolerant consensus algorithm $(\mathcal{G}, \mathcal{C})$. Then for large enough Δ, there exists a deviation from the prescribed communication protocol $\mathcal{C}$ for the coalition $A = S \cap S'$ such that, with positive probability,*

- *The group $B = S \backslash S'$ all decide on an outcome x, and transactions $\boldsymbol{y}(x)$ are realized;*
- *After transactions $\boldsymbol{y}(x)$ are realized, the group $C = S' \backslash S$ all decide on an outcome $x' \neq x$, and transactions $\boldsymbol{y}(x')$ are realized.*

Furthermore, such a deviation exists for any two outcomes x, x' such that x (resp x') is implemented by the consensus algorithm in state θ when agents in S (resp. S') are non-faulty.

Proof sketch. In lieu of a proof, here we provide a sketch of the main intuition. In Appendix C we present a formal proof including all of the technical details.

Since $A \cup B = S$ constitutes a majority of agents and the consensus algorithm is fault-tolerant, then by Condition 1, it must be possible for $S = A \cup B$ to reach a consensus if they communicate honestly, even if all nodes in C are faulty. Hence, there must be some finite number of messages that nodes in $A \cup B$ send to each other before reaching consensus on some outcome x, even if they have received no messages from C (since faulty nodes do not communicate, by Assumption 1). Let T_S be the minimum number of rounds in which consensus can be reached by $A \cup B$ with positive probability.[22] By the same logic, under honest communication, there must be some finite number of messages that nodes in $A \cup C$ will send each other before reaching agreement on x', even if they never communicate with B. Let $T_{S'}$ be the minimum number of rounds of communication in which this occurs with positive probability. The stopping times T_S and $T_{S'}$ are determined by the communication protocol chosen by the designer, who does not know the maximum message delay Δ (by Assumption 4).

Consider the following coalitional deviation for agents in A:

[21] That is, the cardinalities of these subsets, $|S|$ and $|S'|$, are both greater than $\frac{N}{2}$.

[22] Specifically, the minimum number of rounds T_S is the number of rounds it would take to reach consensus if every message sent by a member of $A \cup B$ was delivered in a single round.

- Communicate with nodes in B honestly according to the communication protocol $\mathcal{C}$, until agreement is reached on some outcome x.
- After agreement is reached on x, communicate with nodes in C according to the communication protocol $\mathcal{C}$, as if they have never received any messages from nodes in B.

This deviation is illustrated in Figure 4. Now, suppose that all messages from B to C, and C to B, are delayed by $D > T_S + T_{S'}$ rounds. Since T_S and $T_{S'}$ are independent of Δ, there exists Δ large enough that this occurs with positive probability. By Assumption 3, this deviation is undetectable by any agent in $B \cup C$ before round $T_S + T_{S'}$: it would have been impossible for A to provide proof to C that they have never communicated with B, so B and C cannot learn this fact until they communicate with each other. With positive probability, then, A will convince B to decide on outcome x in round T_S and then convince C to decide on x' in round $T_S + T_{S'}$. Transactions $\mathbf{y}(x)$ and $\mathbf{y}(x')$ are realized before A's deviation is revealed, which occurs in round D at the earliest.

We leave to Appendix C the proof that such a deviation is possible for *any* two outcomes x, x' that are implemented by the consensus algorithm in state θ.

□

These types of deviations are unique to unmediated settings: if there were a trusted mediator, she would report the same outcome to all agents, ensuring no two nodes that follow the communication protocol ever decide on different outcomes. Furthermore, the assumption of asynchronous communication is crucial as well: as we show in Section 6.2, in a setting with synchronous communication, these types of deviations are no longer possible (with the caveat that the synchronous communication is often an unrealistic ideal). Hence, the lemma demonstrates the obstacles that are specific to the implementation of unmediated, asynchronous consensus algorithms. Note, also, that the argument in the proof of Lemma 2 applies only to fault-tolerant consensus algorithms. This argument then highlights how tolerance to faults (non-strategic abstention from communication) generates scope for strategic deviations by a coalition of attackers that wish to manipulate outcomes, which is a key concept underlying our Trilemma.

3.2 Proof sketch of the impossibility result

Having proved the key double-spend lemma, we move on to the impossibility result in the Blockchain Trilemma, which states that under asynchronous communication, no consensus algorithm can simultaneously achieve (1) fault-tolerance, (2) resource-efficiency, and (3) full transferability. We provide a proof sketch below.

Proof sketch: Impossibility. We proceed by contradiction. Suppose that there is a consensus algorithm $(\mathcal{G}, \mathcal{C})$ that achieves all three of the desired properties. Pick two distinct majorities of agents, S and S'. As in the proof of the double-spend lemma, we define $A = S \cap S'$, $B = S \backslash S'$, and $C = S' \backslash S$. The basic logic of the deviation pursued by agents in A is that they will transfer

their entire balance on the ledger to B but also transact with C in order to receive some additional utility.

Let x be an outcome such that agents in A transfer their entire balance on the ledger to agents in B, that is, $t_n(x) = -v_n$ for all $n \in A$. Similarly, let x' be an outcome in which agents in A transfer their entire balance on the ledger to agents in C. By Assumption 2, such a pair of outcomes x, x' exists. The second part of Assumption 2 implies that there exists a state of the world $\theta \in \Theta$ such that the set of individually rational transactions is precisely $\mathbf{y}(x) \cup \mathbf{y}(x')$.

Note that if agents in A behave honestly and consensus is reached on outcome x when the state of the world is θ, each $n \in A$ receives utility

$$U_n^H = \sum_{y \in \mathbf{y}(x)} u_n(y|\theta) + v_n + t_n(x) = \sum_{y \in y(x)} u_n(y|\theta)$$

(since they spend their entire balance on the ledger, $t_n(x) = -v_n \; \forall \; n \in A$). By contrast, if agents in A behave honestly until the transactions $\mathbf{y}(x)$ occur, and then they behave dishonestly by engaging in a deviation that causes the transactions $\mathbf{y}(x')$ to occur as well, they receive utility of

$$U_n^D \geq \sum_{y \in \mathbf{y}(x) \cup \mathbf{y}(x')} u_n(y|\theta) > \sum_{y \in \mathbf{y}(x)} u_n(y|\theta) = U_n^H.$$

The first inequality holds because no matter what outcome eventually becomes a consensus, agents in A can never lose more than their entire balance on the ledger (and we have normalized the minimum ledger balance to zero). Therefore, they must receive at least $\sum_{y \in \mathbf{y}(x) \cup \mathbf{y}(x')} u_n(y|\theta)$. The outcome x' is individually rational, so we can conclude that $\sum_{y \in \mathbf{y}(x)} u_n(y|\theta) > 0$ for all $n \in A$, since in the outcome x', agents in A spend their balances v_n on the ledger.

By construction, both outcomes x and x' are individually rational in state θ. By the full transferability property, outcome x (resp. x') must then be implementable with positive probability under the consensus algorithm $(\mathcal{G}, \mathcal{C})$ when agents in S (resp. S') are non-faulty and the state of the world is θ. We can then apply the double-spend lemma (Lemma 2): the consensus algorithm $(\mathcal{G}, \mathcal{C})$ is fault-tolerant, and Assumptions 1, 3, and 4 are assumed to hold. Therefore, Lemma 2 implies the existence of a deviation for agents in A such that if, after nodes in A have communicated with B honestly and reached agreement on x, they still have not received any messages from C, they can attempt to fool C into deciding on the outcome x'. With positive probability, nodes in C are in fact fooled and decide on outcome x', yielding a preferred set of transactions $\mathbf{y}(x) \cup \mathbf{y}(x')$ for all agents in A. By the resource-efficiency property, it is costless for agents in A to engage in this deviation ex-ante, so the deviation yields higher expected utility than honest behavior for all agents in coalition A.

We have shown that by assuming the existence of a fault-tolerant, resource-efficient consensus algorithm that achieves full transferability, we can derive a profitable deviation for some coalition of agents, contradicting Condition 2 (which, by the fault-tolerance assumption, must hold whenever

any majority of agents have non-faulty nodes). Therefore, such a consensus algorithm cannot exist. □

4 Digital Record-Keeping in Practice

In this section, we address the ways in which our model is able to capture the relevant elements of digital ledger design and show how it addresses the fundamental issues. We begin by describing how our model environment encapsulates a wide variety of record-keeping systems as well as possible deviations from honesty within those systems. Then, we discuss how the Trilemma relates to three types of ledgers often used in reality: centralized ledgers, proof-of-work blockchains, and proof-of-stake blockchains. Finally, we discuss how our model applies to ledgers in which not all transactions are public.

4.1 Digital ledgers and the theory of consensus

We have adapted the theory of consensus pioneered by the seminal work of Lamport, Shostak, and Pease (1982) to a game-theoretic setting suitable for studying the issues that arise in digital record-keeping. We now discuss the connection between this theory and record-keeping.

The consensus problem in record-keeping: To understand the connection between the abstract theory of consensus algorithms and the design of digital record-keeping in practice (especially using blockchains), it helps to keep in mind what a digital ledger actually is: it is a sequence of entries, each one consisting of some data. The *state* of the ledger can be determined by reading the sequence of entries. In reality, a ledger entry is often represented by a "block" in a *blockchain*, which is a full copy of the ledger. Each node in the network keeps track of updates to the ledger. When a node "decides" on a block, it adds that block to its internal ledger, considering it to be final and updating its current view of the ledger's state. A node should not decide on a new block unless it is convinced that all others will eventually decide on that same block as well (i.e., unless there will be a consensus on that block).

In our model, all nodes agree on an initial state of the ledger. We consider a one-time update of the ledger, in which nodes must reach agreement on the addition of a single entry. An entry in the ledger corresponds to a collection of transactions. The state of the ledger, in turn, is just the distribution of ledger "balances." Balances on the ledger could represent, for example, continuation payoffs that agents will receive in some dynamic game that they play (as in Abreu, Pearce, and Stacchetti, 1990).[23]

The consensus problem is particularly salient in blockchain-based systems. When several conflicting new entries to the ledger have been proposed, a *fork* is created in the blockchain. Each possible new entry represents a "branch" of the fork. The consensus problem is simply the problem of resolving a fork. All nodes should decide on the same branch – a node that decides on a branch that does not eventually become a consensus will have accepted transactions that are not viewed as

[23] See Online Appendix D for additional discussion of this point.

final by others. Hence, the agent who owns that node may accept payments that are not recognized. As we will discuss, the double-spend problem is intrinsically related to this issue.

