Blockchain Economics*

Joseph Abadi and Markus Brunnermeier

August 31, 2019

Abstract

Traditional centralized record-keeping systems establish a consensus based on trust in the record-keeper. Trust arises from the ability to incentivize honest reporting. Rents extracted by the record-keeper create an internal source of trust, allowing the system to be self-sufficient. Blockchains decentralize record-keeping, dispensing with the need for trust in a single entity. Some build a consensus based on externally verifiable resource costs (proof-of-work), whereas others do not (proof-of-stake). We prove a blockchain trilemma: it is impossible for any digital record-keeping system to simultaneously be (i) self-sufficient, (ii) rent-free, and (iii) resource-efficient. Record-keeping systems without rents or resource costs must ultimately rely on some external source of trust.

*We are grateful for insightful discussions by Gur Huberman and Will Cong, helpful comments from Zhiguo He, Stephen Morris, Ulrich Müller, Arvind Narayanan, Wolfgang Pesendorfer, Fahad Saleh, Raphael Auer, conference participants at 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 Blockchain and Cryptocurrencies Conference, the St. Louis Fed, and the NYU Intermediation Conference, and seminar participants at the SEC, Yale University, the Wharton School at the University of Pennsylvania, the Princeton Department of Computer Science, the Princeton Department of Economics, and the BIS.

1 Introduction

Traditionally, records have been maintained by centralized entities. Blockchain has provided us with a radical decentralized alternative to record information. It has the potential to be as groundbreaking as the invention of double-entry bookkeeping in fourteenth-century Italy. Blockchain could revolutionize record-keeping of financial transactions and ownership data.

The central problem in digital record-keeping is how to ensure agents come to a *consensus* on the true history of events. Consensus, in turn, requires that a ledger's record-keepers have the incentive to report honestly, i.e., the ledger should be devoid of fraud. These incentives can be provided in three ways. First, agents may face external punishments for dishonest behavior, which may be social, commercial, or legal in nature. In this case, the ledger is not self-sufficient: a failure of the external punishment mechanism results in a failure of the ledger. Second, record-keepers may be punished through a loss of rents if users of the system abandon it upon discovering fraudulent activity. Third, record-keepers may face physical resource costs to write on the ledger, as in most decentralized blockchains, rendering dishonesty unprofitable from an ex-ante perspective.

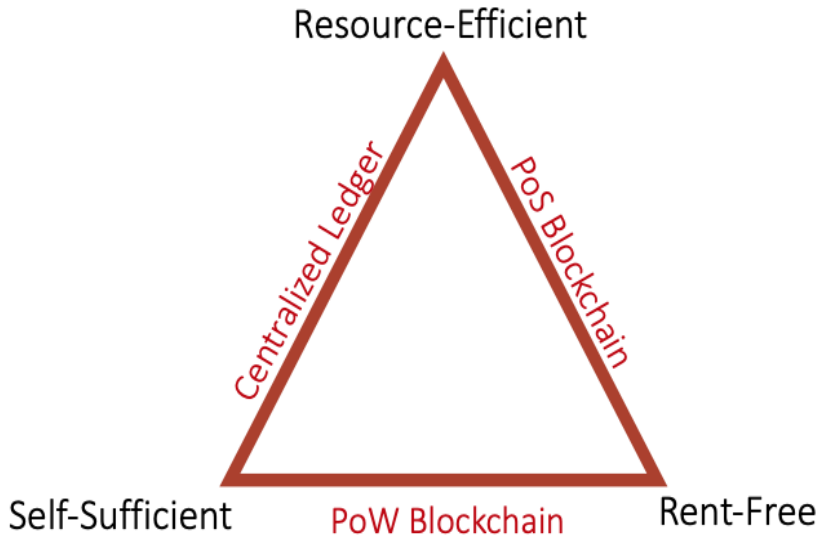

Figure 1: The blockchain trilemma.

In this paper we prove a "blockchain trilemma": consensus requires external punishments, rents, or ex-ante resource costs to write on the ledger. Hence, it is impossible for any digital ledger to simultaneously be (i) self-sufficient, (ii) rent-free, and (iii) resource-efficient.

In centralized record-keeping systems, consensus is achieved through *trust* in the

record-keeper. Agents simply ask the record-keeper to report the history of events to them, giving the record-keeper ample opportunity to benefit by misreporting. Trust can emerge from the record-keeping system itself if the record-keeper earns sufficient rents to keep it honest. Trust can also stem from external mechanisms: there may be legal authorities to punish fraud or commercial relationships that incentivize the record-keeper to keep its reputation intact.

Blockchains seek to minimize the role of trust in achieving consensus. A blockchain is a type of *distributed ledger* written by decentralized and usually anonymous groups of agents rather than known centralized parties. In a decentralized setting, users of the system may be faced with multiple internally consistent but mutually conflicting ledgers, making consensus all the more important. Record-keepers effectively "vote" on the history of events that they believe to be correct, and users of the system then aggregate those votes to determine the current state using a *consensus algorithm*.

The consensus algorithms used by some blockchains are completely objective in nature: they permit any two agents looking at the same set of ledgers to come to the same conclusion regarding the current state. The most popular blockchain consensus algorithm, called proof-of-work (PoW), has this property. Anonymous record-keepers (known as "miners") effectively vote on the true state (i.e., a chain of blocks) by extending that chain, which in turn requires an expenditure of computational power. When deciding the true state, agents simply look for the chain of blocks to which the greatest amount of computational power has been contributed. A new user of the system with no prior knowledge of the state, therefore, would come to the same conclusion as all others. Furthermore, this consensus algorithm disincentivizes misbehavior by making it costly for any agent to alter the state, so there is no need for trust in any particular entity.

Some blockchain consensus algorithms have sought to eliminate the resource costs entailed by the PoW algorithm. The most popular such consensus algorithm, known as proof-of-stake (PoS), instead allocates voting power based on the number of tokens held in each account. It is not costly to vote, so voting is secure only to the extent that the record-keepers can be punished for casting conflicting votes. Record-keepers are identified only by the pseudonyms attached to their accounts. Once the accounts that held a majority of the votes at some point in the past are empty, there is no way to guarantee that the owners of those accounts will not misbehave. They may conspire to use their historical majority to produce an alternate history that, to a new user, looks indistinguishable from the true history. To protect against such "long-range" attacks, new users need access to a trusted source of information in order to begin using the system, although no trust is required thereafter. Therefore, PoS blockchains require some external social trust and are not self-sufficient.

We formalize these intuitions underlying our trilemma by building a general model of record-keeping. Agents in our model keep track of transfers of digital tokens using a publicly viewable ledger. The tokens serve to facilitate the transfer of goods in some underlying mechanism, with few restrictions placed on the nature of that mechanism. Our

model can capture record-keeping by a single centralized entity, a group of known entities, or anonymous entities. Different systems of record-keeping correspond to different ways of allocating voting power on the ledger and forming a consensus on the current state. In a fully centralized record-keeping system, a single entity votes by signing the ledger with a digital signature. By contrast, in a PoW system, any agent may vote by paying a computational cost, and in a PoS system, voting power is given to token holders. Resource costs correspond to proof-of-work, rents are associated with surplus accruing to record-keepers, and external punishment (trust) comes from a social network in which agents have mutually beneficial relationships that may break down.

Importantly, in our model, nothing links agents directly to digital identities (e.g., accounts or digital signatures). Hence, agents may create alternate, internally consistent ledgers by simulating the behavior of honest record-keepers who observed the corresponding alternate history. In a centralized system, the record-keeper could, for example, circulate different ledgers with different groups, always maintaining with each group that any other ledgers it receives are forgeries. In a PoW system, any agent may create an alternate history by acquiring enough computing power, and in a PoS system, agents who held a majority of votes in the past may engage in a long-range attack. Formally, we prove that in the class of models we consider, a *mimicking lemma* holds: any agent may, in principle, create a ledger that appears indistinguishable from the true consensus ledger in terms of the votes cast on each ledger.

The trilemma is a consequence of the mimicking lemma. We consider the situation faced by a new user of the system who sees conflicting ledgers reflecting two different histories: the true history and an alternate history created by an attacker. If the two ledgers contain identical votes, the new user will need to ask her social connections to tell her the true history. For the new user to be able to rely on that information, she must trust her social connections to some extent. She may impose punishments by ceasing to participate in the underlying mechanism, destroying any rents that those connections derive from her participation, or by breaking off mutually beneficial social relationships. Therefore, in order to prevent her social connections from conspiring to defraud her, they must either face sufficiently large ex-ante costs of creating a conflicting ledger or sufficiently large ex-post punishments. In turn, ex-ante costs correspond to resource inefficiency, and ex-post punishments correspond to rents or external punishments (a failure of self-sufficiency).

