Geeq

John P. Conley
Vanderbilt University
August 2018

The Geeq Project Technical Paper

Version 1.0

Proof of Honesty: Coalition-Proof Blockchain Validation without Proof of Work or Stake

Proof of Honesty: Coalition-Proof Blockchain Validation without Proof of Work or Stake[1]

John P. Conley[2]

Vanderbilt University

August 2018

Version 1.0

Abstract

Blockchains are distributed, immutable, append only, ledgers designed to make trustless interactions between anonymous agents feasible and safe. The ledgers are maintained by networks of independent nodes who process transactions and come to a consensus view of which are valid and how this affects the ledger state. The integrity of blockchain ledgers therefore depends on the incentives contained in the consensus protocols that are designed to make the validating nodes behave honestly. In this paper, we argue that existing protocols based on Proof of Work, Proof of Stake, Directed Acyclic Graphs, and so on, offer relatively weak security guarantees. In part, this is because existing protocols have their roots in algorithmic game theory instead of economic mechanism design. We propose a new blockchain validation protocol called Proof of Honesty. We show that when Proof of Honesty is combined with a non-cooperative game called the Catastrophic Dissent Mechanism, the resulting consensus protocol implements truthful blockchain validation in coalition-proof equilibrium. Thus, Blockchains using Proof of Honesty offer users Strategically Provable Security and 100% Byzantine Fault Tolerance.

---

1 This technical paper contains proofs that the propitiatory, patent pending, blockchain consensus mechanism used by GeeqChain is 100% BFT and offers SPS. Proof of Honesty, PoH, Strategically Provable Security, SPS, Catastrophic Dissent Mechanism, CDM, GeeqCoin, GeeqChain, and Geeqosystem are all registered trademarks of the Geeq Corporation.

2 j.p.conley@vanderbilt.edu. Proof of Honesty is an element of a more general protocol developed for GeeqChain, a new infrastructure blockchain project in which the author is a participant. Part of this research was done while the author was on sabbatical at Microsoft Research over the 2016-2017 academic year. The author would like to thank Ric Asselstine, Darryl Patterson, Yorke Rhodes, Stephanie So, Serge Sverdlov, Simon Wilkie, and Lun Shin Yuen for useful discussions about blockchain, applications, and ICT. The author takes sole responsibility for the contents of this paper.

1. Introduction

Blockchains are electronic ledgers that group transactions together into "blocks" and append them sequentially to an existing "chain." From a functional standpoint, blockchains are direct descendants of the paper ledgers using double-entry bookkeeping invented in the fifteenth century by the Milanese Franciscan monk, Luca Pacioli. Ledgers have a particularly simple data structure recording only account numbers/owners and current balances. They lack the meta and semantic web data of XML, the keys and relational table structures of SQL type databases, and even the columns and rows of spreadsheets.

Conventional databases are typically kept on a central server under the control of a single party. This party has the power to determine who has access to various parts of the database and also the ability to alter or delete records. This means the data is only as trustworthy as the party in control. In addition, since the data is kept in one place (or at least by a single entity), such databases have a central point of failure. As a result, state actors, courts, criminals, and hackers may be able to censor, alter, or even deny access to such data.

In contrast, blockchain ledgers are updated and stored by a widely distributed set of agents, sometimes called nodes or miners. Transactions are sent by users to these nodes who communicate them to one another through peer-to-peer networks. Each node examines candidate transitions and decides if they are valid. The network of nodes as a whole comes to a consensus agreement and then uses the set of valid transactions to update the current ledger state. In computer science, blockchains are described as transition-state machines that use distributed ledger technology.

Although blockchains dramatically reduce the richness and utility of data, they offer several advantages. The most important are that they do not rely on the good behavior of a single agent to maintain the database, and they do not have a central point of failure that might be attacked by bad actors. Instead, copies of the ledger are kept by many independent nodes. Alterations, deletions, or denial of access to such ledgers would require the coordination and cooperation (or the coercion) of thousands of distributed, anonymous, actors. As a result, users are able to interact and exchange value with each other in a trustless way and without requiring permission for any central authority. Depending upon how a given blockchain is constructed, it may also create a transaction record that is immutable in the sense that it would be computationally impractical to change historical data, as well as allowing users to make transactions while maintaining their anonymity.

All of the advantages that blockchains offer depend on honest transaction verification and block-writing by the network of nodes. Blockchains use a number of different validation protocols to ensure this. Protocols lay out the set of rules that users and miners are supposed to follow and include incentive structures to make sure that they do. This sounds like a problem that is quite familiar to economists. Protocols seem very much like economic mechanisms in that they are designed to implement desirable outcomes by aligning the incentives of the players or agents with the objectives on the mechanism designer.

In this paper we do three things. First, we argue that the incentive structures of the most prominent blockchain protocols are wholly inadequate in that they do not provide strong guarantees of

honest transaction validation by the network of nodes. Second, we suggest that the reason for this is that protocols, which have their roots in computer science and algorithmic game theory, are not quite the same thing as mechanisms, which have their roots in economics and noncooperative game theory. Finally, we propose a new approach to blockchain validation called Proof of Honesty (PoH) and show that when it is combined with a non-cooperative game called the Catastrophic Dissent Mechanism (CDM), the resulting consensus protocol implements truthful blockchain validation in coalition-proof equilibrium. Thus, Proof of Honestly offers users Strategically Provable Security (SPS) and 100% Byzantine Fault Tolerance (BFT).[3]

2. Some Background on Protocols

To fix ideas, this section gives a short outline of how the Bitcoin protocol invented by *S*atoshi Nakamoto[4] (2009) works. A few of the more important technical ideas are discussed in more detail in Appendix 2. Readers familiar with Bitcoin and blockchain protocols should feel free to skip to the next section.

The Nakamoto/Bitcoin Protocol

1. Thousands of anonymous nodes: Bitcoin depends on a network of more than 10,000 nodes, also called miners, to verify and secure its ledger. Nodes are nothing more than computers attached to the internet at some IP address that have been set up to run software that contains the Bitcoin protocol.

2. Nodes join a P2P validations network: Agents can become validators by broadcasting their presence on a peer-to-peer network that connects all the nodes. The advantage of this approach is that there is no central server or definitive record of nodes that might be censored or blocked. Nodes connect directly to other close-by nodes, which in turn, are connected to other nodes further away.

3. Users send transactions to nodes that are propagated to the entire network: Users find the IP address of a Bitcoin node and send it a transaction request. This request is then broadcast to a group of peer nodes, known to the receiving node, and from there it spreads out to the rest of the nodes on the network. This is sometimes called a "gossip network" as transactions, finished blocks, and other communications traffic are passed from node to node until they are known by the whole of the network.

---

3 BFT comes from Lamport, Shostak, and Pease (1982) in which they describe "The Byzantine Generals Problem". To understand the idea, suppose that ten generals of the Byzantine empire are surrounding a city, some of whom have been bribed not to attack. Suppose the participation of at least seven is required in order to conquer the city. Otherwise, the attack fails. Such battle plan is said to be 30% BFT. Most blockchain protocols have BFT of 50% of less. More generally, BFT is a measure of how tolerant a system is to faulty components. Unfortunately, characterizing robustness in this way tends to make protocol designers think of nodes as parts of a system that either work as expected. or fail, instead of as rational agents with preferences who are capable of doing either depending upon the circumstances.