The generality of the framework: The problem of designing a consensus algorithm is effectively that of designing a voting system in which agents must provide proof of their voting power. Each node aggregates the votes it receives in order to determine whether a block should be added to the ledger. In our three examples, the requisite proofs are:

- **Centralized**: A single node (the ledger's owner) must authenticate and finalize transactions using its signature.
- **Proof-of-Stake blockchain**: Nodes known as "validators" vote by proving ownership of a certain quantity of tokens they have staked as collateral.
- **Proof-of-Work blockchain**: Nodes vote by proving they have solved computationally difficult problems, so voting power is allocated in proportion to computational expenditures.

Assumption 3 guarantees that our model can encapsulate all of these systems. Clearly, it allows us to study systems based on proof-of-work. The designer can require agents to pay a cost before their nodes send certain types of messages (i.e., votes). It also allows us to consider centralized record-keeping systems: the proof-of-identity assumption implies that the ledger's owner can send signed messages authorizing transactions. Finally, proof-of-identity also allows us to study proof-of-stake systems. If a node can send a signed message proving its owner is agent n, that message also constitutes a proof of its ledger balance v_n.

From static consensus to dynamic record-keeping: As in previous work, we study a static model in which the objective is to design an algorithm permitting nodes to reach agreement on a value (i.e., a set of transactions). This approach may, at first glance, appear puzzling: it might not be entirely clear how the abstract problem of inducing consensus on a value in a static model is related to practical issues of designing digital record-keeping systems in dynamic environments.

An ensemble of nodes can construct a blockchain through repeated consensus on updates to a state (i.e., the repeated addition of blocks). As we have discussed, updates to the ledger in our model could represent updates to a vector of continuation values in a dynamic game. In computer science, these types of dynamic updating problems are often referred to as *state machine replication* problems (see Schneider, 1990). A fundamental building block of this problem, of course, is the process by which nodes reach consensus on a single block to be added to the ledger. The algorithms used to solve this problem are precisely those developed by the distributed consensus literature. This is why previous work (e.g., Narayanan et al. 2016) has treated the static consensus problem as central to digital record-keeping.[24]

The resolution of forks in a dynamic environment can be thought of similarly to that in a static environment. Figure 5 depicts a situation in which there are two branches of a fork, one of which is longer than the other. Some nodes may have already decided to accept one of the two branches

[24]Garay and Kiayias (2020) also comment that, under certain conditions, the canonical static consensus problem is equivalent to a dynamic problem of achieving consensus on updates to a ledger ("Nakamoto" consensus).

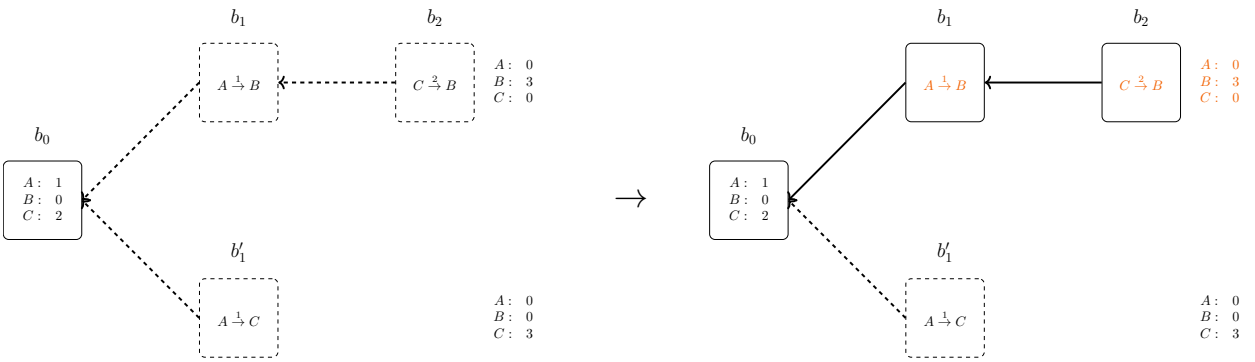

Figure 5: An example of a double-spend in a dynamic environment. Agents agree on an initial state of the ledger, in which (A, B, C) have balances $(1, 0, 2)$. They need to reach consensus on either the top branch, which contains two blocks, or the bottom branch, which contains a single block. Eventually, they reach agreement on the top branch, and the state is updated to $(0, 3, 0)$.

at some point in the past. Each branch, however, corresponds to a single set of transactions and a single terminal state, so the process of reaching consensus on one of the two branches can be thought of as the static consensus process in our model.

4.2 Types of attacks on the ledger

Digital record-keeping systems are subject to several types of attacks. In this section, we provide a brief overview of attacks that could occur on digital ledgers. This discussion will, in particular, highlight the role of our model's assumptions in determining the sets of attacks that are possible and the security measures that can be implemented against those attacks. We will argue that each type of attack (other than double-spending) is either easy to prevent or not relevant within our model, hence justifying our focus on double-spends. We will also discuss how the double-spend we derive in Lemma 2 is representative of double-spends often observed in reality.

Stealing: A digital ledger must prevent agents from simply directly stealing others' balances on the ledger. That is, users of the ledger should not be able to spend tokens held by others. This is typically easy to guarantee using cryptographic methods. Agents cannot spend a token held in a particular account unless they provide proof that they know a secret "password" associated with that account. In order to accomplish this in practice, digital ledgers use public-key encryption schemes. Assumption 3 in our model allows the designer to effectively replicate these cryptographic security measures: the designer can require agents to include their unforgeable signatures (i.e., provide proof-of-identity) whenever they attempt to spend balances on the ledger.

Denial-of-service: An attacker might want to engage in censorship, preventing certain agents from entering transactions in the ledger. In a centralized system, this is a difficult problem to circumvent: the ledger's owner can always prevent a particular agent from transacting (or extract a fee from the agent in exchange for including their transactions). On the other hand, in a decentralized system, a single agent is typically unable to prevent others from transacting: as long as other nodes follow the consensus algorithm honestly, even if one agent fails to vote in favor of

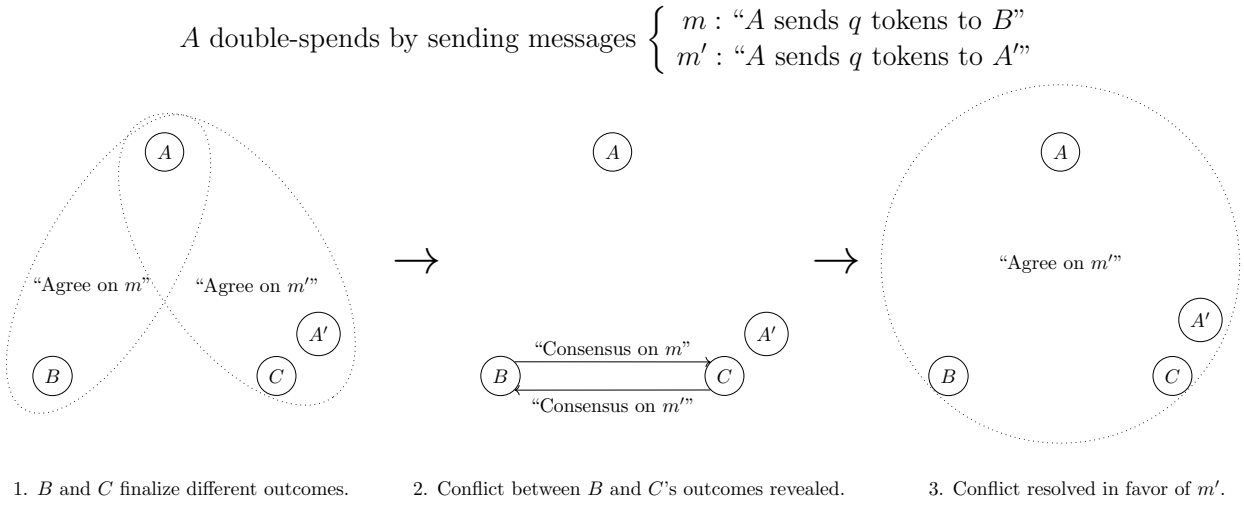

Figure 6: Alice double-spends by sending tokens to Bob in message m, and then generating another message m' in which she sends tokens to herself (i.e., to another node A' that she owns). Alice and Bob reach agreement on m, whereas Alice and Carol reach agreement on m'. Eventually the conflict resolves in favor of m'.

a transaction, others will do so, leading to its inclusion in the ledger. Our model incorporates the possibility of such denial-of-service attacks (censorship) because nodes are always permitted to ignore messages they received in the past. That is, Assumption 3 does not permit agents to prove they *have not* seen a message. However, our assumptions on preferences imply that agents would never want to censor transactions, since they care about only their own transactions. We view a study of denial-of-service attacks as important work for future research on decentralized record-keeping.

Double-spends: In the double-spend attack we considered in our motivating example, Alice convinced two other agents, Bob and Carol, to accept payment using the same tokens. Double-spend deviations in reality often take a slightly different form: Alice would instead send tokens to Bob (and convince him to accept them) while also sending those same tokens to a different account that she holds, aiming to induce consensus on the second transaction. Our model accommodates these types of double-spend deviations as well. Instead of convincing Carol to accept tokens herself, Alice could instead attempt to convince Carol that she had sent tokens to another one of her accounts A' (represented by a different node). If Carol is convinced that tokens were sent to A', then Bob and Carol would be left to reconcile two different ledgers: one in which Alice sent tokens to Bob, and another in which she sent tokens to A'. If Bob and Carol eventually reach consensus on the latter, Alice will have succeeded in receiving goods from Bob while retaining her tokens. The double-spend lemma also implies that such deviations are feasible and succeed with positive probability. Figure 6 illustrates this type of deviation.

In a blockchain-based system, a double-spend can be accomplished by sending tokens to a recipient, waiting until that recipient accepts a block containing that transaction, and then creating a fork of the blockchain that does not contain the transaction. That is, the sender, who may control

multiple nodes in the network, proposes a new block (or sequence of blocks) and then attempts to induce consensus on the newly created branch of the blockchain. If the attacker succeeds, the initial payment will not be recognized by any node as final.

4.3 Mapping ledgers to the Trilemma

The discussion above shows that in the context of our model, double-spend attacks are the only types of attacks on the ledger that give rise to a need for incentive provision. In this section, we describe how in each of the examples of digital record-keeping we consider, the security measures put in place against double-spending require a violation of one of the properties in the Trilemma.

Centralized ledgers: Agents will accept a transaction as final whenever the ledger's owner (i.e., the record-keeper) approves it. In order to double-spend, then, the record-keeper simply has to send different transactions to different agents, causing them to regard conflicting transactions as final. The record-keeper is incentivized not to do so only by the prospect of losing some value promised to her in the future.

This corresponds to the record-keeper having some non-transferable value v in our model, which could represent, for example, rents that the ledger's owner will earn in the future. Hence, centralized ledgers violate full transferability. On the other hand, the consensus protocol used by a centralized ledger does not require proof-of-work, so it is resource-efficient. Furthermore, consensus can be achieved whenever the record-keeper behaves honestly. While not all majorities of nodes can achieve consensus, centralized ledgers tolerate faults in all nodes other than the record-keeper's.