After proving the trilemma, we discuss how other important issues related to blockchains could be addressed by extensions of our model. We outline the potential mechanisms through which decentralization of record-keeping can be beneficial. While our general model speaks to the ways in which incentive provision differs between centralized and decentralized record-keeping systems, it does not explicitly take a stance on how a centralized record-keeper's rents arise and how decentralization eliminates those rents.

We also make the important point that while blockchains guarantee transfers of *ownership*, some sort of enforcement is required to ensure transfers of *possession*. For example,

in a housing market, the owner of the house is the person whose name is on the deed, but the possessor of the house is the person who resides in it. The buyer of the deed needs to be certain that once she holds the deed, her ownership of the house will be enforced. In the stock market, the purchaser of a share has ownership of future dividends but not necessarily possession, since the delivery of dividends needs to be enforced. Broadly, blockchains can record obligations. Currency, for example, is special because its value derives only from the fact that it can be passed on, so no obligations need to be enforced. Punishing those who default on their obligations is another matter: it typically requires a trusted legal enforcement entity. We propose that it may be beneficial to bundle record-keeping and enforcement duties given that the punishment mechanisms that create trust in the enforcer also allow for trust in its role as record-keeper.

Related Literature. Our paper is related to the emergent literature on the economic properties and implications of blockchains. The paper most closely related to ours is Budish (2018), which studies the costs of incentivizing honesty for cryptocurrency blockchains in isolation, whereas our work compares the cost and incentive schemes required to secure both centralized and decentralized record-keeping systems. Biais et al. (2019) study coordination among miners in a blockchain-based system. They 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. We study forks as well, in the sense that agents in our model may attempt to defraud others by forking the blockchain. Cong and He (2019) focus mostly on the issue of how ledger transparency leads to a greater scope for collusion between users of the system, placing more emphasis on users' perspective than on record-keepers', which is where our interest lies.

Some of the recent literature on blockchains in economics focuses on the security and the costs of the system. Huberman, Leshno, and Moallemi (2017) study transaction fees in Bitcoin and compare that environment to one with a monopolistic intermediary. They emphasize the role of free entry and conclude that the blockchain market structure completely eliminates the rents that a monopolist would extract in an identical market, whereas we focus on the role of rents as a disciplining device for centralized record-keepers and compare that to the incentive provision mechanisms of decentralized blockchains. Easley, O'Hara, and Basu (2017) use a game-theoretic framework to analyze the emergence of transaction fees in Bitcoin and the implications of these fees for mining costs. The R&D race between Bitcoin mining pools is described in Gans, Ma, and Tourky (2018), who argue that regulation of Bitcoin mining would reduce the overall costs of the system and improve welfare. We focus on the key economic role of mining costs, which is to dissuade agents from creating alternate histories of events for their own benefit.

We also relate to the literature on cryptocurrencies. Chiu and Koeppl (2017) develop a macroeconomic model in which the sizes of cryptocurrency transactions are capped by the possibility of an attack on the blockchain and derive optimal compensation schemes for record-keepers. Schilling and Uhlig (2018) study cryptocurrency pricing in a monetary model and derive necessary conditions for speculation to occur in equilibrium. Pagnotta

and Buraschi (2018) derive a pricing framework for cryptocurrencies that explicitly accounts for the interplay between demand for the currency and the cryptographic security provided by miners.

Recent computer science literature has studied blockchain security extensively. Most papers in computer science, such as that by Gervais et al. (2016), study how to defend against "double-spend" attacks or other types of attacks that could be undertaken by a single individual who holds control over a large portion of the network's computing power. The conclusion of studies in the computer science literature is that a large fraction of the blockchain record-keepers must always play honestly in order for the network to be secure. In contrast, we do not assume any record-keepers are compelled to play honestly. Rather, agents are permitted to act and collude in arbitrary ways, and they have well-defined incentives to behave as the system's protocol dictates they should. Our model shows that the real cost of operating a PoW blockchain is intrinsically linked to the benefit of a successful attack. It also outlines the types of incentives required for a PoS blockchain to operate correctly.

The rest of the paper is structured as follows. Section 2 discusses the basics of blockchain technology and introduces some concepts and notation used in the model. Section 3 develops intuition for the trilemma by providing examples of the opportunities for fraud and the incentive schemes used in centralized record-keeping systems as well as PoW and PoS blockchains. Section 4 presents our general model of record-keeping and proves the blockchain trilemma. Section 5 discusses other issues related to blockchain from a less formal perspective. Section 6 concludes.

2 Digital Assets and Blockchain Technology

2.1 The digital record-keeping problem

The motivation behind blockchains, and digital record-keeping algorithms in general, is that it is often necessary to maintain ledgers that keep track of sequences of events. In particular, it is often important to determine whether one event occurred before or after another. A canonical example of the necessity of sequential record-keeping arises in the context of digital assets (often referred to as coins, or tokens).

In account-based digital token systems, users have secure accounts protected by *public key encryption*. With public key encryption, it is possible for users to sign messages with a signature that publicly proves they own an account without revealing the secret "password" (private key) that grants access to the account. Tokens are transferred by messages: the owner of an account can send a signed message indicating a transfer from her own account to another. Thus, it is not possible to spend others' tokens.

Unlike physical assets, however, the transfer of digital assets does not inherently preclude their reuse by the original owner. That is, there is no fundamental scarcity of digital assets. To see this point clearly, we consider a simple example. Suppose Alice

owns accounts 1 and 2, Bob owns account 3, and each account holds one token. The initial state is denoted $\omega_0 = (1, 1, 1)$. If Alice may transfer the token held in account 1 to Bob by sending a message m_1, what prevents her from sending the same token again to account 2 by sending another message m_2? Alice may want to do this, for instance, because she wants a third user Carol to believe that she has two tokens. One might say that as long as it is commonly known that m_1 preceded m_2, all users of the digital token system would simply ignore m_2 because they consider the tokens to be in Bob's account, concluding that the state is $\omega_1 = (0, 1, 2)$.

There are two issues with this reasoning. First, latency in the arrival of digital messages may result in Carol seeing m_2 *before* m_1 if the two messages were sent in close succession. Second, if Carol joins the system long after both messages were sent, she would be unable to discern which message came first. She would see two public messages attempting to send the same token but would be uncertain as to whether it currently resides in account 2 or account 3.

The problem of sequencing the two messages is known as the *double-spend* problem, which highlights the need for a stable consensus on the ordering of messages. It is the main issue that most consensus algorithms attempt to solve. We will work with this example throughout to build intuition for the concepts introduced.

2.2 What is a blockchain?

A blockchain is a digital ledger in which information is recorded sequentially in data structures known as blocks. A block consists of (1) a set of messages, (usually) (2) a *pointer* to another block, and (3) a *header*. In the context of a cryptocurrency (or other digital asset) ledger, the messages contained in blocks correspond to transfers of tokens from one account to another or seignorage received by accounts.[1] The blockchain is effectively a device to keep track of a *state*, i.e., the quantity of tokens in each account. Pointers serve to order the messages contained in blocks. The first block in a blockchain, known as the *genesis block*, has no pointer, and pointers on other blocks indicate immediate their predecessors in the blockchain.

Simply adding pointers to collections of messages does not solve the double-spend problem, however. In the context of the previous example, suppose that the token is in Alice's account in the genesis block b_0, and there is a block b_1 pointing to b_0 that contains message m_1. Then the blockchain $C_1 = \{b_0, b_1\}$ results in state $\omega_1 = (0, 1, 2)$. However, the existence of chain C_1 does not preclude the possibility that someone could create a block b_2 that points to b_0 and contains message m_2. This would create another chain $C_2 = \{b_0, b_2\}$ that conflicts with C_1 and implies instead that the state is $\omega_2 = (0, 2, 1)$, so Alice still owns two tokens. A situation in which there are two conflicting ledgers (chains of blocks) with a common genesis is known as a *fork*, and blocks on different branches of a fork are called *conflicting blocks*.

[1] These messages are collected into a data structure known as a *Merkle tree*.

When faced with multiple valid yet conflicting chains, the consensus algorithm dictates which chain agents should consider to be the current state. The outcome of the algorithm, in turn, depends on block headers. The form of these headers depends on whether the consensus algorithm is permissioned, proof-of-stake, or proof-of-work, so we discuss each case separately below.