4 *S*atoshi Nakamoto is a pseudonym. The true author or authors of this paper are unknown.

4. Each node builds a block of valid user transactions: Each node chooses a set of 1000-2000 transactions and puts them together in a "block". Each node is required by protocol to verify that the transactions are valid. This means that three things must be true. First, the account mentioned in the transaction request exists in the ledger and has enough bitcoin to its credit to cover the requested transfer (no over-spending). Second, the transaction is digitally signed in a way that makes it certain that the true owner of the account is making the request. Third, there is no double spending, meaning, that if there are several transaction requests for the same account, they collectively do not add up to more than the account contains. (See Appendix 2 for an explanation of public private keys (PPK) and digital signatures).

5. Each node mines its own block with PoW: Each node starts to solve a sort of "guess and check" cryptographic puzzle. This takes a great deal of computational effort and resources. In fact, more than three billion dollars worth of electricity was used to validate the Bitcoin blockchain in 2017. As of June, 2018, the Bitcoin mining network as a whole has a hashrate of 35 exa-hashes per second. That is, the Bitcoin network is able to make 10^{18} guesses at the solution to the current puzzle per second. The difficulty of the puzzle is adjusted every two weeks so that it takes the network 10 minutes on average to find a solution and thereby "mine" the next block.

6. The first node to mine its block broadcasts it to other nodes on the P2P network: The first miner to guess correctly discovers a number called a "nonce". The magical thing about a nonce is that, although it takes a lot of computational effort to find, it is trivial to verify that it is correct. As a result, a miner can prove that he worked (spent computational effort) by including the nonce in the block he just mined. The successful miner then broadcasts his new block over the gossip network to all the other nodes.

7. Only the successful miner gets paid transaction fees and mining rewards: The successful miner gets to collect transactions fees offered by account holders to process their requests, and more importantly, a mining reward. The bitcoin protocol allows the successful miner to create a certain number of new bitcoins (currently 12.5 BTC) and write them into its own account to compensate it for its efforts.

8. All other (unsuccessful) miners abandon their own blocks and commit the one received from the first successful miner to their version of the blockchain: Each node that receives a newly mined block is required by protocol to verify the block is valid in the sense that the signatures on the transactions are correct, there is no over-spending or double spending, and the nonce is correct. If everything checks out, each node is required by protocol to "commit" the block to the end of the existing blockchain (more on this below). Note that this means that only the successful miner gets paid anything for his work on the puzzle and providing verification services to the network. Unsuccessful miners stop working on their current block, choose a new set of transactions, and start to guess at the new nonce.

9. New blocks are appended to the end of the current chain using a Merkle tree: Blocks are chained together using a recursive hashing technique called a Merkle tree. Roughly speaking, the previous block is run through a publicly known hashing fuction. The result is a 256 bit "finger print" of the previous block (if SHA-256 is the function used) called a hash. This hash is added to the new block that is about to be appended to the existing chain. The next block takes

a hash of this block and includes it, and so on. This process links the blocks together cryptographically. In particular, if someone were to change anything, even a single bit in a block earlier in the chain, the hash or fingerprint would be different. This means that the next block would have to contain the new hash. But then this block would have to change, and its hash would be different, and so on up to the last block. (See Appendix 2 for more discussion of hashing and Merkle trees.)

10. The Merkle tree plus PoW makes the Bitcoin blockchain immutable from a practical standpoint: This bit of Merkle magic is what buys Bitcoin its immutability. If anyone wanted to change something buried in the chain, not only would its hash change, but the nonce would be different as well. Thus, to make the "Merkle proof" correct, the changed block would have to be re-mined (which takes lots of work in the form of computational effort), and then the next block would have to be re-mined for the correct nonce, and so on up to the current block. In other words, the further back you wanted to make a change in, or otherwise falsify, a record, the higher the computational cost. It costs on the order of $ 100,000 to mine a block, so to change something 50 blocks back would require $ 5,000,000 of effort. Even this would not be enough since new blocks would have been written in the meantime, and so these would have to be re-mined as well.

11. Miners must append new blocks to the longest correct chain if more than one fork exists: The last important element of the protocol involves what is called a "fork". Suppose that two miners find the nonce at roughly the same time. Then some nodes on the gossip network might receive one of these proposed blocks and commit it to its chain while other nodes might receive the second proposed block first and commit it instead. This means that the chain has forked since there are now two versions of the blockchain with different last blocks. Nodes are required by protocol to build on the longest chain, but if both are equal, they have a choice. It is unlikely that this kind of simultaneous block mining will happen very often, and even less likely that it will happen many times in direct succession. As as a result, one of the forks eventually grows longer and the other fork is abandoned or "orphaned". The transactions on the orphaned chain from the point of divergence onward are forgotten (including any mining rewards). It is as if they never happened. Forks can also take place if nodes simply decide to add different blocks to the chain and then continue to build on the forks regardless of size. This is a violation of protocol and is considered dishonest, but nothing can prevent nodes from doing so if they wish to.

3. Security

As we note in the introduction, all the advantages that blockchain offer depend on honest transaction verification and block-writing. Bitcoin uses the Satoshi Proof of Work (PoW) protocol outlined above, while Ethereum (the second largest cryptocurrency) and many other "alt-coins" use a variant. PoW protocols have in common that they are susceptible to 51% percent attacks. That is, if more than 50% of the nodes/miners are dishonest, then they can take over the chain and write any transactions they please. If 50% or more are honest and follow protocol, an authoritative fork will exist in which all transactions are correct.[5] As a result, Bitcoin and similar protocols are said to

5 However, Eyal and Sirer (2014) describe a type of coordinated attack by miners holding only 25% of Bitcoin's hashing power that can compromise the blockchain implying that Bitcoin is only 25% BFT.

have a Byzantine Fault Tolerance (BFT) of 50%. We discuss the value of BFT as a measure of security in more detail below.

In PoW protocols, nodes are generally run by anonymous agents. The principle of one CPU, one vote, applies. Any agent who is willing to bear the computational cost of trying to mine a block can join the validation network anonymously and as an equal. The hope was that this cost deters Sybil attacks in which many "fake" identities are created in order to gain majority control of the validation process. If votes must be paid for with work, then it is hoped that it should be unprofitable to mount such an attack.

In practice, many Bitcoin and Ethereum nodes are owned by the same real-world agent or are part of mining pools in which hardware may be owned by different agents, but which coordinate their efforts and share rewards. Many Bitcoin pools have chosen to self-identify which makes them vulnerable to pressure from state-actors or others. Mining pools are so concentrated at this point that if no more than three were to collude, they could mount a successful 51% attack. In effect, Bitcoin is not validated by thousands of independent nodes but depends instead on the honesty of three or fewer agents. Put another way, if Bitcoin has 10,000 nodes, an attack by only three agents would be successful. In a real sense, this means that the BFT of Bitcoin is only 3/10,000 or .03%.