Proof-of-stake blockchains: In many proof-of-stake systems, a block is finalized when a two-thirds supermajority of validators (weighted by their token holdings) votes in favor of it. It is then possible for a set of validators who hold a supermajority of voting tokens to double-spend: after spending tokens, they can send votes to other agents in which they send those same tokens to themselves.

Typically, proposals for proof-of-stake consensus algorithms specify that if validators are found to have voted on conflicting transactions, the tokens they staked as collateral are stripped (called a "slasher" punishment). Our model allows for such ex-post punishments, since the final consensus state can include transactions in which agents lose their balances on the ledger. However, in order for such punishments to be implementable, validators must not be able to transfer the tokens they set aside as collateral, so full transferability is violated. Proof-of-work is not used, so these systems are resource-efficient, and consensus can be achieved whenever a supermajority of validators acts honestly, so they tolerate some faults.

Proof-of-work blockchains: In proof-of-work blockchains, double-spends are disincentivized by rendering them expensive. Nodes must solve computationally difficult problems in order to create blocks. A node then effectively votes in favor of a ledger by creating a block and appending it to that ledger. Proof-of-work blockchains use a "longest chain rule" to determine the ledger's current state, so a block b is considered to be final if it is in the longest chain of blocks C.[25] A

[25] More specifically, a block is usually considered final if it is in the longest chain and there are sufficiently many

group of attackers can then revert the finality of a block b (and double-spend) by creating a chain of blocks C' that is longer than the consensus chain C and does not contain b. In order to succeed in creating a longer chain with high probability, attackers would need to control a majority of the network's computing power, so these are called 51% attacks. The cost of conducting a 51% attack is what our model captures by allowing for costly communication.

Clearly, proof-of-work blockchains are not resource-efficient. However, they allow arbitrary transfers of value (therefore satisfying full transferability). Moreover, any majority that acts honestly can achieve consensus on a new block, so proof-of-work blockchains are maximally fault-tolerant.

Collusion among attackers: One final point is worth noting regarding the nature of attacks on decentralized blockchains. In both the case of proof-of-stake and proof-of-work, attackers needed to be in control of several nodes at once for their double-spend attempt to succeed. Typically, an attacker with access to a single node will be unable to accomplish anything through dishonesty. Blockchains are therefore designed to prevent collusion among groups of attackers attempting to subvert the ledger, which is why we require coalition-proofness in our equilibrium concept. If only robustness to unilateral deviations were required, it would be easy to design a consensus algorithm that entirely prevents profitable dishonest behavior. More specifically, we use the Strong Nash concept of Aumann (1959), which permits the deviating coalition to make binding agreements with one another, because blockchain security protocols often envision situations in which several nodes are in the control of a single entity.[26]

4.4 Consensus without a public ledger

In our model, we effectively assume a public ledger. During the consensus process, agents have access to the *entire* set of transactions that are being added to the ledger. This may seem unrealistic in the particular case of centralized ledgers, despite the fact that we claim our model applies to such systems.

It is simple to extend our model to systems in which nodes can see only transactions in which they are involved. Given an outcome x in our model, define $\mathbf{y}_n(x) = \{y \in \mathbf{y}(x) : \ n \in S(y)\}$ to be the set of transactions in outcome x that involve agent n. Then, suppose that during the course of the game, instead of deciding on an outcome x, each node n instead decides on a set of transactions $\tilde{\mathbf{y}}_n$. That is, nodes do not necessarily see others' transactions – instead, they can agree to the proposed transactions in which they are involved. A consensus is a situation in which there exists an outcome x such that $\tilde{\mathbf{y}}_n = \mathbf{y}_n(x)$ for all non-faulty nodes n. In this case, all nodes agree to a set of transactions that are consistent with a particular entry in the ledger, despite the fact that, perhaps, none of them see that entry. With this reformulation of the communication game, none

blocks (called "confirmations") following it.

[26]The (more common) coalition-proofness concept of Bernheim, Peleg, and Whinston (1987), by contrast, considers situations in which deviating agents cannot commit to a deviation. Instead, a deviation is considered to be stable only if it is "self-enforcing," meaning that no sub-coalition has an incentive to alter their strategies given the agreed-upon deviation.

of our results would change, so our model can accommodate systems with private transactions.

As an example, consider a network of payments via bank accounts. A seller accepting a payment must verify that funds have been deposited in his account before delivering goods to a buyer. The seller therefore "decides" on a transaction when his banker informs him the transaction has been completed (which can be interpreted as receiving a "vote" from the banker), but he need not know anything about other transactions that are occurring simultaneously. Another example is cryptocurrency blockchains that use "zero-knowledge proofs" to provide transaction anonymity (e.g., Zcash). No node in the network can see others' transactions, but all nodes can verify payments they receive and determine whether a transaction has received enough votes to be considered final.

5 The Fault-Tolerance Requirement

The Blockchain Trilemma characterizes conditions under which a record-keeping system requires the provision of incentives against double-spending. In this respect, our model's key assumptions are (1) that the record-keeping system is fault-tolerant to some extent, and (2) that communication is asynchronous. The key intuition behind the proof is that under any fault-tolerant consensus algorithm in an asynchronous network, dishonest behavior can cause different nodes to draw different conclusions about the transactions that were added to the ledger. Necessarily, then, some node must accept a set of transactions that is not actually final.

In this section, we will first discuss the role of fault-tolerance in our model. Then, we will extend our notion of fault-tolerance to be much more general: while in our benchmark model we restricted faults to be situations in which some agents' computers are offline, we will extend our results to a setting in which faulty nodes can engage in arbitrary behavior. Section 6, in turn, will discuss the role of asynchronicity.

5.1 The role of fault-tolerance

If no degree of fault-tolerance were required, it would be possible to design a consensus algorithm under which all honest nodes are guaranteed to reach agreement on a finalized set of transactions. When nodes cannot be faulty, the designer can specify a protocol of the following form:

1. "Once all nodes have confirmed that they believe a particular set of transactions x should be added to the ledger, decide that x is final."

2. "After deciding on a final set of transactions x, do not revert that decision."

This is feasible because when no node is faulty, then each node n will *eventually* receive a message from all other nodes n'. After sending enough messages to one another, nodes will be able to reach agreement on a set of transactions to finalize.

Under such a protocol, no two honest nodes can ever decide on different outcomes x and x'. Moreover, even if some nodes act dishonestly after a consensus on some outcome x has been reached, nodes that act honestly will never be fooled into reverting the original set of finalized transactions.

Our existence result in the Blockchain Trilemma, in fact, proves that this protocol achieves consensus without need for incentives: we show that when the fault-tolerance requirement is given up, there exists a consensus algorithm that is resource-efficient and achieves full transferability.

In reality, though, computer networks are bound to have faults. The problem posed by a fault-tolerance requirement is that it is infeasible to wait for a confirmation of an outcome x from *all* other nodes. Faulty nodes cannot be relied upon to confirm any set of transactions, so non-faulty nodes must proceed without their participation. Fault-tolerant protocols must therefore permit a node n to make a decision even without receiving confirmation from some other node n'. In turn, this property can create a situation in which n and n' temporarily decide on different outcomes x and x', meaning that one of those outcomes will not be final.

As it turns out, a fault-tolerance requirement on its own is insufficient to render a protocol vulnerable to double-spend attacks. Section 6 will demonstrate why the asynchronicity assumption is also essential. For now, we will generalize our model to show that our conclusions apply to essentially *any* fault-tolerant system rather than just the ones we consider in our benchmark model, in which faulty nodes simply do not communicate.

5.2 A richer model of faults

In our benchmark model, we make a somewhat weak assumption: faulty nodes do not communicate, representing computers that are currently offline or have suffered from a crash. A "faulty" node could also represent a potential new user who has not yet joined the network. In reality, however, the design of consensus algorithms often takes into account other faults that nodes may experience. For instance, in some cases it is appropriate to account for the possibility that nodes may suffer from programming errors that cause them to behave erratically, sending random messages to others. In other cases, designers of digital ledgers worry that some agents might be malicious in the sense that they have an incentive to undermine the ledger, intentionally aiming to prevent consensus.

Our consensus framework can be easily extended to accommodate these considerations.

Assumption 1′. *There are two frictions in communication.*

1. *Messages are delivered with a* ***random lag*** *of at most Δ rounds.*
2. *Some agents have* ***faulty*** *nodes, which may exhibit arbitrary behavior.*

Agents with non-faulty nodes act strategically, taking the behavior of arbitrary nodes as given. Their strategies must remain an equilibrium no matter how faulty nodes behave (even if they act maliciously in order to prevent consensus). Furthermore, the consensus algorithm must be designed so that agents who follow the communication protocol reach agreement regardless of the behavior of faulty nodes. In this setting, we therefore strengthen Conditions 1 and 2, updating our definition of what it means for a set of agents S to achieve consensus.

Definition 2′. *A subset of agents $S \subset \mathcal{N}$* ***can achieve consensus*** *under the consensus algorithm $(\mathcal{G}, \mathcal{C})$ if, whenever agents in S have non-faulty nodes, the following two conditions hold:*

- *(**Condition 1′**) Agents in S are guaranteed to reach consensus on an individually rational outcome if they follow the communication protocol $\mathcal{C}$, regardless of how faulty nodes behave.*
- *(**Condition 2′**) Taking as given any feasible behavior of faulty nodes, no coalition of agents has an incentive to jointly deviate from the protocol $\mathcal{C}$.*

Condition 2′, clearly, is a substantial strengthening of our equilibrium concept. We expand on this definition in the Appendix (Definition A.4).

We must also modify our definition of fault-tolerance. When faulty nodes can exhibit a wider range of behaviors, it is no longer possible to require that any majority of agents be able to reach consensus. This is a well-known result in the distributed consensus literature (see Bracha and Toueg, 1985). However, in this case, consensus can typically be achieved as long as a supermajority of two-thirds of nodes follows the communication protocol.

Definition 3′. *A consensus algorithm $(\mathcal{G}, \mathcal{C})$ is* ***strongly fault-tolerant*** *if any subset of more than two-thirds of agents can achieve consensus (in the sense of Definition 2′).*

The rest of our model remains unmodified. We can prove all of the main results in our paper under the more stringent requirement that equilibria must tolerate arbitrary faults.

Proposition 1. *Under Assumptions 1′ and 2 - 4, Lemma 1 and Lemma 2 continue to hold. The Blockchain Trilemma holds as well, so long as the fault-tolerance requirement is replaced with strong fault-tolerance (Definition 3′).*

The proofs of the double-spend lemma and the impossibility result are in Appendix C. The proof of the existence result can be found in the Online Appendix.