2.3 Consensus algorithms

In our setting, the purpose of a consensus algorithm is to permit agents to agree on a *consensus state*. Consensus algorithms are essentially voting systems: when faced with multiple valid chains, agents can look at the "votes" cast on each chain in order to come to a consensus. We examine three distinct classes of blockchains that differ in their consensus algorithms: permissioned blockchains, proof-of-stake (PoS) blockchains, and proof-of-work (PoW) blockchains. These types of systems differ mainly in how they allocate voting power, and, as we will show later, in how they incentivize correct record-keeping.

A permissioned blockchain is one in which a known consortium of entities has full, unimpeachable power to update the ledger. When the blockchain is operated by a single known entity, it is called a private blockchain. A simple example of a private blockchain is one in which a monopolist maintains and updates the ledger of a transaction system it operates. The monopolist may add transactions to blocks and approve blocks by signing them with a digital signature (e.g., by proving that he has access to a secret password). As long as no other agent has access to this digital signature, the monopolist will be the only one able to alter the ledger. In principle, any agent who knows the monopolist's signature could verify whether the monopolist has equivocated by publishing conflicting ledgers, but agents still need some trusted source of information to learn this signature. Typically, the monopolist is the trusted source.

PoS and PoW blockchains are types of *public blockchains*, in which anonymous agents can become eligible to write on the ledger. The question, then, is how voting power to approve blocks should be allocated. In an ideal world, one might imagine a "one-person-one-vote" system in which all users of the ledger have equal voting power. However, this type of voting system is infeasible, because it is not costly for one person to acquire multiple digital identities (e.g., IP addresses) and pretend to be a larger group of people. This type of attack, which will play an important role in our analysis, is known as a *Sybil attack*. The possibility of Sybil attacks creates an *identity management problem*, which is what the PoS and PoW algorithms attempt to solve.

The PoS and PoW algorithms differ in how they allocate voting power. In PoS blockchains, voting power is allocated to accounts in proportion to the number of tokens held in those accounts. PoW blockchains, by contrast, allocate voting power to agents who prove they have solved intrinsically useless computational problems. Effectively, then, PoW allocates voting power in proportion to expenditures on computational

resources. The act of solving computational problems is typically known as *mining*, and miners are compensated for their expenses through *block rewards*. Mining expenditures and block rewards are both sizable in reality. For example, as of this writing, the Bitcoin block reward was worth over $125,000 and was distributed every ten minutes on average.

2.4 Blockchain security

In communication across anonymous digital channels, nothing intrinsically links votes to individuals. At best, votes are linked to accounts (as in PoS and private blockchains), but additional information is needed to determine which accounts belong to specific individuals. This feature of digital communication creates opportunities to deviate that are at the heart of our analysis.

In the context of a completely centralized private blockchain, a monopolist can cheat by creating two internally consistent ledgers with different genesis blocks, each with its own special permissioned account (digital signature) used to update the blockchain. It can then send these different ledgers to different groups. Nothing links those permissioned accounts to the monopolist, so even if a group discovers the ledger circulated by the other, it would not constitute proof that the monopolist had equivocated. The monopolist may always insist that the other ledger is an unsanctioned forgery.

In PoS systems, the problem is similar, as voting power is linked to accounts holding monetary tokens. As in a centralized system, a group of attackers may create a fraudulent genesis block and a new set of accounts owning all tokens in that block. The attackers would then have all of the voting power necessary to update that ledger as they chose, making it indistinguishable from a ledger used for transactions in the economy. The attackers could then proliferate this seemingly valid ledger to unsuspecting new users, who would need additional information to distinguish it from the true ledger used by others.[2] More realistically, attackers can attempt to acquire the private keys corresponding to accounts that held a supermajority of votes long in the past (e,g., on the black market). It is plausible that the accounts that held a supermajority in a block b_{past} are eventually emptied, so it could be cheap to acquire these keys and start a fork from block b_{past} that looks just as valid as the true consensus chain to new users. In this situation, it would also be impossible to say for certain that any agent had equivocated. This attack, known as a "long-range attack," is of principal importance in the study of PoS security.

Finally, in PoW systems, attackers can mimic the behavior of agents reporting honestly simply by paying a high enough computational cost to create blocks at the same pace as the rest of the network. Nothing about the attackers' blocks indicates that they were fraudulently created, as the votes on those blocks look identical to those on any other block. The consensus algorithm that PoW blockchains use is typically a "longest chain

[2]The genesis block is often hard-coded into the software client used to run a blockchain protocol. In order to fool unsuspecting new users, the attackers would have to create a modified version of the software client and tell those users that their version is the correct one.

rule": all users converge on the chain with the greatest amount of work committed to it, which typically coincides with the longest chain of blocks. Hence, an attacker can generate a consensus on a chain he privately created, so long as he controls a majority of the network's computational power. This type of "51% attack" is the primary concern in the security of PoW systems.

3 Examples of consensus algorithms

Formally, all three classes of consensus algorithms allow agents who know (1) the previous consensus state (given by a chain C) and (2) the current set of blocks (B) to come to an agreement on a new consensus state when blocks are added. A consensus algorithm can then be described by a function $g(C, B) = C'$. We will assume that, as in reality, the consensus algorithm depends only on the headers of each block. In particular, the header of a block b contains a message that we will call a *vote*, $v(b)$. The consensus algorithm is effectively a way to decide on the next state based on the previous state and any votes cast on newly formed blocks.

We now give a rough description of the three broad classes of consensus algorithms and outline the problems that may arise in each case. We will continue to work with our example in Section 2.1. For concreteness, we assume there are two periods, $t = 1, 2$. Alice and Bob are both present at $t = 1$ and know the true genesis block b_0 and the state ω_0, but Carol does not know this information and arrives only at $t = 2$.

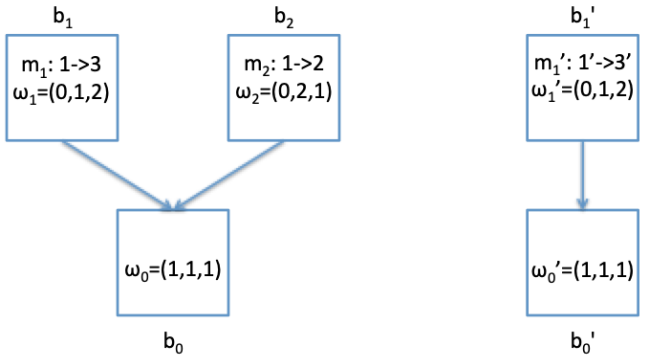

Figure 2: The setup in the example of Section 3.

Alice may also create a new genesis block at $t = 1$, b'_0, with state $\omega'_0 = (1, 1, 1)$.[3] We assume she owns all the private keys corresponding to the accounts in the new genesis

[3]There is another realistic interpretation of an attack involving a fraudulent genesis block. In our setting, this type of attack can be thought of as a metaphor for "long-range attacks" in proof-of-stake systems, where attackers cheaply acquire the private keys to defunct accounts that held voting tokens a long time in the past.

block, but this fact is not publicly known. She may also send a message m_1' in which account 1 sends a token to account 3 on this new chain, leading to state $\omega_1' = (0, 1, 2)$ if it is included in a block b_1' following b_0'. Therefore, these two blocks may be present in addition to b_1 and b_2, which contain messages m_1 and m_2 as in Section 2.1. Figure 3 illustrates this setup.

Note that a consensus algorithm $g(C, B)$ does not necessarily guarantee consensus when some agents do not know the initial state. In our example, Carol may require information about the initial state when she arrives in order to arrive at a consensus. We assume she is socially connected to Alice or Bob, who may communicate with her through private messages.

3.1 Private blockchain

For simplicity, here we consider a situation with a monopolist who holds a single non-transferable voting token in an account. This voting token effectively constitutes the monopolist's signature, as the monopolist is the only one in possession of the private key that can be used to sign messages from the corresponding account. We denote a vote on a block b by a vote $v(b) = 1$.

The consensus algorithm gives any agent with knowledge of the genesis block the ability to deduce the current state. Agents simply look for the longest chain in which the voting token has voted on every block. However, if the monopolist has voted on conflicting chains, then agents detect this deviation, and consensus is lost (or, equivalently, agents abandon the system). Formally, the algorithm chooses $g(C, B) = C'$ to be the longest chain C' (containing the known genesis) such that $v = 1$ for every block in C'. If there is more than one such chain, instead choose $g(C, B) = \emptyset$, regardless of the initial chain C.