One must wonder why these pools do not, in fact, merge. In any industry, mergers create market power. For example, if there are four firms selling cellphone service each making $\$X$ per year in profits, the monopoly created by merging all four firms would make more than $\$4X$ in profits. At worst, the merged firm could simply proceed as if it was still four separate firms and make exactly $\$4X$. Taking advantage of monopoly pricing or economies of scale, however, would certainly bring profits above $\$4X$. Merging is in the interest of the shareholders of all four firms. In the same way, if three mining pools are each making $\$X$ in net profits from mining rewards and transactions fees, the merged pool could make at least $\$3X$ by continuing to act as they did before. On the other hand, the merged pool would be able to mount a successful 51% attack and take over the Bitcoin blockchain, gaining whatever additional profits this might entail. The fact that they do not choose to do so must mean that something besides the PoW protocol is keeping them honest. The most likely candidate is the fear that stealing bitcoins would result in a hard fork[6] that would prevent the merged pool from profiting from its theft. In other words, to the extent that PoW blockchains are trustworthy, it is only because of the belief that "code is law" is a lie. It is in fact the threat to break protocol, not the protocol itself, that keeps Bitcoin and Ethereum safe.

Proof of Stake (PoS) is the other main approach to verification. Several banks, for example, might set up a private blockchain in which the members vote on whether a new block is correct and should be added to the chain. The banks put their reputations at stake in this case.[7] Stake can also be established by posting a bond (money or tokens), investing in the performance of useful services

6 In this case, the "hard fork" would involve breaking protocol and ignoring the "validated" blocks containing transactions that were judged as stealing coins. New blocks would be added starting from the last "honest" block. More generally, a hard fork takes place when part of a group decides to take a project in a new direction starting from the existing code base, dataset, or other IP. This is usually because the group has a different vision for the best path forward. Hard forks are especially problematic in context of blockchain because it breaks the rule that "code is law" by creating a fork with new laws and protocols. Even if the motivations are pure, if you can break the law for good, you can break the law for bad. Hard forks are very corrosive to the trustless, anonymous, distributed nature of blockchain.

on a platform, providing resources such as storage or bandwidth, participating in platform activities, etc. Voting power is then distributed in proportion to the stake. In many cases, stakeholders choose a smaller set of delegates to be their proxy and represent their interests.[8] Depending on the implementation, PoS approaches are both cheaper and more scalable than PoW. On the other hand, they generally offer a BFT of 33% or less, so these advantages come at the cost of some security. Of even more concern is that PoS protocols depend on the honesty of the stake-weighted super-majority of agents who have decided that it is worthwhile to acquire stake, and on the nonmanipulability of the voting or delegation system. Since the number of voting stakeholders or delegates (tens or hundreds) is typically much smaller than the number of validating nodes used by PoW blockchains (thousands or tens of thousands), collusion by validators is much more likely. It is not clear how much confidence a claim of 33% BFT should give us even if we believe it to be true.[9]

It is worth noting that even if stake-holders are numerous and anonymous, we run into the same concentration problem that we see in PoW protocols. If the profit a validator gets from posting a bond is worth it, why not post the same stake under many identities? At worst, each identity makes enough profit to pay for the cost of posting the bond. Creating enough identities to gain the majority of the total voting stake makes it possible for a single real-world agent to take over the blockchain. Again, it is only the threat of out-of-protocol actions that creates disincentives to make such an attempt.

4. Algorithmic Game Theory

Blockchain protocols have their roots in algorithmic game theory which adapts traditional noncooperative game theory for use in computational environments. The costs of calculating best responses, equilibrium outcomes, operations of the validation mechanism, and the complexity of the algorithm itself, are central concerns. For example, computer scientists worry that agents using complicated protocols without a complete understanding of how they work may have difficulty determining fully optimal actions. As a consequence, agents are often modeled as following *ad hoc* behavior patterns. For example, agents might be assumed to be either honest or malicious-type players since fully rational play may exceed their cognitive limitations. It is also seen as reasonable

7 Note that since all the participating banks are in the same sector, their economic fortunes are highly correlated. In a recession or financial crisis, all banks are likely to be under financial pressure, the threat of bankruptcy, and the possibility of being taken over by the Federal Reserve. It would not be at all surprising if a financial crisis, such as the one that began in 2007, were to result in five out of eight banks in a PoS blockchain being placed under federal supervision or forced to merge. Bank officers might be willing to take desperate measures to survive. The threat of a lost reputation is not much of a deterrent to a bank or any firm facing extinction. In addition, the identity of the validators is known and so they can be pressured by state-actors to break validation protocol in support of legal judgments or state policy.

8 There are also many hybrid approaches that use combinations of PoW and PoS or even more complicated means of choosing agents to verify transactions.

9 If a fixed set of stakeholders validate a blockchain, then honest behavior depends on the incentive structure faced by these specific agents. For example, one might be confident that the reputational damage of dishonest behavior would be enough to make Bank of America or Deutsche Bank behave correctly. When voting stakeholders can choose actions that affect their voting power, however, dishonest agents, who have the most to gain from subverting the blockchain, have the greatest incentive to expend the effort required. Thus, protocols that use escrowed tokens or Proof of Effort of some kind may end up systematically choosing dishonest validators. BFT loses its meaning as a measure of security in such cases.

to balance computational costs against security and nonmanipulability. Mechanisms that give desirable outcomes with high probability are considered to be good enough for the real-world applications. If the odds that one or a few bad actors can change the outcome of a mechanism are sufficiently small, then the mechanism is considered to be robust to malicious behavior.

Algorithmic approaches tend to pay less attention to certain other elements of games and mechanisms. The most important of these is that games often have multiple equilibria, some of which may be quite undesirable. What would lead us to expect that a desirable outcome is more likely than any one of the less desirable equilibrium outcomes in such a case? The fact that malicious actors are unable to affect the equilibrium outcome does not really matter if we are at the wrong equilibrium to begin with. Thus, designing protocols such that no undesirable outcome is a stable equilibrium is essential.

The centrality of Nash equilibrium in algorithmic approaches is also a concern. Nash equilibrium is only one of many equilibrium concepts in noncooperative game theory and is not a particularly strong one. Dominant strategy and coalition-proof equilibrium, for example, are much more robust and appropriate to computational situations. In addition, most mechanisms in computational environments are used many times in succession. For example, the Bitcoin protocol enables miners to write new blocks approximately every ten minutes. Miners engage in a race to solve a cryptographic puzzle that gives them the right to propose a new block and receive a reward. Once the puzzle is solved, the miners start working on the next block. In other words, the miners play the same one-shot game repeatedly. Understanding the sequential game derived from playing one-shot games many times in a row requires considering such things as the information available to agents, their beliefs about the actions and objectives of other agents, computational limitations on rationality, and the robustness of equilibria.

Algorithmic game theorists are certainly aware of these problems, just as economists are aware that computational costs can be a binding constraint for both players and mechanisms. We can all agree that consensus or other mechanisms that are trusted with billions of dollars of transactions must be extremely robust. If they are vulnerable to manipulations by coalitions of agents, have bad as well as good equilibria, or have incentive structures that make it profitable for agents to behave badly in a repeated game or under certain information and belief structures, then the fact than the odds are low that a single agent can subvert the mechanism in a given period is of little comfort.

5. Protocols and Mechanisms

The perspectives that economists and computer scientists bring to the table are different, and each have their value. The problems typically addressed by protocol builders and economics mechanism designers, however, turn out to be different in at least two important ways.