The double-spend lemma (Lemma 2) holds because it only relied on the possibility that faulty nodes may shut down, which of course is a possibility that we allow for in the extended framework. The impossibility result of the Trilemma therefore goes through as well, since it depended only on that lemma. Proving the existence result in the Trilemma, on the other hand, requires that we construct different consensus algorithms from those used in the original proof, but again we are able to adapt methods from the distributed consensus literature.

The results in this section show that there is a sense in which, up to a point, it is simple to design a consensus algorithm that is robust to arbitrary types of non-strategic behavior. That is, the algorithm described in Section 5.1 can be generalized to this setting. However, a fault-tolerant consensus algorithm always creates the possibility of profitable dishonest behavior by strategic agents. Hence, the problem of designing a consensus algorithm that is robust both to non-strategic and strategic deviations is hard. The designer must always provide some type of incentive for honesty, which requires giving up full transferability or resource-efficiency.

Importantly, our impossibility result rests only on the weak assumption that computers may fail to communicate. By contrast, in this section we are able to prove our existence result under the much stronger assumption that attacks on the network may be malicious. Therefore, we actually prove our Trilemma under the most stringent possible assumptions on the behavior of faulty nodes.

6 The Asynchronicity Assumption

In this section, we address the second key property of communication that leads to the Blockchain Trilemma: asynchronicity (Assumption 4). Again, we will first discuss the role of this assumption and then examine what happens if the assumption is relaxed. We will show that asynchronicity is indeed essential for our results. However, we demonstrate that even when Assumption 4 is relaxed, a consensus protocol satisfying the three properties in the Trilemma does not scale well to large systems.

6.1 The role of asynchronicity

From our discussion in Section 5.1, it may seem as if it is hopeless to design a foolproof, fault-tolerant consensus algorithm. However, this is not actually the case: the assumption of asynchronicity is required as well. By assuming asynchronous communication, we impose that the designer cannot make the protocol rules contingent on Δ, the maximum possible message lag between nodes. That is, the protocol cannot provide instructions of the form:

1. "If more than Δ rounds pass without receiving a message from node n', label n' as faulty. Ignore n' thereafter."

2. "Once a confirmation of the set of transactions x has been received from all nodes that have not been labeled faulty, decide that x is final."

3. "After deciding on a final set of transactions x, do not revert that decision."

Note that it is equivalent to assume that agents do not have access to perfectly synchronized clocks that can tell them how many "rounds" to wait, hence the label "asynchronous."

This type of protocol circumvents the problem posed by the presence of faulty nodes: those who follow the protocol will never be left waiting for a message from a faulty node, because after Δ rounds have passed, any node that follows the protocol will learn the identities of faulty nodes and label them as such. Hence, nodes that follow the protocol do not have to wait for confirmations from all nodes (which is infeasible). Rather, it is feasible to wait for confirmation from all nodes that have not deviated from the protocol, and this process is guaranteed to terminate in finite time. If n and n' both follow the protocol honestly, then, they cannot decide on different sets of transactions x and x'. This is because n must wait for confirmation from n' before deciding (and vice-versa). Per the protocol instructions, after nodes have decided on a set of transactions, it is never reverted, so attackers cannot subvert the finality of transactions. Indeed, we will show that under synchronous

communication (i.e., when the designer knows Δ), it is possible to design a fault-tolerant and resource-efficient consensus algorithm that achieves full transferability (Proposition 2).

How does this protocol guarantee transaction finality? It permits those who follow the protocol to reach *common knowledge* of the identities of nodes that have disobeyed. In this sense, synchronicity is an extremely strong assumption on agents' information. After a set of nodes $S \subset \mathcal{N}$ have agreed on a collection of transactions x, they know that any node outside of S must either be faulty or otherwise disobedient (for malicious reasons, perhaps). Hence, after deciding on x, if nodes in S ever receive evidence that a node $n' \notin S$ believes a different set of transactions x' should be added to the ledger, then they know that n' must be controlled by an agent who disobeyed the protocol and can therefore safely ignore n'.

In reality, though, synchronous communication is far too strong an assumption for most applications: on most public networks, such as the internet, it is impossible to know precisely the length of transmission lags between *all* users of the network. In principle, transmission lags can also be extremely long – a new user of a network may join years after its inception. Achieving common knowledge of the identities of agents who disobeyed the protocol is therefore not achievable in practice. Asynchronicity, then, can be viewed as a perturbation of the information structure that precludes common knowledge of certain events. Hence, when a set of nodes S agrees that a set of transactions x is final, they cannot necessarily conclude that any $n' \notin S$ who disagrees with that conclusion is dishonest. The double-spend lemma proves this point: given a fault-tolerant consensus algorithm in an asynchronous network, it is *always* possible for two honest nodes to temporarily finalize different sets of transactions.

Attempting to circumvent asynchronicity: We have effectively assumed that message delays are exogenous properties of the communication network, but one might imagine that the designer could implement a scheme that incentivizes agents to alter the communication network and remove these frictions. In fact, the designer could provide certain record-keepers with rewards for staying online at regular intervals. Some proof-of-stake systems do exactly this: validators are rewarded for regularly sending messages, and if they fail to do so, they are removed. In principle, it would also be possible to reward some agents for ensuring their computers are free of faults.

While it may be possible to provide *some* important record-keepers with rewards to overcome communication frictions, however, what would be required is actually that *all* potential users of the network stay online at regular intervals. Above, we illustrate that our results require only the possibility that two honest agents disagree about the set of final transactions, which in turn is possible whenever they cannot be relied upon to communicate synchronously. It does not seem possible, in practice, to reward all potential users of a system for staying online continuously, especially because the set of users may be unknown ex-ante.

6.2 Synchronous settings and scalability

We next address how our results change when we relax the asynchronicity assumption (Assumption 4), which was key to the proof of many of our main results. In this section, we assume that

communication is instead synchronous.

Assumption 4′. *The maximum message delay Δ is known to the designer.*

Recall that the synchronicity assumption amounts to a restriction on the set of communication protocols the designer can choose. We formally define the class of permissible communication protocols in the Appendix (Definition A.10). The assumption of synchronous communication is perhaps appropriate in a private network of known participants who are guaranteed to be connected to one another at regular intervals. For example, a consortium of firms who build a network to track deliveries to one another (e.g., a *permissioned* supply chain management blockchain) would fall into this category. Nevertheless, we address this case for completeness.

When communication is synchronous, it is *always* possible to simultaneously achieve fault-tolerance, resource-efficiency, and full transferability (as long as it is possible to do so with a trusted mediator).

Proposition 2. *Under Assumptions 1′, 3, and 4′, there exists a consensus algorithm $(\mathcal{G}, \mathcal{C})$ that is strongly fault-tolerant, resource-efficient, and achieves full transferability.*

We prove this proposition (as well as Proposition 3 below) in the Online Appendix.

We are able to prove the result under the assumption that faulty nodes may act in arbitrary ways (which is clearly more difficult than the case in which they simply do not communicate). Our construction is related to the algorithm originally derived by Lamport, Shostak, and Pease (1980) in the context of the "Byzantine Generals" problem. The simple intuition underlying Proposition 2 is that in the deviation described by the double-spend lemma (Lemma 2), what permits dishonest agents to deviate is that those who fall victim to the attack cannot detect whether others have faulty nodes (e.g., Bob could not tell whether Carol's node was faulty). When communication is synchronous, by contrast, it is possible to do so: if a node n has not received a message from another node n' in more than Δ rounds, n can safely conclude that n' is either faulty or has strategically deviated from the protocol.

A primary concern facing designers of decentralized consensus systems is the *scalability* of the consensus algorithms they use.[27] Decentralized blockchains have so far not been proven capable of scaling to the same level as centralized systems. For example, while Bitcoin's payment system processes about seven transactions per minute, Visa's system can process several million. The algorithm constructed in Proposition 2 resolves the Blockchain Trilemma, but this result raises an important question: is the synchronous consensus algorithm scalable? We prove a negative result: as the number of agents using the record-keeping system grows, the time to consensus grows linearly as well.

Proposition 3. *Suppose that Assumptions 1′, 3, and 4′ hold. Then any consensus algorithm $(\mathcal{G}, \mathcal{C})$ that is strongly fault-tolerant, resource-efficient, and achieves full transferability must take at least $\frac{N}{3} \cdot \Delta$ rounds of communication to reach consensus.*

[27] For example, Vitalik Buterin has posited a trilemma in which blockchains cannot simultaneously be secure, decentralized, and scalable.

Proposition 3 says that when faulty nodes can behave in arbitrary ways, it is impossible to resolve the Blockchain Trilemma with any algorithm that runs in less than $O(N)$ time, where N is the number of agents. A scalable system, though, should require that consensus be attained in $O(1)$ time – the time for one agent's transaction to process should be (roughly) independent of the number of users. Note, additionally, that the time to reach consensus scales with Δ, the maximum time for messages to be delivered between nodes (which, in turn, depends on how long any node in the network can remain offline). Hence, the Trilemma is resolved only by sacrificing scalability. In a sense, then, if the designer is required to provide agents with a scalable record-keeping system, our results carry over even to settings in which agents have access to synchronized clocks.

The reason is that reaching agreement in the presence of faults is hard. To reach consensus, non-faulty agents must attain common knowledge of a statement of the form "all other non-faulty agents will decide that the outcome is x." Faulty nodes can act maliciously, making it as difficult as possible for non-faulty agents to realize exactly who has deviated from the consensus algorithm. This means, though, that when a number N^F of nodes are faulty, agents who act honestly must then check all possible *joint* deviations by coalitions of N^F nodes. As it turns out, this takes N^F+1 rounds of cross-checking via back-and-forth messages. Only then is it possible to guarantee that a consensus has been reached. Therefore, consensus cannot be reached in fewer than $\frac{N}{3}\Delta$ rounds of communication when there can be as many as $\frac{N}{3}$ faulty nodes (since a message is delivered in at most Δ rounds).

7 Conclusion

The fundamental problem in digital record-keeping is to develop a method by which agents can reach an agreement on an update to a ledger in the presence of faults. When an inherently trustworthy mediator is available, this problem is easy to solve: the mediator can report updates to the ledger to all of its users, guaranteeing they remain in agreement about the ledger's contents. When record-keepers are self-interested, however, they have incentives to equivocate and report mutually inconsistent outcomes to different sets of agents. In an economic environment, then, the provision of incentives for honesty is central to the digital record-keeping problem.

We have developed a framework to study the design of consensus algorithms in a setting without trustworthy agents (in contrast to the computer science literature, which assumes certain nodes will act honestly). Our main result is a Blockchain Trilemma, which proves that any consensus algorithm must give up fault-tolerance, resource-efficiency, or full transferability. The central tension faced by the designer is between fault-tolerance and incentive provision. The designer can eliminate the need for incentives by giving up on the fault-tolerance requirement, but in many important digital record-keeping applications, it is unrealistic to assume that agents' computers will always function properly. Hence, the designer must provide either ex-ante incentives for agents to behave honestly (by giving up resource-efficiency) or impose ex-post punishments on dishonest agents (by giving up full transferability).