There is no a priori way to distinguish between two blockchains with different genesis blocks, though. In other words, agents lack prior knowledge about the monopolist's signature, which opens up the possibility of fraud.

In our example, we will assume that Alice is the monopolist. She is able to choose whether to include m_1 or m_2 in the blockchain with genesis b_0, but it is impossible to choose to include m_1 and then reverse that decision by creating a block in which m_2 is included, for example. Therefore, if Bob is to believe he has received payment, Alice must vote on b_1 at $t = 1$, and this transaction will be irreversible from Bob's perspective.

Carol may still be fooled, however. Suppose that Alice casts vote $v = 1$ on block b_1' as well. When Carol arrives, she sees one blockchain in which the current state is ω_1 and another in which it is ω_1', with no way to distinguish between the two other than through social messages. If Carol trusts Alice (the monopolist) to provide this information, Alice can claim that genesis block b_0 is a forgery, and her true signature is on block b_0'.

How could Alice be incentivized to provide Carol with the correct state? Carol can punish Alice for lying by abandoning the system if the lie is detected. If Alice extracts rents from users of her monopolistic system, she may prefer to keep Carol as a user

long-term rather than deceiving her in the short term.

3.2 Proof-of-Stake blockchain

In a proof-of-stake system, voting power is typically assigned to monetary tokens. When determining which message is ultimately included in the blockchain, voting power is assigned based on token holdings before transfers are executed rather than after. That is, in block b_0, Alice has $\frac{2}{3}$ of the voting shares, and Bob has the remainder, no matter which message is chosen. Proof-of-stake blockchains typically operate using a supermajority rule. A vote v in this context consists of the identities of the accounts that vote in favor of a particular block. We say a fraction α of tokens vote on a block if the fraction of tokens held in accounts voting on that block is α. The consensus algorithm chooses $g(C, B) = C'$ such that

1. C' is the longest chain containing C such that at least two-thirds of all tokens vote on each block in C'; and
2. The fraction of tokens that have voted on conflicting blocks is less than $\frac{1}{3}$.

When condition (2) fails, it means that at least $\frac{1}{3}$ of tokens have been used to cast multiple conflicting votes. If C is contained in a block tree on which such conflicting votes have been cast, instead, $g(C, B) = \emptyset$. Sufficiently severe equivocation is punished by a loss of consensus in the same way as in the case of a private blockchain, although with a PoS blockchain there is some tolerance for a small fraction of equivocating voters.[4]

It is easy to check that no state contained in a chain C can be reversed by this updating rule. For instance, if Alice and Bob both use a token to vote on b_1, Alice must then use all $\frac{2}{3}$ of her tokens to vote on a block containing b_2 if she wants ω_2 to have any chance of being considered a consensus state. This would require that Alice use at least one account to vote on both ω_1 and ω_2, though, so instead of creating a situation in which Carol believes ω_2 could be the consensus state, Alice destroys consensus by equivocating. As in the case of private blockchain, then, knowledge of an initial state is enough to generate irreversible consensus. Nevertheless, this system is still subjective and thus exposed to the same type of attack as the private system: Alice may use her supermajority in state ω_0' to vote on ω_1', which will fool Carol if Alice communicates inaccurate social messages. For instance, Alice could tell Carol that any software that considers b_0 to be the true genesis block is fraudulent.

In this context, rent extraction may not provide security. When account ownership is anonymous, Carol may not be able to conclude whether Alice extracts any rents from the system, so a threat to abandon the system might not affect Alice. Rather, Carol may

[4]It is generally agreed upon that there is no secure algorithm to establish consensus when a sufficiently large fraction of the voting stake belongs to an agent who has equivocated. For instance, in a white paper for the Casper TFG proof-of-stake algorithm, Vitalik Buterin instead proposes that these instances of equivocation be resolved by market forces or social consensus instead.

rely on social trust between herself and Alice that is external to the system, such as a mutually beneficial business or personal relationship that may be broken off if Alice lies.

3.3 Proof-of-Work blockchain

Proof-of-work blockchains allocate voting power in proportion to the computational resources spent by agents. A vote in this context is just a real number v corresponding to the aggregate amount of computational resources spent by agents on the corresponding block. The chain updating rule in PoW systems is extremely simple: fix some $\alpha > 0$, and for collection of blocks B, let the set of valid chains in B be $\tilde{C}_v = \{C \subset B : v(b) \geq \alpha \ \forall \ b \in C\}$. That is, the set of valid chains includes all chains in which every block has received at least α votes. Then $g(C, B) = C_{max}$, where C_{max} is the chain with the greatest aggregate resource expenditure given by

$$C_{max} = \underset{C' \subset \tilde{C}_v}{\arg\max}\, v(C') \equiv \underset{C' \subset \tilde{C}_v}{\arg\max} \sum_{b \in C'} v(b).$$

No matter what the initial chain C, the output is always the same given a block chain B. Hence, agents require no initial information to come to a consensus. This rule is the analogue of the "longest chain rule" used by most PoW blockchains. In practice, the consensus state in a PoW blockchain is given by the chain with the greatest aggregate difficulty of PoW problems solved, and adding an additional block requires a fixed amount of work (on average).[5]

Suppose that at $t = 1$, b_1 received α votes (combined) from Alice and Bob. The consensus state at the end of $t = 1$ is then ω_1. If Alice wishes Carol to believe the current state is ω_2 at the end of $t = 2$, she simply needs to pay a cost $\alpha + \epsilon$ ($\epsilon > 0$) to vote on b_2. Once she does so, ω_2 will become the consensus state, and both Carol *and* Bob will act according to that state rather than s_1. The payment to Bob has effectively been reversed. Attacks such as this one are why the computer science literature has extensively studied "51% attacks," in which one agent acquires a majority of the available computing power.

Whereas Alice faced ex-post costs of deceiving Carol in the private blockchain and PoS models, in the PoW model the cost of deceiving Carol is incurred ex-ante. The only thing preventing Alice from reversing her payment to Bob (in everyone's eyes) is the cost of computing power required to override the initial payment.[6]

[5] One element we omit from our analysis is the endogenous adjustment of the difficulty of PoW problems. Typically, the difficulty of the problems to be solved adjusts to target a fixed rate at which blocks should be added to the chain. If blocks are added at a higher rate than desired, the difficulty adjusts upwards, and the opposite occurs when blocks are added at a lower rate than desired.

[6] For simplicity, here we assume that Bob will accept the payment after it is included in a single block. In reality, vendors usually deliver goods only after several blocks have followed the initial block including the payment, so Alice would have to overwrite multiple blocks rather than just one.

4 Model

In this section, we present the model. We summarize some of the notation related to blockchains introduced in Sections 2 and 3 and then outline the primitives of the game played among players as well as the way in which players interact with the blockchain.

4.1 State space model

The fundamental purpose of a blockchain in our model is to keep track of a state $\omega \in \Omega$, which consists of allocations of N_T different types of digital tokens in a countable number $\mathbb{N}$ of anonymously owned accounts. Messages are of the form $m = (n, n', s)$, which indicates a transfer of a vector $s \in \mathbb{R}^{N_T}$ of tokens from account n to account n'. If $n = 0$, the message is instead interpreted as seignorage of s tokens distributed to account n'. Messages are used to update the state in the obvious way.

4.2 Blockchain definitions

Blocks: A *genesis block* is the beginning of a blockchain: it is an object (data structure) containing a single state ω. The state contained in a genesis block b is sometimes written as $\omega(b)$. An *ordinary block* is an object containing (1) a set of messages $\tilde{m} \subset M$, (2) a *pointer* to another block, and (3) a *vote*. The set of messages, pointer, and vote corresponding to a block b are denoted by $\tilde{m}(b), p(b)$, and $v(b)$, respectively.

A *blockchain* C is an ordered set of blocks. The ordering on a blockchain is induced by block pointers. The state implied by a blockchain $C = \{b_0, b_1, \ldots, b_K\}$ can be found by starting with the genesis state $\omega(b_0)$ and updating that state with the messages included in each successive block. We will denote the state associated with a blockchain C by $\omega(C)$.

Voting: Votes may be generated either by digital assets internal to the system (proof-of-stake or permissioned) or physical resources external to the system (proof-of-work).[7] In the case of proof-of-work, a vote is simply a real number $v \in \mathbb{R}_+$ indicating the amount of computational resources contributed to a particular block. In the proof-of-stake and private blockchain cases, a vote is a subset $v \subset \mathbb{N}$ indicating the identities of accounts that voted on a particular block.