First, in most consensus protocols, the truthfulness of the validators is externally observable and provable. Bitcoin and other blockchains all have an established set of rules and procedures that are published and widely known. For a transaction to be valid, for example, the request must be signed with the private key of the outgoing account holder. Thus, for coins to be moved on the ledger from one account to another, there must be a properly signed transaction request recorded in the previous block. If the transaction request is not there, the signature is false, or the amount of tokens

moved is incorrect or is moved to the wrong account, anyone who can see the block is able to detect error. Similarly, if a block is committed to the chain without the correct nonce that proves a node expended the required work, or if the Merkle proof and the recursive hashing is false, it is easily detectable.

In contrast, most economic mechanisms are designed to find optimal or desirable outcomes when agents have private information. For example, auctions are designed to get agents to reveal how much they value an object and to make sure it goes to the agent who values it the most. Similarly, how much an agent might be willing to contribute to building a public good is known only to the agent in question. Agents therefore try to free-ride off of the contributions of others. Groves and Ledyard (1971) and Vickrey (1961) auctions (among many others) are examples of mechanisms aimed at making it incentive compatible for agents to truthfully reveal their willingness to pay for public goods. If the designer already knew these willingnesses to pay, on the other hand, he could simply impose taxes on agents and build whatever public projects were optimal. No mechanism would be necessary in this case.

Second, in economic mechanisms, the designer generally sets up a game in which he imposes both a strategy space and a payoff fuction. Agents may choose not to participate if they don't expect to benefit, but if they do participate, they have no alternative but to choose one of the permitted strategies and receive payoffs as determined by the designer.

In consensus protocols, there are also rules that are supposed to be followed and specific sets of messages and actions that are allowed. The payoffs, however, are distributed by the consensus mechanism itself. That is, the validating nodes decide for themselves whether to write rewards and punishments into the blockchain they are validating. They always have the option to ignore the protocol and write anything they wish to into the block. In addition, nodes may choose to take actions which are not allowed by protocol and may not even have been contemplated by the protocol builder at all.

6. Simplified PoW Protocols as Games

Validation protocols involve many agents with heterogeneous interests and constraints. From a mechanistic standpoint, the within-protocol message space is extremely complicated and the more general out-of-protocol strategy set is even more so. Given this, writing a clear payoff function would be an almost impossible task. This is probably why the Bitcoin protocol has never been fully expressed and analyzed as a formal game. Unfortunately, this is a task beyond the abilities of the present author as well.

Instead, this section and the next will consider simplified games that contain some of the most important elements of a generic PoW validation protocol. We argue that including omitted elements would not materially alter the conclusions we come to. We begin by considering what PoW might look like as a one-shot game.

Agents: Assume there are a total of N nodes, and N is fixed. In reality, nodes can join or leave the validation network, so the size of the network is endogenous. We will explore this in a limited way in the next section.

Strategies: This is a complicated question. In algorithmic game theory, this is often reduced to nodes choosing to be honest or dishonest (or maybe being innately honest or dishonest). In reality, there are an infinity of ways to be dishonest. For example a node could do any of the following:

- Propose a block without finding a nonce.
- Write block rewards incorrectly (double them, for example).
- Write a block that steals tokens only from dead accounts.[10]
- Write a block that steals tokens from live accounts.

Of course, many other things are possible and there is also the question of how many tokens to steal, from whom exactly, where to transfer the stolen tokens, whether to accept flawed blocks from other nodes and under what circumstances, and so on. To make matters worse, there are also many ways to be honest. For example, nodes can choose which transactions to include in blocks they are mining or to include none at all. They can distribute the task of finding a nonce over a pool of miners and share the rewards on whatever basis they choose. They can also renege on these agreements. All of these actions, and many others, are allowed within protocol and therefore considered to be honest node behavior.

Payoffs: Without a well-specified strategy space, we cannot even begin to write a payoff function.

Time to start simplifying. Let's begin by assuming that the tokens have a fixed value and that by moving them (honestly or dishonestly) into their accounts, nodes realize their full value. (This is unrealistic, but we will discuss the implications of relaxing this assumption below.) Given this, what sorts of Nash equilibria are possible?

Case 1: Suppose that some node n proposed stealing all tokens from both live and dead accounts and distributing them equally over the N nodes. Suppose that all nodes believe that the others will choose to accept the block and commit it to their version of the chain. Clearly, no single node would be made better off defecting from the strategy of accepting the false block. Accepting the block is therefore a Nash equilibrium.

Case 2: Now suppose that all nodes simultaneously produced dishonest blocks that stole some or all of the tokens and distributed them such that all nodes were given at least a small payoff. The logic above holds. If a node thinks that all the other nodes will accept any specific one of these proposed blocks, his best response is to do so as well. In other words, we have a kind of trivial folk theorem. Anything that is individually rational is a Nash equilibrium.

Case 3: What if (for some reason) it was necessary for at least half the nodes to accept the proposed block for the tokens to effectively be stolen. This incorporates the sort of consensus element that PoW protocols have.

10Approximately 20% of bitcoins have not moved between accounts in the last five years, and more than 40% have not moved in two years. It is likely that the owners of these accounts have lost their private keys. If this is true, the coins in these accounts are completely and permanently inaccessible to anyone. In effect, it is as if these bitcoins have been burned and are no longer part of the coinbase. See Conley (2017).

Note that the core is empty under this assumption. For an allocation to be in the core, all the tokens must be stolen since otherwise the smaller theft could be blocked by the grand coalition proposing to steal everything. A core allocation must also give a positive payoff to exactly $N/2+1$ nodes and zero to the rest. Otherwise, a blocking coalition would exist that could exclude some of the unneeded nodes and distribute their shares over a smaller coalition that has a minimal majority. Unfortunately, the $N/2-1$ nodes who are allocated zero could block this by offering to give the two least well compensated nodes in the current majority more than they are currently allocated and then distributing the remainder over themselves in any way they liked. Thus, there will always exist a blocking coalition for any allocation that might possibly be in the core and so the core is empty.

It is still the case that any feasible, individually rational allocation of any amount of stolen tokens is a Nash equilibrium. The logic from Case 2 applies. Of course this means that honest behavior is also a Nash equilibrium.

Case 4: Suppose that one node had the exclusive right to propose a block (for some reason). Then the protocol becomes an ultimatum game. All tokens are stolen and go to the block proposer except for the smallest possible amount given to $N/2$ nodes to make it rational for them to vote "yes" instead of rejecting the block and receiving a payoff off of zero. The Nash equilibrium and the core allocations are identical in this case. Empowering a single node to propose a block like this through random selection, election by stakeholders, Proof of Work, and other methods, is an element of many protocols.

Case 5: Suppose the block proposer was determined through Proof of Work (although why dishonest nodes would care that the proposer performed work is unclear). This does not materially affect anything as compared to Case 3. Any individually rational division of any amount of stolen tokens is a Nash equilibrium, honest behavior included. (This means that the proposer must receive at least enough to cover the cost of the work he performed.) The logic of Case 3 also implies that the core is empty.