We have shown that the logic underlying our result is quite general and extends to settings beyond our benchmark model. In particular, we are able to introduce arbitrary misbehavior by faulty nodes without changing our results. Moreover, we show that even if consensus were synchronous, the designer could achieve the goals of fault-tolerance, resource-efficiency, and full transferability only by designing a protocol that takes a prohibitively long time to run.

There are two promising directions future research might take. First, we have not characterized constrained-optimal consensus algorithms. The nature of such an algorithm would likely be quite specific to the relevant application, so we have not analyzed that problem in our general model. Second, in mechanism design, it is typical to study implementation protocols that guarantee *all* equilibria of a given mechanism yield a desirable outcome, while in our analysis, the designer is content with specifying a single equilibrium (i.e., communication protocol) that yields a desirable outcome. Future work could study the problem of designing communication games without "bad" equilibria, as implementation theory has done.

Bibliography

Abreu, D., D. Pearce, and E. Stacchetti (1990): "Toward a Theory of Discounted Repeated Games with Imperfect Monitoring," *Econometrica*, 58(5), 1041–1063.

Auer, R., C. Monnet, and H. Shin (2021): "Permissioned Distributed Ledgers and the Governance of Money," *Working Paper*.

Aumann, R. (1959): "Acceptable Points in General Cooperative n-Person Games," in *Contributions to the Theory of Games IV*, pp. 321–353. Princeton University Press.

Aumann, R., and S. Hart (2003): "Long Cheap Talk," *Econometrica*, 71(6), 1619–1660.

Ben-Or, M. (1983): "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pp. 27–30.

Ben-Porath, E. (1998): "Correlation without Mediation: Expanding the Set of Equilibrium Outcomes by 'Cheap' Pre-Play Procedures," *Journal of Economic Theory*, 80(1), 108–122.

——— (2003): "Cheap Talk in Games with Incomplete Information," *Journal of Economic Theory*, 108(1), 45–71.

Bernheim, B. D., B. Peleg, and M. Whinston (1987): "Coalition-Proof Nash Equilibria I. Concepts," *Journal of Economic Theory*, 42(1), 1–12.

Biais, B., C. Bisière, M. Bouvard, and C. Casamatta (2019): "The Blockchain Folk Theorem," *Review of Financial Studies*, 32(5), 1662–1715.

Bracha, G., and S. Toueg (1985): "Asynchronous Consensus and Broadcast Protocols," *Journal of the ACM (JACM)*, 32(4), 824–840.

Budish, E. (2018): "The Economic Limits of Bitcoin and the Blockchain," *Working paper*.

Castro, M., and B. Liskov (1999): "Practical Byzantine Fault Tolerance," *OSDI*, pp. 173–186.

Chiu, J., and T. Koeppl (2019): "The Economics of Cryptocurrencies – Bitcoin and Beyond," *Bank of Canada Staff Working Paper 2019-40*.

Cong, L. W., and Z. He (2019): "Blockchain Disruption and Smart Contracts," *Review of Financial Studies*, 32(5), 1754–1797.

Cong, L. W., Y. Li, and N. Wang (2021): "Tokenomics: Dynamic Adoption and Valuation," *Review of Financial Studies*, 34(3), 1105–1155.

Easley, D., M. O'Hara, and S. Basu (2019): "From Mining to Markets: The Evolution of Bitcoin Transaction Fees," *Journal of Financial Economics*, 134(1), 91–109.

Eliaz, K. (2002): "Fault Tolerant Implementation," *Review of Economic Studies*, 69(3), 589–610.

Fischer, M., N. Lynch, and M. Paterson (1985): "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM (JACM)*, 32(2), 374–382.

Forges, F. (1986): "An Approach to Communication Equilibria," *Econometrica*, 54(6), 1375–1385.

——— (2020): "Correlated Equilibria and Communication in Games," in *Complex Social and Behavioral Systems: Game Theory and Agent-Based Models*, pp. 107–118.

Gans, J., and N. Gandal (2019): "More (or Less) Economics Limits of the Blockchain," *NBER Working Paper 26534*.

Garay, J., and A. Kiayias (2020): "SoK: A Consensus Taxonomy for the Blockchain Era," in *Cryptographers' Track at the RSA Conference*, ed. by J. S. Springer.

Gerardi, D. (2004): "Unmediated Communication in Games with Complete and Incomplete Information," *Journal of Economic Theory*, 114(1), 104–131.

Gilad, Y., R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich (2017): "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *SOSP '17: Proceedings of the 26th Symposium of Operating Systems Principles*, pp. 51–68.

Halaburda, H., Z. He, and J. Li (2021): "An Economic Model of Consensus on Digital Ledgers," *NBER Working Paper 29515*.

Huberman, G., J. Leshno, and C. Moallemi (2021): "Monopoly without a Monopolist: An Economic Analysis of the Bitcoin Payment System," *Review of Economic Studies*, 88(6), 3011–3040.

Lamport, L., D. Shostak, and M. Pease (1980): "Reaching Agreement in the Presence of Faults," *Journal of the ACM (JACM)*, 27(2), 228–234.

——— (1982): "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, 4(3), 382–401.

Linial, N. (1994): "Game-Theoretic Aspects of Computing," in *Handbook of Game Theory with Economic Applications*, ed. by R. Aumann, and S. Hart, pp. 1339–1395. Elsevier.

Maskin, E. (1978): "Implementation and Strong Nash Equilibrium," *Unpublished manuscript*.

——— (1999): "Nash Equilibrium and Welfare Optimality," *Review of Economic Studies*, 66(1), 23–38.

Myerson, R. (1986): "Multistage Games with Communication," *Econometrica*, 54(2), 323–358.

Nakamoto, S. (2008): "Bitcoin: A Peer-to-Peer Electronic Cash System," *Whitepaper*.

Narayanan, A., J. Bonneau, E. Felten, A. Miller, and S. Goldfeder (2016): *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press.

Pagnotta, E. (2022): "Decentralizing Money: Bitcoin Prices and Blockchain Security," *Review of Financial Studies*, 35(2), 866–907.

Rubinstein, A. (1989): "The Electronic Mail Game: Strategic Behavior Under 'Almost Common Knowledge'," *American Economic Review*, 79(3), 385–391.

Saleh, F. (2020): "Blockchain without Waste: Proof-of-Stake," *Review of Financial Studies*, 34(3), 1156–1190.

Schilling, L., and H. Uhlig (2019): "Some Simple Bitcoin Economics," *Journal of Monetary Economics*, 106, 16–26.

Schneider, F. (1990): "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys (CSUR)*, 22(4), 299–319.

Sockin, M., and W. Xiong (2021): "A Model of Cryptocurrencies," *NBER Working Paper 26816*.

Toueg, S. (1984): "Randomized Byzantine Agreements," in *PODC '84: Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pp. 163–178.

A Formal Preliminaries

In this section, we set up the formal framework required for a precise statement of our results as well as our proofs.

A.1 Communication games and an equilibrium concept

We begin by defining the class of communication games considered in Section 2.2. First, we set up some notation. The set of outcomes will be denoted $\mathcal{X}$, and the subset of outcomes that are achievable by $S \subset \mathcal{N}$ is denoted $\mathcal{X}_S = \{(\mathbf{y}(x), \mathbf{t}(x)) : S(y) \subset S \ \forall \ y \in \mathbf{y}(x)\}$. If in a round k, a node's most recent decision was x, we write $d_{nK} = x$.

A *communication game* specifies a message vocabulary, the cost of each message, and the set of messages that each node is permitted to send as well as the set of outcomes on which it can decide in each round.

Definition A.1. *A **communication game** $\mathcal{G}$ consists of*

- *A vocabulary $\mathcal{M}$ of messages that nodes may send to one another, with each $m \in \mathcal{M}$ having an associated proof-of-work cost $\kappa(m) \geq 0$;*
- *A set of permissible messages $M_{nk} \subset \mathcal{M}$ that can be sent, and outcomes $D_{nk} \subset \mathcal{X}$ that can be decided on, by each node $n \in \mathcal{N}$ in each round $k = 1, 2, \ldots$ (which may depend on the history H_{nk} of messages that node n has exchanged through round k).*

When a set S of agents have non-faulty nodes, the resulting communication game played among those agents is denoted $\mathcal{G}_S$.

Recall that agents communicate until a round K such that all non-faulty nodes have decided on an outcome x^*, $d_{nK} = x^*$ for all non-faulty n. At this point, payoffs $v_n + t_n(x^*)$ are realized. The set of transactions realized in the game is

$$\mathbf{y} = \{y \in \mathcal{Y} : \exists \ x \in \mathcal{X}, \ k \in \mathbb{N} \text{ s.t. } y \in \mathbf{y}(x), \ d_{nk} = x \ \forall \ n \in S(y)\}.$$

Agents' payoffs are given by Equation 1.

Strategy spaces and expected payoffs: Here we detail the sets of strategies available to each agent. Agents program their nodes to follow a *behavior*, which specifies how the node communicates with others and when it will decide on an outcome, based on the information it possesses. In turn, the information held by a node in round k consists of the state of the world θ, which should be thought of as the node's initial input, and the history of messages it has exchanged with others up until round k.

We must first define the history of messages exchanged by a node n. In a round k, a node will send a set of messages $\hat{M}^S_{nk}$ and receive a set of messages $\hat{M}^R_{nk}$. We let $h_{nk} = \hat{M}^S_{nk} \cup \hat{M}^R_{nk} \subset \mathcal{M}$ be the set of messages exchanged by node n in round k, so that the history of messages exchanged

by n through round k is $H_{nk} = (h_{n0}, h_{n1}, \ldots, h_{n,k}) \in \left(2^{\mathcal{M}} \times 2^{\mathcal{M}}\right)^{k+1}$. We let $\mathcal{H}_{nk}$ be the set of histories that node n can exchange through round k.

In round k of communication, a node may send a message in $m_{n,n',k} \in M_{nk}$ to each other node $n' \in \mathcal{N}$. A permissible messaging strategy of node n towards n' in round k is then $\sigma^m_{n,n',k} : \mathcal{H}_{nk} \times \Theta \to M_{nk}$. We let $\sigma^m_n = \{\sigma_{n,n',k} : \mathcal{H}_{nk} \times \Theta \to M_{nk} \mid n' \in \mathcal{N}, k \in \mathbb{N}\}$ denote a full messaging strategy for agent n, and we define Σ^m to be the set of feasible messaging strategies for n.