Blocks are formed by voting messages. As we explained in Section 3, voting messages can be sent from any account n in favor of including a collection of token transfer messages $\tilde{m}$ in a new block at the end of chain C. Each voting message also includes a vote v corresponding to tokens held in the account (PoS or private) or the computational cost incurred by the account's owner (PoW). Voting messages are thus of the form $m_v = (n, \tilde{m}, C, v)$.

[7] Our analysis generalizes to the case in which the voting protocol is a hybrid of proof-of-work and proof-of-stake, but we present the concepts separately here for ease of exposition.

When a collection of voting messages share a common message collection $\tilde{m}$ and chain C, a new block is formed containing messages $\tilde{m}$. Additionally, a collection of *voting reward messages* $\tilde{m}'$ may be added to the block, so the block messages are $\tilde{m}(b) = \tilde{m} \cup \tilde{m}'$. These reward messages distribute seignorage to accounts that voted on the new block in some way. The block vote $v(b)$ is simply an aggregation of the votes included in voting messages. The block pointer $p(b)$ points to the terminal block of chain C.

Consensus algorithm: Record-keeping with a blockchain requires a consensus protocol $g(C, B)$ to update the state. We will assume that the rule g depends only on block pointers and votes rather than on the messages contained in each block.

It is also necessary to specify how agents interpret the blockchain when they do not start with knowledge of an initial state. In practice, this issue is important because this is precisely the situation faced by a new user of a blockchain (as with Carol in Section 3). The process of onboarding a new user is usually termed "bootstrapping." In our model, the state update rule g induces a *bootstrapping protocol* $\hat{g}$ that takes a block collection B and outputs a collection of chains in B, $\hat{g}(B) = \{C'_1, \ldots, C'_K\}$ with $C'_k \subset B$ for all k. The bootstrapping protocol is derived via

$$\hat{g}(B) = \{C' \subset B : g(C, B) = C' \text{ for some } C \subset B\}.$$

4.3 Primitives

Time is discrete and infinite, $t = 0, 1, 2, \ldots$. There are N players who may be present or absent at each t. The set of present players at t is denoted by $P_t \subset N$. We assume that the probability of a player's presence depends only on whether the player was present in the previous period. When players become absent, they lose all recollection of previous play. It is as if players are replaced whenever they become absent, so players who become present are akin to new users.

In this model, there are two types of activities. There are *fundamental activities*, which correspond to a blockchain-based record-keeping mechanism in which players anonymously trade a single numeraire good. These activities consist of consumption and production of goods as well as any proof-of-work expenditures that are incurred in record-keeping (if the blockchain uses PoW).[8] There are also *social activities*, which correspond to bilateral relationships among players. These activities are completely external to the mechanism implemented by the blockchain record-keeping system, but, as we show, they may be useful in building social trust that permits new users of the blockchain to reliably acquire the correct state.

Fundamental activities: Player n has a fundamental type $\theta_{n,t} \sim F_n(\theta)$, where $\theta_{n,t} \in \Theta_n$. Player n's consumption and production at t are denoted by c_{nt} and y_{nt}, respectively. The period utility function is $u(c, y, \theta)$. Players' intertemporal discount

[8] If the blockchain uses some other consensus algorithm, players will not have to engage in any *physical* action in order to keep records. Hence, record-keeping does not fundamentally affect utilities in other settings.

factor is $\delta \in (0,1)$. We set up fundamental preferences in this way because it allows us to work within a simple framework in which players may agree to do favors for others in exchange for tokens.

Players also may incur physical computational expenditures to vote in proof-of-work systems. In order to generate v votes, players must pay a linear cost κv, where κ represents the rental cost of a unit of computational power.[9]

Players' fundamental preference profiles are thus characterized by

$$U^F_{n,t} = \mathbb{E}_t\left[\sum_{s=0}^{\infty} \delta^s \left(u(c_{n,t+s}, y_{n,t}, \theta_{n,t+s}) - \kappa v_{n,t+s}\right)\right].$$

Social activities: Players are connected by a social network $G = (N, E)$, where $E \subset N \times N$ is a set of edges. If $(n, n') \in E$, then players n and n' are said to be connected. Connected players may send private messages to each other and may also derive mutual benefits from their relationship. The per-period bilateral utility obtained by players connected along edge $(n, n') \in E$ (when both are present) is $z_{n,n'} = z_{n',n}$, but these relationships may break down when players become absent. The overall (expected) social utility derived by player n going forward from period t is

$$U^S_{n,t} = \mathbb{E}_t\left[\sum_{n'} \mathbf{1}\{n' \in P_{t+s}\} z_{n,n'}\right].$$

This social network will be important in our analysis because it represents an external source of trust between players that may be useful in attaining consensus. This type of social trust is precisely what would have dissuaded Alice from lying to Carol in our PoS example. In reality, this reduced-form bilateral utility could reflect the value of a business, personal, or political relationship, but it could also stand in for a capacity for neighbors to punish each other (legally or otherwise). Importantly, though, the social interactions in G are completely unrelated to trade in goods. They are important only to the extent that they enable new users to trust information about the blockchain state given to them by their social connections.[10]

Players' overall preference profiles are then

$$U_{n,t} = U^F_{n,t} + U^S_{n,t}.$$

The per-period utility obtained by a player is the sum of (1) the fundamental utilities derived from rendering and receiving services, (2) the bilateral utility obtained through trust relationships with present neighbors in G, and (3) computational expenditures.

We allow for coordinated deviations, at least in the communication of social messages in G, by positing the existence of coalitions $g \subset N$. Each possible coalition g is a connected

[9] The results generalize to situations in which there are additional fixed costs of computational power.

[10] We model external trust as coming from bilateral relationships for simplicity, but our results would extend to a more general setting in which trust comes from the surplus generated by larger coalitions of players.

subset of the graph G. The preferences of a coalition g are given by

$$U_{g,t} = \sum_{n \in g} U_{n,t}.$$

In this context, it is important to allow for some coordinated deviations in reporting the state. If such deviations were not possible, record-keeping by two centralized institutions would be completely secure– if one of the two institutions were to deviate, all agents would receive conflicting reports and thus would immediately detect the deviation.

4.4 Communication and interaction

In this section, we present a model of how players communicate and interact using a blockchain. We first discuss the initial setup and the way in which players keep track of the state. We then outline the user side of the model, which involves the transfer of tokens among users and actions taken in response to those transfers, and finally discuss the record-keeping side, which involves the creation of blocks.

Setup and internal state: At the beginning of $t = 0$, a set of players $P_0 \subset N$ are present. There is a genesis block b_0 (commonly known to players in P_0 but not others) corresponding to state $\omega(b_0)$. As before, a state $\omega \in \Omega$ consists of an assignment of tokens to accounts. There are initially $|P_0|$ numbered accounts, with one account being owned by each player $n \in P_0$. Blocks will periodically be published as play proceeds. Once a block is published, it becomes publicly viewable to all present players in all future periods.

Players must individually keep track of the state in this setting. Each player regards a chain of blocks, called that player's *internal chain*, as a summary of the current state. The state summarized by a player's internal chain is called the *internal state*. When all present players' internal chains agree on a terminal state, it is said to be the *consensus state*. When new blocks are produced, present players update their internal states using the consensus algorithm g.

Newly arriving players use the bootstrapping protocol $\hat{g}$ to narrow down the list of possible consensus states, but they may require additional information. In that case, the new player n asks her neighbors in G to provide her with the chain C corresponding to the current consensus state. If all neighbors in G report the same chain, she accepts that as her internal chain. Otherwise, she immediately leaves (becomes absent). Importantly, a player's neighbors in G may lie about the consensus state. If player n's neighbors lie at time t, n discovers the lie at $t + 1$ and leaves.[11] Hence, lying to new users may provide the short-term benefit of distorting their view of the state but results in two types of long-term losses. First, lying players lose the social trust stemming from the mutually beneficial relationship with the new user (i.e., the social connection in$G$). Second, lying players lose any rents extracted from the new user through the mechanism. These two

[11] As will become evident, in our context the assumption that players always discover lies one period later is conservative. Our results would carry through if players discovered lies only with some probability.

forces reflect the tradeoff faced by Alice in our private blockchain and PoS examples when deciding whether to lie to Carol.

User side: The assignment of tokens to accounts is public knowledge, but account ownership is private information.[12] In any period, players may create new accounts with no token holdings.