One of the most important things that prevents these kinds of dishonest behaviors in practice is that if nodes are seen to be stealing tokens, token holders and potential token holders would lose confidence in the system.[11] Agents would be less willing to accept tokens in exchange for things of real-world value. Thus, dishonesty on the part of nodes is likely to decrease token value. The question is: by how much? This may depend upon the type of dishonesty the nodes practice.

For example, suppose a node proposed a block that doubled the mining reward he gets. This is out-of-protocol, but does not directly harm token holders (besides a little inflation). Would token holders choose to simply walk away from the blockchain and abandon their accounts? Who knows. It is certainly plausible that this might have something less than a catastrophic effect on token value. Users might see this as a misdemeanor violation of protocol.

11 Note that loss of token value is an endogenous disincentive for bad behavior that does not violate "code is law". An out of protocol "hard fork" would also prevent token theft, or at least make it unprofitable. We ignore this here because it is equivalent to central planning by an all powerful authority. If blockchains are subject to such authorities, then they are not, in fact, trustless and decentralized. There is no need to analyze the incentive structures of the protocols at all in this case. Focus should be on the motivations and constraints of the central planners who control the content of the ledger.

Case 6: Under the assumptions of Case 3, suppose that a node proposed a block that doubled the mining reward he gets from r tokens to $2r$ tokens. As a result, users lose some amount of confidence in the blockchain and token value decreases to a proportion $p<1$ of what it was previously. Assume that any other type of dishonesty results in token value going to zero. Since other nodes besides the proposer may also hold tokens on the chain, they must be compensated for their loss if they are to be persuaded to accept the block. Suppose that the $N/2$ nodes with the smallest number of tokens in their accounts and the proposer collectively hold a total of c tokens. Then if $(1-p)c<p2r-r$ the combined losses of the proposer plus the poorest half of the nodes is smaller than the gain the proposer gets from increasing his block reward. Therefore, any allocation that at least compensates the poorest nodes and the block proposer for the loss of token value and distributes the remaining tokens in any way at all is a Nash equilibrium and benefits the dishonest nodes.

With this very specific set of beliefs about the effect of dishonesty on token value, the range of dishonest behavior that can be supported as a Nash equilibria narrows considerably. Unfortunately, this should not increase our confidence very much. The broader point still stands: the set of possible Nash equilbria depends on how users react to different sorts of honest and dishonest behavior. More specifically, it depends on what nodes believe about how users will react. Beliefs such as these are in no way under the control of the protocol maker. What nodes believe and what is in fact true about the consequences of dishonesty is unknown, uncontrollable, but has a profound effect on node behavior and the proper functioning of the validation protocol. To be secure, a protocol must be robust at least to the entire range of beliefs that a node could plausibly be expected to hold.

To sum up:

- The core may be empty or trivial.
- Almost any individually rational allocation is a Nash equilibrium for at least some set of beliefs.
- Whether nodes can profit from dishonesty depends at least as much on user behavior and the beliefs that nodes have about user behavior than on the details of the validation protocol.

7. More Complicated PoW Games

The PoW game discussed in Section 6 leaves out many complexities. The most important are that real PoW games are both dynamic and probabilistic. Nodes have a chance of being allowed to propose the next block and earning the block reward that is proportional to the hashing power they bring to the game. This means that they have an investment and care to some greater or lesser degree about the value of the chain in the future.

One of the major questions that treating validation protocols as a repeated game must address is what happens if forks in the chain occur. By protocol, no valid chain can contain an error (or theft). Such a chain or fork of a chain is *per se* invalid. Protocol requires that newly mined blocks are appended to the longest fork available that is without error. What happens, however, if no fork without error exists? What if a chain with an error is by far the longest? What if there are multiple honest forks that refuse to sync? PoW protocols do not adequately address these questions. Even if they did, the more important issue is how users and nodes might react to such situations.

As an example, what if a majority coalition decided to ignore correctly mined blocks from other nodes, but otherwise did not violate protocol or steal tokens. Users would see that the majority coalition is writing the longest fork and that it has no errors (although the shorter fork written by the minority is also correct). We don't know for sure what users would do even if they were aware and could prove that the majority coalition was ignoring valid blocks produced by nodes in the minority coalition. We speculate that their probable action would be to accept that transactions on the longest chain as definitive despite the failure of the majority coalition to abide by protocol and accept newly mined blocks from the minority nodes. If users refused to acknowledge the validity of transactions on the shorter chain, then any tokens appearing only in accounts on the minority fork would be orphaned. In particular, the block rewards for the minority coalition would not be present on the majority fork and so would not be valid in the eyes of users.

Why would the majority coalition do this? There is no immediate benefit. A coalition of Bitcoin nodes with 60% of the hashing power would mine blocks approximately every 17 minutes on average regardless of whether it accepts or rejects blocks from the minority. Thus, the mining rewards to the majority coalition would be the same; blocks would just get mined more slowly over all since the hashing power of the minority is not allowed to contribute. The minority coalition, however, would not be getting paid mining rewards on a chain users recognize as valid since none of their blocks end up being included in the authoritative chain. This removes any incentive that minority nodes have to to stay and continue exerting computational effort to validate the chain. The minority nodes would therefore shut down or move on to a different blockchain. Recall that the Bitcoin protocol checks average block writing intervals every two weeks and responds automatically by making the mining puzzle harder or easier so that blocks are written every 10 minutes. Since this strategy results in blocks being written every 17 minutes, the protocol adjusts the puzzle's difficulty downward so that the majority coalition ends up getting block rewards every 10 minutes with only 60% of the effort.

This makes mining profitable in an economic sense, and one might think that this should incentivize other agents to enter the network and act as nodes to take advantage of the arbitrage. Potential entrants, however, would realize that if they joined the network, any blocks they mine would be ignored. Thus, the majority coalition ends up being the totality of the mining pool and does not need to worry about its position being contested. It might even start powering down nodes so it could get the same rewards with less energy cost. Of course, it could also take advantage of its position by stealing tokens on the only fork that exists.

A deeper problem is what happens if the network fragments either because of true network segmentation or strategic behavior. If there are many forks of varying lengths containing errors of greater or lesser importance, how would or should users respond? How should miners estimate the value of following a strategy that produces such a situation or respond to it if it happens? The answer, of course, is that it depends on expectations of how users and other nodes react.

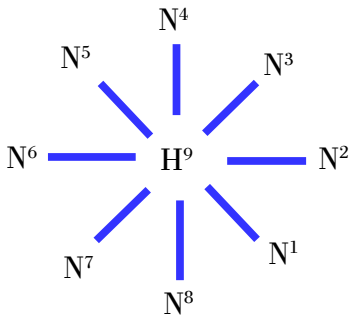
The conclusion here is that the incentives that nodes have to behave honestly depend critically on how users react to the existence of forks and errors, or more accurately, how potentially dishonest nodes estimate that they will. *The fact that users can identify dishonest nodes and forks does not empower them to do anything within protocol about it.* Users are left with a set of bad choices. For this and other reasons, theory does not help us to reject many belief structures as wrong or irra-

tional. Unfortunately, there is not much empirical evidence to help us estimate which belief structures are likely to emerge either. All of this suggests that the incentives contained in PoW and other existing protocols provide very weak security guarantees.[12]

8. Random Hub and Spoke Blockchain Architecture[13]

In this section, we construct a random hub and spoke network for communications between validating nodes. Hub and spoke networks transmit the minimal set of messages possible and also allow messages to reach the entire network more quickly. In contract, messages on gossip networks may be repeatedly sent to peers who have already received them from other peers. The time it takes a message to find a path to all members of the network is random and may be non-trivial. Delays in the reception of newly mined blocks, for example, is one of the reasons that Bitcoin experiences forks. The main attraction of gossip networks is that nodes can join and leave anonymously and that no central list or server that night be blocked or manipulated exists. Both of these elements are essential for a validation network to be trustless, robust, and uncensorable. We will show below how a hub and spoke network can be designed to share these features.