Similarly, if D_{nk} is the set of outcomes on which that node n may decide in round k, then a feasible decision strategy for node n in round k is $\sigma^d_{nk} : \mathcal{H}_{nk} \times \Theta \to D_{nk}$, and a full action strategy is $\sigma^d_n = \{\sigma^d_{nk} : \mathcal{H}_{nk} \times \Theta \to D_{nk} \mid k \in \mathbb{N}\}$, with Σ^d_n denoting the set of action strategies. Then, the set of behaviors available to node n is $\Sigma_n = \Sigma^m_n \times \Sigma^d_n$.

Definition A.2. *A **behavior** σ_n for a node n consists of a messaging strategy $m_{n,n'}(H_{nk}, \theta) \in M_{nk}$, specifying which message it will send to node n', and a decision strategy $d_n(H_{nk}, \theta) \in D_{nk}$, specifying which action it will recommend to its owner, at each information set (H_{nk}, θ). A **communication protocol** $\mathcal{C} = \{\sigma_n\}_{n \in \mathcal{N}}$ is a profile of feasible behaviors for each node.*

A pure strategy for agent n is simply a behavior $\sigma \in \Sigma_n$ for that agent's node. We will sometimes denote a strategy profile by $\boldsymbol{\sigma} \in \prod_{n \in \mathcal{N}} \Sigma_n$. We let

$$V_n(\boldsymbol{\sigma}) = \mathbb{E}\left[\sum_{y \in \mathbf{y}} u_n(y|\theta) + v_n + t_n - \sum_{m \in \hat{M}_n} \kappa(m) \;\middle|\; \boldsymbol{\sigma}\right]$$

denote the expected payoff of agent n when the strategy profile is $\boldsymbol{\sigma}$ (where the expectation is taken over the state of the world θ as well as the lag in the delivery of messages, which in turn determine the messages m that are sent by each node, the set of transactions $\mathbf{y}$ that are realized, and the final consensus ledger update $\mathbf{t}$).

Equilibrium definition: If S is the set of non-faulty agents, we denote a profile of feasible behaviors of their nodes by $\boldsymbol{\sigma}_S = \{\sigma_n\}_{n \in S} \in \prod_{n \in S} \Sigma_n$. Under our baseline assumptions (Assumption 1), faulty nodes do not communicate at all, nor do they make any decisions. We denote this behavior by σ^0. Hence, the profile of behaviors of faulty nodes will be denoted $\boldsymbol{\sigma}^0 = \{\sigma^0\}_{n \notin S}$. The expected payoff of a non-faulty agent $n \in S$ under the behavior profile $\boldsymbol{\sigma}_S$ is denoted $V_n(\boldsymbol{\sigma}_S, \boldsymbol{\sigma}^0)$. These payoffs define the communication game $\mathcal{G}_S$ played among agents in S when agents $\mathcal{N} - S$ have faulty nodes. When a coalition $S' \subset S$ deviates from the prescribed strategy profile, we denote the altered strategy profile by $(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \boldsymbol{\sigma}^0)$, so that agents' expected payoffs are $V_n(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \boldsymbol{\sigma}^0)$. Using this notation, our equilibrium concept can then be expressed as follows.

Definition A.3. *A communication protocol $\mathcal{C} = \boldsymbol{\sigma} \in \Sigma^{\mathcal{N}}$ constitutes a **fault-tolerant equilibrium** for a subset $S \subset \mathcal{N}$ of agents if, whenever nodes in $\mathcal{N} - S$ are faulty, the behaviors $\{\sigma_n\}_{n \in S}$ constitute*

a Strong Nash Equilibrium of $\mathcal{G}_S$. That is, for all potential deviating coalitions $S' \subset S$,

$$\nexists\ \tilde{\boldsymbol{\sigma}}_{S'} \in \prod_{n \in S'} \Sigma_n \text{ s.t. } V_n(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \boldsymbol{\sigma}^0) \geq V_n(\boldsymbol{\sigma}_S, \boldsymbol{\sigma}^0)\ \forall\ n \in S'$$
$$V_n(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \boldsymbol{\sigma}^0) > V_n(\boldsymbol{\sigma}_S, \boldsymbol{\sigma}^0) \text{ for some } n \in S'.$$

Note that under this definition, agents are assumed to *know* the identities of faulty nodes.

Extension in Section 5.2: In Section 5.2, we permit a faulty node n to exhibit any arbitrary behavior in $\tilde{\sigma}_n^F \in \Sigma_n$. When the behavior profile of faulty nodes is $\tilde{\boldsymbol{\sigma}}^F = \{\tilde{\sigma}_n^F\}_{n \notin S} \in \Sigma_n$, the expected payoff of an agent $n \in S$ under the behavior profile $\boldsymbol{\sigma}_S$ is denoted $V_n(\boldsymbol{\sigma}_S, \tilde{\boldsymbol{\sigma}}_F)$. These payoffs define the game $\mathcal{G}_S(\tilde{\boldsymbol{\sigma}}^F)$. Similarly to the benchmark case, when a coalition $S' \subset S$ deviates from the prescribed strategy profile, we denote the altered strategy profile by $(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \tilde{\boldsymbol{\sigma}}_F)$, so that agents' expected payoffs are $V_n(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \tilde{\boldsymbol{\sigma}}_F)$. Using this notation, our equilibrium concept can then be expressed as follows:

Definition A.4. *A communication protocol $\mathcal{C} = \boldsymbol{\sigma} \in \Sigma^{\mathcal{N}}$ constitutes a **strongly fault-tolerant equilibrium** for a subset $S \subset \mathcal{N}$ of agents if the behaviors $\{\sigma_n\}_{n \in S}$ constitute a Strong Nash Equilibrium of $\mathcal{G}_S(\tilde{\boldsymbol{\sigma}}_F)$. That is, for all $\tilde{\boldsymbol{\sigma}}_F \in \prod_{n \notin S} \Sigma_n$ and all potential deviating coalitions $S' \subset S$,*

$$\forall\ \tilde{\boldsymbol{\sigma}}_F \in \prod_{n \notin S} \Sigma_n :\ \nexists\ \tilde{\boldsymbol{\sigma}}_{S'} \in \prod_{n \in S'} \Sigma_n \text{ s.t. } V_n(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \tilde{\boldsymbol{\sigma}}_F) \geq V_n(\boldsymbol{\sigma}_S, \tilde{\boldsymbol{\sigma}}_F)\ \forall\ n \in S'$$
$$V_n(\tilde{\boldsymbol{\sigma}}_{S'}, \boldsymbol{\sigma}_{S-S'}, \tilde{\boldsymbol{\sigma}}_F) > V_n(\boldsymbol{\sigma}_S, \tilde{\boldsymbol{\sigma}}_F) \text{ for some } n \in S'.$$

Intuitively, a set of strategies constitutes a strongly fault-tolerant equilibrium for a subset of non-faulty agents S if those strategies are coalition-proof regardless of how faulty nodes communicate.

A.2 The design of consensus algorithms

In this section, we adapt the theory of consensus algorithm design pioneered in computer science by Lamport, Shostak, and Pease (1980) to our economic environment. A consensus algorithm (chosen by the designer) consists of two objects: (1) a *communication game* played by agents, and (2) a *communication protocol* dictating how nodes should be programmed to communicate.

Definition A.5. *A **consensus algorithm** $(\mathcal{G}, \mathcal{C})$ consists of:*

- *A **communication game** $\mathcal{G}$ (as in Definition A.1);*
- *A **communication protocol** $\mathcal{C}$ (as in Definition A.2).*

In what follows, it will also be useful to define the set of outcomes achievable by a set of agents S that are individually rational in state θ. This set is

$$\mathcal{X}_S^{\text{IR}}(\theta) = \{x \in \mathcal{X}_S :\ \text{Inequality } IR \text{ holds in state } \theta\}.$$

Designing a consensus algorithm does not guarantee that the communication strategies specified by the designer will be an equilibrium. The concept of a consensus algorithm by itself, therefore, is economically vacuous. We define a *consensus set* to be a set of agents $S \subset \mathcal{N}$ who have incentives to behave according to the communication protocol even when all other agents are faulty and achieve consensus on an outcome $x \in \mathcal{X}_S$.

Definition A.6. *A subset of agents $S \subset \mathcal{N}$ is a **consensus set** of an algorithm $(\mathcal{G}, \mathcal{C})$ if the following two conditions hold:*

- *(**Condition 1**) Whenever nodes in S communicate according to the communication protocol $\mathcal{C}$ and the state of the world is θ, they eventually achieve consensus on an individually rational outcome $x \in \mathcal{X}_S^{IR}(\theta)$;*
- *(**Condition 2**) The behaviors $\mathcal{C}$ constitute a fault-tolerant equilibrium for agents in S.*

This definition formalizes Conditions 1 and 2 in the main text. A consensus set is simply a set of nodes that can achieve consensus.

Now that we have developed our formal preliminaries, we can define the three ideal properties of a consensus algorithm.

Definition A.7. *We define the following three properties of a consensus algorithm $(\mathcal{G}, \mathcal{C})$:*

- *(**Fault-tolerance**) Any set of agents $S \subset \mathcal{N}$ with $|S| > \frac{N}{2}$ is a consensus set of $(\mathcal{G}, \mathcal{C})$;*
- *(**Resource-efficiency**) The consensus algorithm does not make use of costly messages, $\kappa(m) = 0$ for all m in the message vocabulary $\mathcal{M}$ specified by $\mathcal{G}$;*
- *(**Full transferability**) For each consensus set $S \subset \mathcal{N}$ of $(\mathcal{G}, \mathcal{C})$, whenever nodes in S are non-faulty and the state is θ, when agents in S follow the protocol $\mathcal{C}$, they reach consensus on each $x \in \mathcal{X}_S^{IR}(\theta)$ with positive probability.*

A.3 Model assumptions

In this section, we provide technical details on some of our model's assumptions. Assumptions 1 and 2 have been fully specified at this point. On the other hand, the assumptions about the designer's capabilities, Assumptions 3 and 4 require a formalization. In providing details on Assumption 4, we will fully explain the synchronicity assumption in Section 6.2 (Assumption 4′) as well.

Types of proofs: The designer may impose that nodes cannot send any arbitrary message $m \in \mathcal{M}$. Instead, a node n may have access only to a feasible set of messages M_{nk} that it may send in round k and a feasible set D_{nk} of decisions that it may recommend in round k (as in Definition A.1). A proof is a restriction on the set of messages or decisions that a node is permitted to make: if a node cannot provide the proof required to send a message, it is not permitted to do so. Likewise, if a node does not have the required proof that a certain outcome was entered into the ledger, it

cannot decide on that outcome and send a signal to its owner proving that the ledger was updated. Hence, our assumptions about the proofs the designer can require amount to restrictions on the types of communication games $\mathcal{G}$ that the designer can specify.