In their capacity as users, players may send messages transferring tokens from an account n to an account n'. All such messages are sent with reference to a chain C summarizing the state in which the tokens are to be transferred. In principle, players should send messages only in reference to their internal states, but they may deviate, as in a double-spend. Only the player who owns account n may send messages in which n sends tokens to other accounts, and no player may send tokens from an account that exceed its holdings.

Players may enter *agreements* to produce goods for others in exchange for tokens. Agreements are anonymous and may, in principle, be made between any two players An agreement stipulates that the producer n will send $y_{n,n'}$ goods to the consumer n' if a particular message $m_{n,n'}$ (representing a payment) is included in the blockchain at the end of a chain C. In this model, agreements are needed because token transfer messages must be sent before it is known whether they will ultimately be included in the blockchain.

Importantly, players' internal chains will determine the actions they take. They will agree to produce only if they believe a payment has been included in the consensus state, so their agreements will require that messages be included in their internal chains. Just as in practice, vendors will wait until they believe payments are finalized before transferring goods to buyers.[13]

Record-keeping side: There are two types of messages players may send in their capacity as record-keepers. First, players may send voting messages. A player may cast vote v only if he owns account n and (in PoW systems) completes the requisite amount of work. Further, while players may choose to omit some observed messages from their reports $\tilde{m}$, they may not report seeing messages that were never sent.

Second, players may create new genesis blocks with arbitrary numbers of accounts and token holdings. When they do so, they automatically take possession of all accounts in the new genesis block. We allow for this possibility so that players may execute deviations of the type presented in Section 3 in order to fool newly arriving users.

Timeline: There are two subperiods $\tau = 0, 1$ in each period t. Play in these subperiods proceeds as follows:

1. At the beginning of $\tau = 0$, present players are informed as to whether any of their

[12]The assumption that token holdings are public is not required, but it makes some proofs more transparent. Encryption schemes used in reality, in fact, sometimes prevent accounts' token holdings from becoming public knowledge.

[13]Actual transactions using cryptocurrencies often occur as follows: the buyer will send a message transferring tokens to the seller, the seller will wait until the message is entered into the blockchain, and then the seller transfers the goods to the buyer.

neighbors in G will arrive in the current period. At the end of $\tau = 0$, new players arrive and communicate with present players to obtain information about the state.

2. At $\tau = 1$, there are three phases of play:
   - **Phase 1:** Players enter agreements.
   - **Phase 2:** Players send token transfer messages.
   - **Phase 3:** Players cast votes and incur computational costs if necessary.

   Once the third phase ends, players update their internal states and carry out agreements. Bilateral utilities from relationships in G are also realized.

4.5 Blockchain mechanism

In this section, we outline the blockchain mechanism that dictates token transfers among players, voting strategies, and the actions players take. We then characterize a value function derived from the actions and state transitions in the mechanism.

Player n's private information in a state ω consists of a type θ_n and a set A_n of accounts owned. These pieces of private information can be summarized as a type $\tilde{\theta}_n = (\theta_n, A_n)$. A player's internal chain $C_{n,t}$ is an additional piece of private information that determines which state that player considers to be the consensus state.

The mechanism we consider is effectively a communication protocol. It takes a collection of reports $(\tilde{\theta}_n, C_n)$ and recommends a strategy to be played at $\tau = 1$: players are informed as to which agreements they should enter, which messages they should send contingent on the agreements made, and which votes they should cast contingent on the messages observed.

Communication is anonymous, so players may submit any collection of reports they would like. Hence, a player n who knows the true consensus chain C^* but has tricked n' into believing the consensus chain is C' may truthfully communicate with those who know the consensus state by reporting $(\tilde{\theta}_n, C^*)$. Simultaneously, n may submit a different set of reports to C' in order to trick n' into taking some action. Crucially, communication occurs only among players who report the same internal chain, so the recommendations made to players who know the true consensus state do not depend on those made to players who have been incorrectly informed of the consensus state. That is, players simply ignore those who disagree with them about the state.

We say the mechanism is implemented when it is incentive-compatible for players to follow its recommendations and take no other actions (in any subgame). When the mechanism is implemented, it induces a rule

$$T^*(\tilde{\theta}, \omega) = (c, y, \omega'),$$

where $\tilde{\theta}, c$, and y represent the type, consumption, and production vectors. We assume that this rule is anonymous, meaning it is invariant to permutations of identities. From

this rule, it is possible to derive a value function

$$V_n(\tilde{\theta}_n, \omega) = \mathbb{E}\left[u_n(\tilde{\theta}, \omega) - \kappa v(\tilde{\theta}, \omega) + \delta V_n(\tilde{\theta}'_n, \omega') \middle| \theta_n, \omega\right],$$

where $u_n(\tilde{\theta}, \omega) \equiv u(c_n(\tilde{\theta}, \omega), y_n(\tilde{\theta}, \omega), \theta_n)$ is the flow utility obtained by n from consumption and production.

There is also a value function deriving from social connections,

$$V_n^S(\tilde{\theta}_n) = \sum_{n'=1}^{N} V_{n,n'}^S(\tilde{\theta}_n) = \sum_{n'=1}^{N} \mathbb{E}\left[\mathbf{1}\{n' \in P\} z_{n,n'} + \delta V_{n,n'}^S(\tilde{\theta}'_n) \middle| \tilde{\theta}_n\right].$$

Note that since the probability of a player's presence in the current period depends only on whether that player was present in the previous period, we can write $V_{n,n'}^S(\tilde{\theta}_n) = V_{n,n'}^S(P_{n'})$, where $P_{n'}$ is an indicator variable equal to one when n' was present in the previous period.

4.6 The Blockchain Trilemma

Our main result is the Blockchain Trilemma. It states that no sequential digital record-keeping system can simultaneously satisfy three properties: self-sufficiency, no rent extraction, and no waste of resources. In our context, self-sufficiency means that the system could operate without the mutually beneficial relationships in the underlying social network G, no rent extraction means that new users extract the full surplus they generate, and no waste of resources simply means proof-of-work is unnecessary.

The first important piece of our result is a "Mimicking Lemma" that shows that at a cost, a single player alone can generate a blockchain that mimics the output produced on the consensus chain in equilibrium. With this result in hand, we will then provide a sketch of the Trilemma result.

Mimicking Lemma. *Let $C = \{b_0, \ldots, b_K\}$ be a blockchain created by N players, and let c^* be the cumulative proof-of-work cost paid by those players in creating C. A single player n can create a blockchain identical to C at cost c^*.*

Proof. First, player n creates a genesis block b'_0 with accounts and holdings identical to those in b_0, so $\omega(b_0) = \omega(b'_0)$, except n owns all the accounts corresponding to b'_0.

After creating the new genesis, player n should successively create blocks b'_k for $1 \leq k \leq K$ in the following manner:

1. Create all accounts that appeared for the first time in block b_k.
2. Send messages $\tilde{m}_k$ identical to those that were sent in block b_k.
3. Cast votes v_k identical to those cast in b_k, creating a block with messages $\tilde{m}_k$ and votes v_k but with a pointer to block b'_{k-1}.

All three steps are feasible for n. By an inductive argument, the state implied by b'_{k-1} is identical to that implied by b_{k-1}, so n has the tokens required to send the messages in step 2 (since it was feasible for the account holders on the original chain to send those messages). Then in step 3, n again has the tokens required to cast those votes if necessary. If the system is instead proof-of-work, n pays a cost $\kappa v(b'_k) = \kappa v(b_k)$, so the cumulative cost of casting votes is exactly the same as that paid by the original N players who created C. □

The idea behind the proof is simple. To replicate the blockchain produced by the other N players, n needs to find a way to send identical messages and cast identical votes. In this model, players are prevented from sending arbitrary messages and votes because they do not own certain tokens. However, if n creates a genesis block b'_0 and a set of accounts identical to those in the original genesis b_0, he now owns all the tokens on the alternate blockchain needed to replicate the messages and votes sent in the first block. If votes require proof-of-work, player n can generate the same set of votes simply by paying the same cost that the original N players paid. The the state $\omega(b'_1)$ is identical to $\omega(b_1)$, and player n can follow this procedure in each individual block in order to generate an identical blockchain.

While the mimicking lemma relies on players' ability to create an entirely new genesis block that will be believed by others, in reality it would not always be necessary for an attacker to be able to do that. All that is needed is for an attacker to gain the voting power necessary to create a chain of blocks identical to the true consensus chain. This may be an entirely new chain, as in the proof, or it may be a fork of the existing consensus chain.