The simplest network topology is for a single hub to coordinate the building of each block. For example, a nine node network with node number eight serving as hub would look like the following:



More complicated structures with multiple layers of hubs and random participation rules are possible and may be desirable, depending on the needs of the chain being validated.

Our ultimate objective is to design a blockchain protocol that is 100% BFT. The remainder of this section outlines a work-flow or architecture using a hub and spoke network that allows us to create incentives for honest nodes to report dishonest ones, and for users to be able to detect dishonest behavior by inspecting the blockchain.

The first question to address is how the set of validating nodes is established and maintained. In the Bitcoin protocol, nodes simply broadcast their existence on the P2P network and begin mining blocks. No central list of nodes is kept and nodes maintain their anonymity. Nodes can come and go as they please. Leaving the network is accomplished by no longer sending or receiving messages on the P2P network.

12 As an aside, similar scenarios can be constructed for PoS and DAG protocols. In fact, they are easier in many ways, but, the same fundamental logic and conclusions hold.

13 This is a simplified version of the protocol used by GeeqChain. See Geeq.io for details.

A hub and spoke network requires a bit more structure. Our approach is to keep a roster of all nodes that are currently part of the network in an Active Node List (ANL) which is maintained as an element of every block that is added to the chain. Membership to the validation network is open and agents can get themselves added to the ANL by submitting the proper join request transaction to an existing node. Joining as a validating node also requires the posting of a Good Behavior Bond (GBB), explained later. Thus, a join request transaction includes a public key, an IP address, and permission to transfer enough tokens from the public key address to a system account to cover the GBB. Agents currently in the ANL can also submit resign requests, in which case the GBB is transferred back to the node's public key account and the node is removed from the ANL.

Note that this ANL approach allows nodes to maintain their anonymity. As in the Bitcoin and Ethereum networks, nothing besides a node's IP address and token account is known. The ANL is also distributed in the sense that it appears identically in the blockchains maintained separately by each of the nodes in the network. There is no central server or single authoritative list that might be blocked or censored. At a more detailed level, the Hub and Spoke (HaS) architecture we require uses the following work-flow:

Hub and Spoke (HaS) Architecture

1. Receiving Unverified User Transactions (UUT): Users send UUTs to one of the nodes on the network. This includes join and resign requests for the ANL.

2. Waiting for commitment: Each node accumulates UUTs until the block currently under construction is complete, verified, and committed to the existing blockchain. This process is described in the steps below. Note that each node constructs and keeps its own copy of the block chain, so "commitment" is a local phenomenon that takes place separately at each node.

3. Choosing a hub: Once a node commits a new block, it can calculate a new Current Ledger State (CLS), that is, the current state of all accounts and records given the previous ledger state updated by the block of new transactions just committed. Which node will serve as hub for the Block B+1 is determined by the hash of the CLS for block B. Since the set of transactions that go into block B are unpredictable, the hash of the CLS is effectively a random number that cannot be controlled by users or the nodes in the network.[14]

4. Sending Node Transactions Bundles (NTB) to the hub: Each node takes the set of UTTs that have arrived since the last block was committed, signs each one individually, bundles them together and signs the bundle collectively, and then transmits this to the current hub. This bundle is called the NTB and also includes the block number it is intended for as well as a hash of the node's CLS.

5. Sending the Hub Transactions Bundle (HTB) to all nodes: The current hub signs each new NTB as it arrives, bundles them together into an HTB and signs this, and then sends the HTB back out to each node in the ANL.

14 Actually, some additional protocol elements we will not detail here are required to ensure that the hash is impossible to manipulate.

6. Verifying the HTB and checking for honesty: All nodes (including the hub) start with the same CLS and the same HTB. These are used in combination with the business logic of the chain to check the validity of each UUT to create a set of Verified User Transactions (VUT). Protocol also dictates that each node creates a set of transactions for fee payments to members of the ANL for their validation and virtual machine services. Finally, each node verifies that the CLS hashes that all the other nodes included in their NTBs are identical to its own. If so, then there is unanimous agreement about the CLS after the last block was committed. If any CLS is different, or if the node detects dishonest behavior from inspecting the HTB, then the node initiates an audit using the Catastrophic Dissent Mechanism (CDM) outlined in Section 11.

7. Building a Proposed Transactions Block (PTB): All nodes build a PTB by putting together VUTs, fee transactions, and any transactions and evidence needed to start an audit of any dishonest nodes.

8. Updating the CLS: Each node calculates the new CLS given the PTB it just created. The CLS is kept in a read-only public directory accessible to users and others who want to know the state of the chain and also check it for accuracy.

9. Committing the Full Verified Block (FVB): Each node creates a new block to be appended to the end of its copy of the chain. This involves creating a Committed Block Header (CBH) that includes the block number (that is, the "height" of the block being added), a hash of the previous block, and a hash of the CLS for the previous block. The PTB and HTB[15] are added after the header to complete the FVB which is appended to the local copy of the blockchain by including a hash of the previous block to add on to the Merkle tree.

10. Choosing a new hub: The process begins again with each node returning to step 3 and sending a NTB to the new hub that includes the transactions that have arrived while the current block was being created, verified, and committed to the chain.

One problem common to all blockchain validation protocols is the possibility of strategic network behavior. This is a very serious, but largely ignored, attack vector for dishonest nodes that is independent of falsely verifying transactions, failing to follow staking or work protocols, choosing not to execute smart contracts, committing false results to a block, and other violations of verification and block-writing protocols. The key difference is that strategic network behavior does not always leave evidence in the validated blocks that would allow users to see and prove that nodes are not following communications protocol. An example was described in Section 7 in which a group of nodes chooses to ignore valid blocks it receives on the network if they happen to come from nodes not in their coalition. To make this problem even more difficult, it is often not possible to distinguish between a situation in which a node fails to transmit a required message or falsely claims not to have received one, and genuine network segmentation, failure, or latency.[16]

15 Including both the HTB and PTB is redundant since the PTB can be directly derived from the CLS and the HTB. The HTB must be included in the block because it includes all the raw transactions and signatures that allow for independent audits to expose misbehaving nodes. Including the PTB makes the blockchain easier for humans to understand even though it is not strictly necessary. Although including both does not increase the bandwidth cost of building the chain, it does increase the amount of storage required.

Dealing with these types of network games is challenging. There are, however, a number of mechanisms that can be used to detect, mitigate, or prevent such behavior. For now, however, we will set these problems aside while acknowledging that the claims made in this paper depend on the existence of a solution. To be slightly more formal we will assume Perfect Nonmanipulable Networks:

Perfect Nonmanipulable Networks (PNN): The network that nodes use to communicate is fully functional in the sense that it allows all nodes and users to send messages to one another without latency. In addition, if any node fails to send a message required by protocol or falsely claims that a message was not received, it is provable.