Before proceeding, we first define two notions of a partial order on the set of message histories. We define the collection of messages $\hat{M}(H) \subset \mathcal{M}$ contained in a history $H_{nk} \in \mathcal{H}_{nk}$ by

$$\hat{M}(H_{nk}) = \{m : \exists\ k' \leq k \text{ s.t. } m \in \hat{M}^R_{nk}\}.$$

Then, we define a *subset order* $\preceq^s$ and for two histories of messages $H, H' \in \mathcal{H}_n$ (where $\mathcal{H}_n \equiv \bigcup_{k=0}^{\infty} \mathcal{H}_{nk}$), we define $H \preceq^s H'$ if $\hat{M}(H) \subset \hat{M}(H')$.

We also introduce a *concatenation order* $\preceq^c$ on the set of message histories. We define a message concatenation as a finite sequence of messages $\mathcal{M}$, so that the space of message concatenations is $\Xi(\mathcal{M}) = \bigcup_{J \in \mathbb{N}} \mathcal{M}^J$, with a generic element being $m_1 : m_2 : \cdots : m_J$ (where each $m_j \in \mathcal{M}$). Given an ordered set of messages $M = \{m_1, \ldots, m_J\} \subset \mathcal{M}$, we define the concatenation of that set to be $\xi(M) = m_1 : m_2 : \cdots : m_J$. The concatenation of a history of messages $H_{nk} \in \mathcal{H}_{nk}$ is defined as

$$\xi(H_{nk}) = \xi(h_{n1}) : \xi(h_{n2}) : \cdots : \xi(h_{nk}).$$

We define $H \preceq^c H'$ if $\xi(H')$ is at least as long as $\xi(H)$ and can be written as $\xi(H') = \xi(H) : \xi'$, where $\xi' \in \Xi(\mathcal{M})$ is any other sequence of messages.

Next, we formally define the designer's capabilities laid out in Assumptions 3 and 4.

Definition A.8 (Proof-of-identity)**.** *A node's feasible set of messages M_{nk} and decisions D_{nk} may depend on its identity n.*

Definition A.9 (Proof-of-receipt)**.** *A node's feasible set of messages M_{nk} and decisions D_{nk} are weakly increasing in the history of messages H_{nk} it has previously exchanged (in the subset order).*

Therefore, the set of messages that a node can send and the actions it can recommend, in general, must be a function of the node's identity and the history of messages it has exchanged. Assumption 3 therefore amounts to the restriction that the sets M_{nk} and D_{nk} are of the form $M_{nk} = M(n, H_{nk})$ and $D_{nk} = D(n, H_{nk})$, with $M(n, H) \subset M(n, H')$ and $D(n, H) \subset D(n, H')$ whenever the collection of messages in H' contains that in H, $H \preceq^s H'$. Assumption 3 can therefore be rewritten as:

Assumption A.1. *The designer can choose a communication game G such that:*

- *(**Proof-of-identity and proof-of-receipt**) The permissible sets of messages M_{nk} and decisions D_{nk} for node n in round k take the form $M_{nk} = M(n, H_{nk})$ and $D(n, H_{nk})$, where $M(n, H) \subset M(n, H')$ and $D(n, H) \subset D(n, H')$ whenever $H \preceq^s H'$.*
- *(**Proof-of-work**) Each message $m \in \mathcal{M}$ can be associated with an arbitrary proof-of-work cost $\kappa(m) \geq 0$, which the sender must pay in order to send m.*

Asynchronicity assumptions: Assumption 4 amounts to a restriction on the types of communication protocols $\mathcal{C} \in \prod_{n\in\mathcal{N}} \Sigma_n$ that the designer can choose. When the designer knows Δ, the designer may specify the message sent by nodes in each individual round and, moreover, may use the constant Δ in the specification of the communication protocol. On the other hand, when the designer does not know Δ, the designer does not know the length of a communication round, so it is not possible to condition nodes' instructions on the number of rounds that have passed. Additionally, the designer may not use the constant Δ in the definition of the algorithm. However, it is still possible for the designer to specify instructions based on the *order* in which nodes received messages, since that does not depend on the time interval between rounds of communication. That is, while it is not possible to condition instructions on a node's history H_{nk}, it is possible to condition instructions on $\xi(H_{nk})$.

Definition A.10 (Synchronicity)**.** *A communication protocol $\mathcal{C} \in \prod_{n\in\mathcal{N}} \Sigma_n$ is **synchronous** if each node's behavior is a function of the form $\sigma_n : \bigcup_{k\in\mathbb{N}} \mathcal{H}_{nk} \times \Theta \times \mathbb{N} \to M_{nk} \times D_{nk}$, where the last argument denotes the value of Δ, the maximum message lag.*

*A communication protocol $\mathcal{C} \in \prod_{n\in\mathcal{N}} \Sigma_n$ is **asynchronous** if each node's behavior is a function of the form $\sigma_n : \bigcup_{k\in\mathbb{N}} \mathcal{H}_{nk} \times \Theta \to M_{nk} \times D_{nk}$, such that:*

- *If $H \simeq^c H'$, then $\sigma_n(H, \theta) = \sigma_n(H', \theta)$;*
- *The message-decision pair $(m, d) = \sigma_n(H, \theta)$ does not depend on Δ.*

Assumption A.2. *The designer must choose an asynchronous communication protocol $\mathcal{C}$ (as in Definition A.10).*

B Benchmark Result (Lemma 1)

In this section, we prove Lemma 1. Note that in the proof, we do not use the fact that faulty nodes are restricted to not communicate at all – they are permitted to communicate in arbitrary ways. Hence, Lemma 1 carries over to the setting with arbitrary behavior by faulty nodes outlined in Section 5.2.

B.1 The mediated consensus algorithm

First, we formally describe a consensus algorithm with a trusted mediator. There is a special agent, the trusted mediator, who is assumed to follow the communication protocol without the need for incentives. We denote this agent by $\hat{n} \notin \mathcal{N}$. This agent, just like those in the model, knows the state of the world θ but not the set of faulty agents.[28]

The message space available to agents $n \in \mathcal{N}$ is extremely simple. For each possible type $\theta \in \Theta$, there is a corresponding message $m = \theta$. Hence, $\mathcal{M} = \Theta$. Agents are instructed to send the state θ

[28]The assumption that the mediator knows θ is inessential.

to the mediator in the first round, and they send no messages thereafter, so $m_{n,\hat{n}}(H = \emptyset, \theta) = \theta$, and $m_{n,n'}(H, \theta) = \emptyset$ for all other recipients n' and histories of exchanged messages H. In what follows, there are two cases to consider: the case of synchronous communication (Assumption 4′) and the case of asynchronous communication (Assumption 4), since we do not impose either assumption in the statement of the lemma.

Synchronous communication: In the case of synchronous communication (Assumption 4′), the mediator waits until round Δ to send a recommended outcome to agents. We say the mediator receives a valid input θ from agent n if, by round Δ, the mediator has received precisely one message $m = \theta$ from node n. Let S be the set of agents from whom the mediator received a valid input, and define $\hat{\theta}_n$ to be the message sent to the mediator by each $n \in S$. If all respondents sent the same state of the world to the mediator, i.e., $\hat{\theta}_n = \theta$ for all n who respond, then the mediator computes an outcome $x \in \mathcal{X}_S^{\text{IR}}(\theta)$ at random and sends the message x to all agents (so the message space for the mediator is the space of possible outcomes). Otherwise, if agents send inconsistent reports to the mediator, then the mediator reports nothing to agents.

Once an agent receives a message x from the mediator, the communication protocol specifies that they should program their nodes to decide on outcome x. This means that

$$d_n(H, \theta) = \begin{cases} x & n \text{ has received exactly one outcome } x \text{ from } \hat{n} \\ \emptyset & \text{otherwise} \end{cases}.$$

Asynchronous communication: When communication is asynchronous (Assumption 4), the mediator cannot necessarily wait until messages from all agents have been received. Then, instead of waiting Δ rounds, the mediator first waits until valid inputs have been received from at least a majority of agents. The mediator then follows the same procedure as in the case of synchronous communication (described above), and agents are instructed to respond to the mediator's recommendations in the same way.

B.2 Proof of Lemma 1

Proof. We define the mediated consensus algorithm in Appendix B.1. Under that algorithm, all agents announce the state of the world θ to the mediator, who then computes an outcome $x^* \in \mathcal{X}_S^{\text{IR}}(\theta)$ at random (recalling that the set of agents $\mathcal{N} - S$ with faulty nodes fails to report altogether).

A subset of non-faulty agents $S' \subset S$ cannot profitably deviate by coordinating on a different messaging strategy. If they report a state of the world $\tilde{\theta}$ different from the true state, agents in $S \backslash S'$ will report θ regardless. By the definition of the mediated consensus algorithm, in this case the mediator will not report an outcome back to agents, so consensus will not be reached. Similarly, if agents in S' coordinate on a decision x' different from x, agents in $S \backslash S'$ will not decide on x' and the payoffs $v_n + t_n(x')$ will not be realized.

It is also not possible for the entire set of non-faulty agents S to deviate profitably. They cannot fool the mediator by reporting a different state $\tilde{\theta} \neq \theta$ – the mediator is assumed to know

θ.[29] Additionally, since any outcome recommended by the mediator is individually rational, no agent will opt to remain silent and report nothing to the mediator.

The mediated consensus algorithm, by construction, achieves consensus on each individually rational outcome in state θ with positive probability. It does not make use of costly messages, and as we have shown, no matter which subset of agents S is non-faulty, it is optimal for agents to follow the protocol. The mediated consensus algorithm therefore achieves fault-tolerance, resource-efficiency, and full transferability, as claimed.

□

C Proof of the Impossibility Result

In this section, we present a full, formal proof of the impossibility result in the Blockchain Trilemma (Proposition 3). We begin by proving the double-spend lemma and then prove the impossibility result.

C.1 Proof of Lemma 2

We first state the double-spend lemma more precisely.

Lemma C.1. *Suppose Assumptions 1-4 hold. Let $(\mathcal{G}, \mathcal{C})$ be a consensus algorithm with overlapping consensus sets S, S' such that $S \cap S' = \emptyset$, $S \not\supset S'$, $S' \not\supset S$. Consider a pair of outcomes $x \in \mathcal{X}_S$, $x' \in \mathcal{X}_{S'}$ such that there exists $\theta \in \Theta$ such that, when agents in S (resp. S') are non-faulty and follow the protocol $\mathcal{C}$, they reach consensus on x (resp. x') with positive probability.*

Then for large enough Δ, there exist disjoint coalitions of agents $A = S \cap S'$, $B = S \backslash S'$, and $C = S' \backslash S$ such that in the game $\mathcal{G}_{S \cup S'}$, there is a deviation $\tilde{\boldsymbol{\sigma}}^{x \to x'} = \{\tilde{\sigma}_n^{x \to x'}\}_{n \in A}$ for nodes in A such that in state θ:

- *With some probability $p \in (0, 1)$, $A \cup B$ reaches consensus on the outcome x, and then $A \cup C$ reaches consensus on x';*
- *With probability $1 - p$, agents in A receive precisely the same payoff that they would have received if they had behaved honestly.*

Proof. The first step in our proof is to define the strategy outlined in the proof sketch of the double-spend lemma. We are given a consensus algorithm $(\mathcal{G}, \mathcal{C})$ and assume a game $\mathcal{G}_{S \cup S'}$ and two consensus sets S, S' with $S \cap S' \neq \emptyset$.