In proof-of-work systems, an attacker always has the power to create an identical chain– he merely needs to acquire enough computing power to compete with the rest of the network. In proof-of-stake systems, an attacker would need to acquire the tokens constituting a supermajority of votes in some previous block (as in a long-range attack). If the current consensus block is b_t, it is plausible that the accounts that held a supermajority of tokens in a block long ago, b_{t-s}, may be nearly empty. An attacker could attempt to purchase the private keys from the owners of those accounts and gain the ability to create a mimicking fork starting from b_{t-s}, as in the lemma. In a monopolistic private blockchain system, however, a fork would be concrete proof of equivocation by the monopolist, so an attacker would indeed need to present victims with an entirely different blockchain.

The Blockchain Trilemma follows as a consequence of the Mimicking Lemma. First, we define our concepts of rents, social trust, and resource costs, and then we provide a statement of the result and a proof sketch.

The *rent* extracted by a coalition g from a player $n' \notin g$ in state ω when players in g have types $\tilde{\theta}_n$ is defined as

$$r^*(g, n', \omega, \tilde{\theta}) = \sum_{n \in g} \Big(\mathbb{E}\left[V_n(\tilde{\theta}_n, \omega) \middle| n' \in P\right] - \mathbb{E}\left[V_n(\tilde{\theta}_n, \omega) \middle| n' \notin P\right] \Big).$$

It is simply the value obtained by g when n' is present minus the value obtained by g when n' is absent. When the rent extracted by g is positive, n' does not capture the full surplus generated by her presence.[14]

The *social trust* of a player n' in a coalition g is the value of mutually beneficial relationships for g generated by the presence of n':

$$s^*(g, n', \theta) = \sum_{n \in g} \left(\mathbb{E}\left[V^S_{n,n'} \middle| n' \in P\right] - \mathbb{E}\left[V^S_{n,n'} \middle| n' \notin P\right] \right).$$

The social value is just the expected discounted value of the flow benefits $z_{n,n'}$ generated when n' is present.

The *waste of resources* c^* corresponding to a blockchain C is just the sum of the proof-of-work costs spent on that chain,

$$c^*(C) = \sum_{b \in C} \kappa v(b).$$

While rents and social trust are defined at the level of relationships between players in a mechanism, the waste of resources is defined at the level of a ledger.

We now have the definitions required to present our main result.

Blockchain Trilemma. *No record-keeping mechanism is implementable without rents, social trust, or a waste of resources.*

Proof. Consider any record-keeping mechanism, and let C^* be the publicly viewable blockchain when a player n' becomes present. We want to show that if the set of social connections of n' is the coalition g, then there exists $u^A > 0$ such that

$$V^A \leq r^* + s^* + c^*,$$

where r^* and s^* are the rents and social trust benefits extracted by g from n', respectively, and c^* is the cumulative proof-of-work cost involved in creating blockchain C^*.

By the Mimicking Lemma, when informed at time (t, τ) that n' will arrive, coalition g can pay a cost $c^*(C^*)$ to create a blockchain C^A identical to C^* in which one player $n_0 \in g$ owns all the accounts and tokens. When n' arrives, she sees two identical blockchains, C^* and C^A, so she needs additional social information from g to determine which should be her internal state. The members of g can coordinate to tell n' that the consensus chain is C^A.

Once n' has been convinced that the consensus chain is C^A, player n_0 can simulate

[14] While rents are not usually defined in this way, it is clear that as the number of players becomes large (holding the size of g fixed), this surplus is in fact equal to the rent extracted by the coalition g from player n'. Even if the presence of n' creates value for the full set of players, it cannot create much value for players in g. Any value extracted by players in g from n' must be a result of some transfer of the surplus created by n' to g.

the equilibrium behavior of $\tilde{N}$ players in state ω for any types $\tilde{\theta}^A = (\tilde{\theta}^A_1, \ldots, \tilde{\theta}^A_{\tilde{N}})$ they might have. At $\tau = 1$, n_0 simply submits reports $(\tilde{\theta}^A_k, C^A)$ while n' submits $(\tilde{\theta}_{n'}, C^A)$. In particular, n_0 can choose $\tilde{\theta}^A$ to maximize the goods that n' will produce:

$$\tilde{\theta}^A = \arg\max_{\tilde{\theta}} \mathbb{E}\left[y_{n'}\big((\tilde{\theta}, \tilde{\theta}_{n'}), \omega\big)\right].$$

Player n_0 then acts exactly as players of type $\tilde{\theta}^A_k$ would have acted at $\tau = 1$, generating a set of messages and votes that result in n' sending him $c^A = y_{n'}\big((\tilde{\theta}, \tilde{\theta}_{n'}), \omega\big)$ goods.

The coalition g may act towards all other players in the exact same way it would have if n' had been absent. Player n' leaves following the coordinated lie by coalition g, so both the value derived from the mechanism and the social value enjoyed by g are as if n' had never been present. However, instead of obtaining the flow value $u(c^*, y^*, \tilde{\theta}_{n_0})$ that would have obtained if n' had been absent, player n_0 gets utility $u(c^* + c^A, y^*, \tilde{\theta}_{n_0})$. The value of this attack to the coalition g is therefore

$$\begin{aligned} V^A &= u(c^* + c^A, y^*, \tilde{\theta}_{n_0}) - u(c^*, y^*, \tilde{\theta}_{n_0}) + \sum_{n \in g} \mathbb{E}\left[V_n(\tilde{\theta}_n, \omega) + V^S_n(\tilde{\theta}_n) \middle| n' \notin P\right] - c^* \\ &\equiv u^A + \sum_{n \in g} \mathbb{E}\left[V_n(\tilde{\theta}_n, \omega) + V^S_n(\tilde{\theta}_n) \middle| n' \notin P\right] - c^*. \end{aligned}$$

Here the value u^A is effectively the value of goods stolen by g from n'.

By contrast, if g had not deviated, players would have obtained the value

$$V^* = \sum_{n \in g} \mathbb{E}\left[V_n(\tilde{\theta}_n, \omega) + V^S_n(\tilde{\theta}_n) \middle| n \in P\right].$$

Coalition g pays two costs for deceiving n'. First, it pays the up-front computational cost c^* to mimic the true blockchain. Second, it loses the value it would have derived from the presence of n', both through the mechanism and through mutually beneficial relationships between n' and players in g.

The relevant incentive compatibility condition $V^A \leq V^*$ then clearly reduces to

$$u^A \leq r^* + s^* + c^*,$$

as desired.

□

The Trilemma can be understood through a simple example. Say a new user n' has social contacts in coalition g. The players in g can create a new blockchain that mimics the behavior of the consensus chain, so n' will have to ask players in g which is the true consensus chain. Players in g can lie and then use their anonymous accounts on the fraudulent chain they created to convince n' to deliver some goods to them.

The coalition g faces both ex-ante and ex-post costs for engaging in this deviation.

Ex-ante, the members of the coalition must expend computational resources to create the mimicking blockchain if a proof-of-work consensus algorithm is used. Ex-post, player n' will leave the system after detecting the deviation by g, so players in g suffer both because n' may have provided them with some rents by joining the network and because any social trust relationships with n' break down. Hence, the sum of these three costs must be sufficient to dissuade players in g from initially stealing goods from n'.

5 Discussion

In this section, we present an informal discussion of how our benchmark model could be extended to speak to other important blockchain-related questions. We outline the potential benefits of decentralized record-keeping and comment on the viability of blockchains for which enforcement of ownership rights is necessary.

5.1 The benefits of decentralization

While our model is well-suited to highlighting the tradeoffs between the different ways to incentivize honest record-keeping, it is quite general and thus does not make sharp predictions on other issues. In particular, the benchmark model is silent on the specific mechanisms that may make decentralization worthwhile.

One of the original questions that inspired the design of decentralized record-keeping systems was whether the rents extracted by traditional centralized record-keepers could somehow be eliminated. In the view of many blockchain proponents, decentralized consensus algorithms are useful because they prevent one entity from being the ultimate arbiter of which transactions were included in the ledger. The intuition underlying this view is that competition among multiple record-keepers should prevent any one of them from extorting users by demanding high transaction fees. Our model indeed implies that social trust or PoW resource costs can eliminate the *need* for rents, but it does not provide an explicit channel by which these rents arise in centralized systems and are eliminated in decentralized systems.