We conclude this section with an inductive proof that PNN implies that if a node building a blockchain using the HaS architecture described above does not follow protocol it will be provable.

Claim 1: *Assume that all nodes in the block B-1 ANL have identical CLSs. Then assuming PNN, either all the nodes will produce identical PTBs and FVBs and end up with identical CLSs for block B, or it will be provable that a node failed to follow protocol.*

Proof*:* See the first appendix for all proofs.

The intuition for Claim 1 is the following: First, PNN directly implies that any failure of nodes or hubs to transmit messages as protocol requires can be proven. Second, given all required messages are provabley sent and received, they must be correctly signed by hubs and nodes. In other words, hubs and nodes cannot avoid responsibility for a message by failing to attach the proper cryptographic signature. Third, all UUTs, NTBs, and HTBs must be correctly constructed. Failure to do so by the actors responsible is either provable or harmless. Fourth, these three facts together imply that any attempt to steal tokens, falsify transactions, or deviate in any way from protocol can be be traced to its source which makes it possible to audit the dishonest actor out of the ANL. Thus, all nodes that start with the same block $B-1$ CLS must end up with the same block B CLS or be probably dishonest.

Claim 2: *Assuming PNN, all the nodes will produce identical PTBs and FVBs and CLSs for all blocks or it will be provable that a node failed to follow protocol.*

This is simply the induction step of the proof. We know that the CLS is correct at the block 0 by definition. Claim 1 proves that if the CLS is correct at block $B-1$, it must be correct at Block B or it will be provable that a node failed to follow protocol. Thus, the CLS, etc. must be correct for every block or it will be provable that a node failed to follow protocol.

16 The CAP Theorem tells us that if we insist that all nodes in a network communicate and agree on a ledger update, we cannot guarantee that consensus will be achieved in finite time. In other words, if a mechanism sets an upper bound on the time to achieve consensus on a ledger update, then it must be robust to the possibility that some nodes will either not communicate with the network or will have different or incomplete views of the ledger update. Assumptions similar to Perfect Nonmanipulable Networks are commonly made, explicitly or implicitly, by many protocols. GeeqChain's protocol includes elements that either prevent nodes from using harmful network strategies or identify and exclude the nodes that do.

9. Proof of Honesty

In this section, we describe a simple mechanism called Proof of Honesty (PoH) that leverages the provability of dishonest behavior to produce a new approach to blockchain verification that is 99% BFT. To be precise:

99% Byzantine Fault Tolerance: A blockchain is 99% BFT if it is impossible to steal tokens from rational, honest users provided that there is at least one honest node that follows protocol.[17]

Rational Honest User: A user is honest if he does not participate or in any way benefit from dishonest behavior by validating nodes. A user is rational if he chooses to transact only on forks that maximize the value of his tokens.

We proved in the previous section that if networks are perfect and nonmanipulable, then any failure to follow protocol by nodes is provable. Of course, many types of dishonest behavior are detectable and provable in the Bitcoin, Ethereum, Practical Byzantine Fault Tolerance (PBFT), PoS, and other consensus systems. Indeed, one of the attractions of blockchain is the auditability of the records of transactions that allows the accuracy of the ledger state to be independently verified. The problem is that under existing protocols, there is nothing that users can do if they find dishonest behavior. Transactions are sent through the P2P network and so can be seen and processed by all nodes on the network, honest or not. There is simply no way for a user to exclude any nodes he can prove are behaving dishonestly. Even if he could, the protocols define which of the forks that might exist is authoritative. Users have no independent say.

PoH takes a new "user-centric" approach to blockchain validation. PoH gives users the authority to determine whether a node, fork, or chain is honest, and to choose to transact only with honest nodes. In other words, users are the arbiters of truth under PoH. Since users hold tokens on the chain which are at risk of being stolen by dishonest nodes, this is inherently incentive compatible. In contrast, PoW, PoS, Direct Acyclic Graph (DAG), and all other blockchain consensus protocols of which we are aware are "node-centric" in the sense that nodes, through some mechanism, are the ultimate arbiters of truth and authority. Since nodes are the agents who might benefit from falsely writing transactions that steal tokens, this creates an inherent conflict of interest.

PoH is extremely simple, but requires three things. First, that users have access to the code that honest nodes are supposed to run to validate any given blockchain. Note that this code might include smart contracts, business logic required for some specific use case, special accounting rules for token creation, destruction, or transfer, and so on. Second, that the code be deterministic in the sense that a given set of inputs (transactions, blocks, the existing state of the ledger, etc.) produces one and only one output. Finally, that if any node fails to follow the protocol contained in the code, the deviation can be detected and proven by all users.

17 Although we call this 99% BFT, it really is $(N-1)/N$ BFT where N is the number of nodes in the validating network. Note that if there is a single honest node, there is also exist an honest fork in which all the protocols and logic of the blockchain (business logic, smart contract execution, and so on) are followed and where an honest ledger state may be found.

Of course, it would be unreasonable to think that ordinary users are sophisticated enough to understand a protocol and be able to check a blockchain thoroughly for dishonesty. In practice, checking the honesty of a fork can be done automatically by the client software that users employ to interact with the blockchain. Due diligence regarding the honesty of forks is therefore effortless and invisible from the user's perspective. The client software can be set to any level of paranoia the user wishes. If a user is about to accept a larger number of tokens, he might have his client inspect the chain and any forks in great detail and insist that the transfer take place on the fork he considers to be authoritative. On the other hand, if the blockchain is used for making micropayments between connected devices (IoT or the Internet of Things), then the user may choose to forego due diligence entirely. From a mechanistic standpoint, Proof of Honest works as follows:

Proof of Honesty

1. Chain Discovery: Users discover a given blockchain as well as any forks that might exist. In practice, users might be directed to a node that validates an application that a user wishes to use or find a node though a web search or by consulting a forum. Once he discovers any node, however, he can read the ANL and thereby find the IP address of all other nodes that are validating and keeping copies of the blockchain.

2. Honesty Checking: Users inspect the chain and its forks, if any, to any degree that they wish in order to determine the honesty of the nodes and the validity of the chain or forks.

3. Transaction Creation: Users choose a node and send it a transaction.

Without further elaboration, this simple idea produces blockchain with a BFT of 99%. That is, if even a single honest node exists, then rational, honest users will discover it and choose to write their transactions to the chain it verifies. Formally:

Claim 3: *Rational, honest users will always choose to transact on an honest fork if one exists.*

The intuition is that accepting tokens on a dishonest fork is risky. Such tokens are likely to have been stolen, and are likely to be stolen from any user who accepts them. It is therefore also likely to be difficult to convince other users who can also prove that the fork is dishonest to accept them at face value in the future. Rational agents will therefore prefer tokens on honest forks to dishonest forks since they are worth more.

Claim 4: *Assuming PNN, a blockchain using the HaS architecture outlined in Section 8 verified using PoH is 99% BFT.*

This is almost immediate. Regardless of how many tokens are stolen or improperly added to accounts on dishonest forks, none of these dishonest transactions ever appear on honest forks. Thus, all agents, honest or not, always have the correct balances on honest forks.