The behavior of nodes in A is to first communicate honestly. Then, if they reach consensus with nodes in B without ever receiving a message from nodes in C (or having evidence that nodes

[29] To see that this assumption is inessential, note that agents also would not have incentives to deviate jointly if the mediator chose the distribution over outcomes given (θ, S) so that for all $\tilde{\theta} \neq \theta$, there exists some $n \in S$ with $\mathbb{E}_n[\sum_{y \in \mathbf{y}(x)} u_n(y|\theta) + v_n + t_n(x)|\tilde{\theta}, S] < \mathbb{E}_n[\sum_{y \in \mathbf{y}(x)} u_n(y|\theta) + v_n + t_n(x)|\theta, S]$.

in B did so), they communicate with nodes in C as if they have never received any message from nodes in B (or from each other).

The nodes of agents in A communicate honestly at any time such that agreement on x has not been reached with members of B. If agreement on x has been reached, then nodes in A pursue the following strategy. Once node $n \in A$ decides on outcome x (say in round K), it sends a set of messages $M_{nK} = \{m_{n,n'}(\emptyset, \theta)\}$ to all nodes in S'. For $k > K$, recursively define $\tilde{H}_{nk} = (\tilde{H}_{n,k-1}, \tilde{h}_{nk} \cup M_{nk})$, where

$$\tilde{h}_{nk} = \{m : m \text{ was received by } n \text{ from } n' \in S' \text{ in round } k\},$$

$$M_{nk} = \{m_{n,n'}(\tilde{H}_{nk}, \theta) : n' \in S'\},$$

and $\tilde{H}_{nK} = \emptyset$. That is, nodes in A behave as if they have never received any message prior to round K and ignore all communication from nodes in B.

We now show that for nodes in C, under asynchronous communication, the history of messages exchanged is indistinguishable (with positive probability) from the history of messages that would have been exchanged in the instance where nodes in $\mathcal{N} - (S \cup S')$ and nodes in B are faulty and do not communicate. The only messages that would permit nodes in C to distinguish the history from one in which nodes in A behave honestly are those that were either (1) sent by nodes in B, or (2) were sent by nodes in A to C before round K.

By the definition of a consensus set (namely, Condition 1 in Definition A.6), if nodes in A and C do not communicate with any other node, they must eventually come to consensus on an outcome with probability one. Then there exists a number of rounds K' such that $A \cup C$ comes to a consensus on x' in K' rounds with positive probability. Under the assumption of asynchronous communication (as in Definition A.10), the probability of this event is independent of whether nodes in $A \cup C$ begin communicating in the first round or in round K.

Additionally, given that communication is asynchronous, there exists Δ large enough that, with positive probability, no message sent by B to C arrives before round $K + K'$ and no message sent by A to C before round K arrives before $K + K'$. Therefore, under this strategy, nodes in A and C come to consensus on x' with positive probability after nodes in A and B have come to a consensus on x. Denote this probability by $q \in (0, 1)$.

To summarize, the deviation for a node $n \in A$ is $\sigma_n^{x \to x'} = (m_{n,n'}^{x \to x'}, d_n^{x \to x'})$, where

$$m_{n,n'}^{x \to x'}(H_{nk}, \theta) = \begin{cases} m_{n,n'}(H_{nk}, \theta) & \nexists\ H' \subset H_{nk} : x = d_n(H', \theta) \\ m_{n,n'}(\tilde{H}_{nk}, \theta) & n \in S', \exists\ H' \subset H_{nk} : x = d_n(H', \theta) \\ \emptyset & n \notin S', \exists\ H' \subset H_{nk} : x = d_n(H', \theta) \end{cases}$$

$$d_n^{x \to x'}(H_{nk}, \theta) = \begin{cases} d_n(H_{nk}, \theta) & \nexists\ H' \subset H_{nk} : x = d_n(H', \theta) \\ d_n(\tilde{H}_{nk}, \theta) & \exists\ H' \subset H_{nk} : x = d_n(H', \theta) \end{cases}.$$

Note that under this strategy, nodes in A never decided on any outcome other than precisely the

one on which they would have decided under honest communication unless consensus on x has already been reached in the past. So if $q' \in (0,1)$ is the probability that outcome x is reached by S in state θ under honest communication, then with probability $1-q'$, all agents decide on exactly the same outcome as they would have under honest communication. Conditional on outcome x, then with probability q, outcome x' will occur as well. With probability $1-q$, consensus on x' is not reached, meaning nodes in A never decide on any other outcome, so transactions $\mathbf{y}(x)$ occur. Under honest communication, when transactions $\mathbf{y}(x)$ occur, the balances of agents in A on the ledger go to zero, so they will receive a payoff of zero when consensus is reached. Therefore, the probability p given in the statement of the lemma is $p = qq'$.

□

Note that the restriction in Assumption 1 that faulty nodes *must* not communicate is irrelevant. Under the weaker Assumption 1′ on faulty nodes' behavior, faulty nodes are still *permitted* to remain silent, so our argument remains valid for *some* possible behavior of faulty nodes, which is all we require. Therefore, we have also shown that the double-spend lemma holds as required by 1.

C.2 Proof of Proposition 3 (Impossibility)

Proof. We proceed by contradiction. Assume that $(\mathcal{G}, \mathcal{C})$ is a fault-tolerant, resource-efficient consensus algorithm that achieves full transferability. By the fault-tolerance assumption, there exist overlapping consensus sets S and S' with $S \cap S' \neq \emptyset$, $S \not\supset S'$, and $S' \not\supset S$. Again, let $A = S \cap S'$, $B = S - S'$, $C = S' - S$.

Claim C.1. *By Assumption 2 and the full transferability condition, there exists a state $\theta \in \Theta$ such that:*

- *Under an outcome $x \in \mathcal{X}_S^{IR}(\theta)$, agents in A transfer their entire balance on the ledger to agents in B, i.e., $\sum_{n \in S} t_n(x) = 0$ with $t_n(x) = -v_n$ for all $n \in A$;*
- *Under an outcome $x' \in \mathcal{X}_{S'}^{IR}(\theta)$, agents in A transfer their entire balance on the ledger to agents in C, as above.*

Proof. The first part of Assumption 2 guarantees the existence of such a pair of outcomes. There is an outcome $\tilde{x} = (\mathbf{y}(\tilde{x}), \mathbf{t}(\tilde{x})) \in \mathcal{X}_S$ in which agents in A transfer their entire balance to agents in B. Then, let x be the outcome corresponding to the set of transactions $\mathbf{y}(x) = \{y \in \mathbf{y}(\tilde{x}) : S(y) \not\subset A\}$. Similarly, there is an outcome $\tilde{x}' \in \mathcal{X}_{S'}$ such that agents in A transfer their entire balance to agents in C, and we can define x' to be the outcome corresponding to transactions $\mathbf{y}(x') = \{y \in \mathbf{y}(\tilde{x}') : S(y) \not\subset A\}$. The second part guarantees that there exists a state in which both x and x' are individually rational. We can simply choose a state θ in which the set of individually rational transactions is precisely $\mathbf{y}(x) \cup \mathbf{y}(x')$. □

By the individual rationality of x' in state θ, we have

$$\sum_{y \in \mathbf{y}(x')} u_n(y|\theta) + v_n + t_n(x') = \sum_{y \in \mathbf{y}(x')} u_n(y|\theta) \geq v_n > 0$$

for all $n \in A$. The equality above follows from the fact that we constructed x' to satisfy $t_n(x') = -v_n$ for all $n \in A$, and the first inequality follows from the definition of individual rationality.

Assumptions 1, 3, and 4 are assumed to hold, and the consensus algorithm we consider is fault-tolerant, so the double-spend lemma applies. By Lemma C.1, in state θ, there exists a deviation for agents in A in the game $\mathcal{G}_{S \cup S'}$ (i.e., when agents in $S \cup S'$ are non-faulty) such that with some probability p, in a situation in which only outcome x would have been realized under honesty, both outcomes x and x' are realized. With probability $1 - p$, the payoffs that agents in A receive are precisely those that would have realized under honesty.

We then need only consider what happens if agents in A engage in a deviation that causes all nodes in S to decide on outcome x and all nodes in S' to decide on outcome x'. The set of transactions realized must contain $\mathbf{y}(x) \cup \mathbf{y}(x')$, no matter what occurs after agreement is reached on x and x'. The payoffs of agents in A must then be at least

$$U_n^D \geq \sum_{y \in \mathbf{y}(x) \cup \mathbf{y}(x')} u_n(y|\theta) \ \forall \ n \in A.$$

Note that this inequality uses the fact that agents in A can never lose more than their entire balance on the ledger. Under honest communication, on the other hand, consensus would have been reached on outcome x, and agents in A would have received payoffs

$$U_n^H = \sum_{y \in \mathbf{y}(x)} u_n(y|\theta) + v_n + t_n(x) = \sum_{y \in \mathbf{y}(x)} u_n(y|\theta) < \sum_{y \in \mathbf{y}(x)} u_n(y|\theta) + \sum_{y \in \mathbf{y}(x')} u_n(y|\theta) \leq U_n^D \ \forall \ n \in A.$$

The second equality uses the fact that, by construction, $t_n(x) = -v_n$ for all $n \in A$. The first inequality uses the claim proven above that $\sum_{y \in \mathbf{y}(x')} u_n(y|\theta) > 0$.

The resource-efficiency condition implies that agents' payoffs when the deviation succeeds are exactly U_n^D, since there is no cost of sending messages. Given that $U_n^D > U_n^H$ for all $n \in A$, this deviation strictly benefits all agents in A, so honest communication according to the protocol $\mathcal{C}$ cannot be a fault-tolerant equilibrium for $S \cup S'$, violating the condition that $S \cup S'$ must be a consensus set (by the fault-tolerance property). Then we have shown that the assumption of a fault-tolerant, resource-efficient consensus algorithm that achieves full transferability leads to a contradiction. It must then be that no such consensus algorithm exists.

□

The proof of the impossibility result requires only the double-spend lemma and Assumption 2. Since the double-spend lemma has been shown to hold in the extension of our model where faulty nodes can exhibit arbitrary behavior, the impossibility result holds in that extension as well.