A simple reputation-based mechanism succeeds in generating rents in the centralized case that dissipate in the decentralized case. Suppose that players exchange one type of token for goods, and that a transaction fee compensating record-keepers with tokens is included in each transaction message, so that record-keepers are paid only if the transaction is included in the ledger. In addition to their fundamental types, players also observe the entire history of transactions (and thus fees) included in the ledger. A history in which no transaction with a fee less than $\overline{\tau}$ has been included is denoted by S (for *strong* reputation), and one in which some transaction with a fee less than $\overline{\tau}$ has been included is denoted by W (for *weak*). Suppose that players will submit transactions with fees less than τ only if they believe these transactions will be added to the ledger, and that players expect all transactions to be accepted after a history W.

A single centralized record-keeper may find it beneficial to maintain its reputation by always rejecting transactions with low fees. By doing so, it potentially passes up some transactions with low fees, knowing that accepting a low-fee transaction even once will result in a transition to W and low fees for all future transactions. That is, the fact that the centralized record-keeper extracts all fees provides a strong incentive to maintain the status quo. On the other hand, with a decentralized group of record-keepers, such incentives are minimal. In particular, under PoW systems, record-keepers should break even. With a PoS system, the transaction fees distributed to each individual record-keeper would be small. Hence, decentralized record-keepers would not pass up an opportunity to accept and be compensated for transactions with low fees even if doing so resulted in lower fees going forward.

This simple example illustrates a broader point. There is a distinction between *getting* the current state of the blockchain and *updating* the state. Decentralization has important implications for both functions. In general, the consensus algorithm provides a new user with some information as to the new state, and if that information is incomplete, the user will have to obtain more precise information from social connections. The PoW consensus algorithm provides new users with all the information they need, whereas PoS does not always do so. However, the two decentralized consensus algorithms have similar implications for how the state is updated. Neither gives high profits to any individual agent, and thus both prevent record-keepers from internalizing the destruction of "reputation" if they accept transactions with low fees. Hence, decentralization both determines how new users acquire the state and, under some circumstances, can prevent users from being extorted by high fees. For a more detailed explanation of the competitive implications of decentralized record-keeping, see Huberman, Leshno, and Moallemi (2019).

5.2 Enforcement

In our model, the tokens traded on the blockchain are useful only to the extent that others accept them in exchange. They do not represent claims to physical or financial assets. In reality, there have been several proposals for blockchains on which physical assets such as houses or automobiles can be traded, or on which companies could issue securities that constitute promises to deliver certain cash flows to the owner. The difference between these types of blockchains and the ones we consider is that when tokens represent legally binding claims, there must be some entity willing to enforce those claims. There is an important distinction between *ownership* and *possession*. Blockchains can record transfers of ownership of a digital asset, but an enforcing entity must ensure that the owner of a digital claim can come to be in possession of the financial or physical asset underlying that claim. In most situations, the enforcer would be the relevant legal authority.

Power ultimately lies with the enforcer. To see this point clearly, consider a simple example of a blockchain on which each token represents ownership of a particular house.

Suppose that the blockchain forks into two chains, C_1 and C_2. On C_1, Alice owns house H in one of her accounts, whereas on C_2, Bob owns H. Then who is actually able to live in the house? The enforcer is endowed with the power to decide that one of the two has the legal right to live in the house and may evict the other. Then the enforcer effectively decides which branch of the fork should be considered the true state. There may be rules governing how the enforcer should act, but these rules must be enforced by a legal authority as well. Then, to ensure that the enforcer does not act with impunity, there must be some way to punish the enforcer through the blockchain mechanism (e.g., by decreasing the benefits it derives from the mechanism) or externally (e.g., through elections).

These considerations could be made explicit by extending our model to incorporate physical assets with *digital identities* in the form of tokens traded on the blockchain. There would be an additional player, called an enforcer, who would be tasked with allocating the physical assets. Without the enforcer's assistance, players would have no way of ensuring that their assets are not stolen by others. The mechanism would dictate how the enforcer should allocate assets based on the state of the blockchain. Importantly, in case of a fork, the enforcer would have to make a decision as to which branch of the fork to honor. Of course, the enforcer would need the correct incentives to make the right decision. The Trilemma indicates that there are two options: the enforcer may directly extract rents from the mechanism in some way, or there may be external arrangements that permit others to punish the enforcer (what we call "social trust" in our benchmark model). Realistically, a legal authority may enjoy private benefits from its position, so any mechanism that allows for the removal of that authority would constitute an appropriate punishment option. As long as the enforcer has the correct incentives to follow the rules, all other players may behave as in the benchmark model.

In this modification of our model, the enforcer has full power to decide the current state. That is, the enforcer may ultimately decide the allocation of physical assets in each period. If the enforcer behaves, it is only because it has the correct incentives to do so. This observation has several important implications. First, if a new user needs to acquire information about the current state from social connections in G, it is enough for that new user to ask the enforcer only. The enforcer decides which physical assets will be allocated to the new user, so any reports that conflict with the enforcer's are irrelevant for that user. Second, it may be efficient to bundle the responsibility of record-keeping (i.e., updating the ledger) with enforcement. The enforcer must enjoy some rents or social trust, so it may be that its incentives are sufficient to ensure it keeps records honestly as well. This type of bundling could be beneficial, for example, if allowing another record-keeper to extract rents would create inefficiencies. Nevertheless, there may still be some benefits of decentralized record-keeping even when there is a need for centralized enforcement. As discussed in the previous subsection, decentralization may be an effective tool to lower the transaction fees paid by participants in the mechanism.

6 Conclusion

The fundamental question that inspired the invention of public blockchains is whether consensus can be achieved without trust in a single entity. The blockchain trilemma answers this question: it is possible to replace trust in a single entity (generated by rents) either by imposing resource costs or by relying on external sources of trust among individuals. Proof-of-work blockchains take the former approach. It is possible for a new user who knows nothing other than the blockchain protocol to read the ledger, deduce the current state, and use the blockchain-based mechanism with the confidence that records will be kept honestly. Proof-of-stake blockchains take a less radical approach, achieving decentralization by changing the *structure* of trust required for the system to operate correctly. A new user must rely on external trust with an existing user in order to acquire the state, but there is no need for all new users to share a common trusted source. Instead, users may choose who they trust to provide that basic information. The trilemma implies that these are, in fact, essentially the only possibilities.

We leave to future research the question of which types of ledgers should be centralized and which should be distributed. The benefits of decentralization come from the elimination of rents, potentially above and beyond the minimum level of rents required to ensure honest reporting. Our model is too general to make specific predictions relating the record-keeping protocol to rent extraction, but it seems natural that the free entry of record-keepers permitted by decentralized blockchains should prevent the type of collusion required to extract large fees from users. We also clarify the distinction between ownership, which can be recorded on a blockchain, and possession, which must be enforced by a trusted entity. When there is a need for a trusted entity regardless of whether record-keeping is centralized or not, it may be more efficient to bundle record-keeping with enforcement.

References

[1] Bruno Biais, Christophe Bivière, Matthieu Bouvard, and Catherine Casamatta. The blockchain folk theorem. *The Review of Financial Studies*, 32(5):1662–1715, May 2019.

[2] Eric Budish. The economic limits of bitcoin and the blockchain. Working paper, 2018.

[3] Jonathan Chiu and Thorsten Koeppl. The economics of cryptocurrencies– bitcoin and beyond. Working paper, 2017.

[4] William Lin Cong and Zhiguo He. Blockchain disruption and smart contracts. *The Review of Financial Studies*, 32(5):1754–1797, May 2019.

[5] William Lin Cong, Ye Li, and Neng Wang. Tokenomics: Dynamic adoption and valuation. Working Paper, 2019.

[6] David Easley, Maurenn O'Hara, and Soumya Basu. From mining to markets: The evolution of bitcoin transaction fees. *Journal of Financial Economics*, 134(1):91–109, October 2019.

[7] Joshua Gans, June Ma, and Rabee Tourky. Market structure in bitcoin mining. NBER Working Paper, 2018.

[8] Arthur Gervais, Ghassan Kharame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[9] Gur Huberman, Jacob Leshno, and Ciamac Moallemi. An economic analysis of the bitcoin payment system. Working Paper, 2019.

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

[11] Emiliano Pagnotta and Andrea Buraschi. An equilibrium valuation of bitcoin and decentralized network assets. Working paper, 2018.

[12] Linda Schilling and Harald Uhlig. Some simple bitcoin economics. Forthcoming, Journal of Monetary Economics, 2019.

[13] Michael Sockin and Wei Xiong. A model of cryptocurrencies. Working paper, 2018.

[14] Karl Wüst and Arthur Gervais. Do you need a blockchain? *IACR Cryptology ePrint Archive*, 2017.