To sum up, we have reduced the problem of secure blockchain validation to making sure there exists at least one honest node in the network. It would not matter if there were a hundred or a thousand dishonest nodes in the ANL. In fact, it would not matter if the NSA, Russia, and China

combined their computational resources and flooded the validation network with dishonest nodes. Unlike existing protocols, PoH does not depend on the consensus view of dishonest nodes. Instead, PoH allows users to identify dishonesty and ignore it.

An implication of 99% BFT is that if there is at least one honest node, there is no gain to other nodes from being dishonest. In other words, if nodes know that at least one node is honest, there is no point to writing a dishonest fork. It will be ignored by honest, rational users. In effect, the dishonest nodes would be writing a fictional ledger that honest users would ignore. The dishonest fork will therefore end up being orphaned. At best, the dishonest fork will include only dishonest agents stealing tokens from one another that no one else would be willing to accept. Given this, the best response of any self-interested node is to behave honestly. In short, the existence of a single honest node is enough to compel all the other nodes to follow suit.

10. An Example of a Unanimity Game

Needless to say, 99% BFT is a large improvement over existing consensus protocols that offer 50% BFT at best. To bring security to 100% BFT, we must add another element to the mechanism based in a kind of unanimity game. We will outline this formally in Section 11, but let us begin with a motivating example that provides some basic intuition. Consider the following:

Agents are offered a chance to play a game in exchange for a one dollar admission fee. Each player who pays the fee is sent to a room where a name is written on the wall. Players are asked to write this name on a piece of paper. The papers are then gathered and compared. If they all have the same name, then each player is paid two dollars. If there is any disagreement about the name, all players get zero (which gives each a net payoff of negative one dollar).

It is easy to see that truth-telling is a Nash equilibrium. Suppose that one agent sees that all other players have reported the truth. Clearly, his best response is to tell the truth as well. Reporting the correct name gives the agent a net payoff of $\$1$ while any other report gives him a payoff of $-\$1$.

Unfortunately, this game has many other Nash equilibria as well. For example, suppose at least three players make different reports. All players would get a payoff of $-\$1$ in this case. Since no single player could change his report and generate unanimity, all of the player's strategy choices result in a payoff of $-\$1$. In other words, all reports are (equally bad) best responses for any individual agent.

Alternatively, all the players might get together before the game is played and agree to coordinate on one specific untrue report. In that case, each player would get a payoff of $\$1$ and no single player would benefit from unilaterally changing his report and telling the truth. Of course, coordinating on a false report does not yield a larger payoff than coordinating on the truth, nevertheless, both are Nash equilibria.

In the context of blockchain, it might be the case that players could profit substantially if they coordinated their efforts. Suppose we modified the game so that if all players write down the same name, then the named agent gets $\$1000$. Truth-telling and discoordinated reports would still be Nash equilibria, however, players would profit more if they coordinated correctly. For example,

they could agree to write down one of their own names and then split the $1000 received. In so doing, players would still get the $1 payoff for their unanimous reports and would get an equal share of $1000 in addition. Agents, therefore, have a positive incentive to collude, unlike the simple game first described.

To fix this, we could alter the game again to allow a small amount of auditing. Suppose we required all agents to sign their reports. If the reports are unanimous, then agents get a payoff of $1 and $1000 goes to the agent they name. However, if the reports are not unanimous, then the door to the room is opened, and the name on the wall is read. Any player who wrote down the correct name would get a payoff of $1 plus an equal share of a $1000 bonus. Players who lie would receive nothing and be banned from ever playing the game again.

With this addition, truth-telling becomes a dominant strategy. That is, regardless of what other players report, it is a best response for each individual to report the truth. Better still, truth-telling is a coalition-proof equilibrium. That is, no group or coalition of agents, including the coalition containing all the agents, could profit from lying. Even if all agents were able to agree on a false report, any single agent who reneged on the agreement and told the truth instead would get a payoff of $1001. If several agents reneged, they would get to share the bonus divided among the set of defectors (which would be more than the $1000 shared over all players that a coordinated false report gives). Thus, truth-telling is always a better strategy than lying or trying to collude. In other words, truth-telling is the *only* coalition-proof equilibrium. As a result, there would never actually be a need to do an audit and pay the bonus.

11. Catastrophic Dissent Mechanism (CDM)

In this section we build on the intuition of the unanimity game above to provide a validation protocol that is 100% BFT. More specifically, we develop the Catastrophic Dissent Mechanism (CDM) and show that when combined with PoH and a HaS architecture, the result is a blockchain with Strategically Provable Security (SPS).

As we point out above, all we need is a single honest node to ensure honest blockchain validation. But what if there are no honest nodes or honest forks? Even though users can see this, PoH depends on the existence of an honest fork, so it would seem that users are out of luck in such a case. The CMD is designed to deal with the possibility that 100% of the nodes are dishonest and able to coordinate their actions to steal tokens or otherwise compromise the integrity of the blockchain.

To make things realistic (and our life more difficult), we will assume that all nodes are dishonest in the sense that they will always do the most profitable thing available. We will not make the typical assumption that at least 50% (or more) of the nodes are honest. To an economist, everyone is dishonest. (Economists are not very nice people.) We will also assume that nodes are able to costlessly coordinate with one another. Many protocols implicitly rely on the difficulty of coordinating or committing to dishonest strategies on a large network for security. It is felt that this makes it unlikely or too costly for a large group of nodes to exploit coalitional strategies. Finally, we will not assume that nodes are separate and independent. Since nodes are anonymous, we can never really

know who stands behind them in the real world. It might be that one real world agent runs hundreds of allegedly independently validating nodes. This is the so-called "Sybiling" problem. A large network may in fact be run by a small number of real world entities. This means that groups of nodes share common interests instead of competing ones.

Thus, we assume that:

- All nodes are dishonest.
- All nodes are able to communicate and coordinate and trust one another not to deviate from agreements to be dishonest and share the resulting profits.
- Many of the nodes may be Sybils run by the same agent.

We achieve SPS using the CDM which works as follows:

Catastrophic Dissent Mechanism

1. Verifying CLS agreement: At step 6 of the HaS work-flow, each node verifies that the CLS hashes that all the other nodes included in their NTB are identical to its own. If any CLS is different, or if the node detects any other dishonest behavior from inspecting the HTB, then the node initiates an audit.

2. Auditing dishonest nodes: Initiating an audit requires that the node create a special audit transaction to be included in the next NTB sent to the hub that coordinates the building of the next block. The transaction includes a formatted description of the claimed dishonest behavior and evidence from the HTB or the versions of the CLS or blockchains kept by the dishonest nodes.[18] Note that all honest nodes will detect the same dishonest behavior and will create identical audit transactions.

3. Checking audit claims and punishing dishonest nodes: These audit transactions are included in the HTB sent to the nodes as the next block is built. Nodes verify the claims, and if they are true, write a punishment transaction that confiscates the GBBs of the nodes found to be dishonest and shares them equally over nodes who sent in the audit transactions that exposed the dishonest behavior. In addition to confiscating the GBB, honest nodes also write a special ANL transaction ejecting the dishonest nodes.

Now consider what it would take to hold a coalition of all the nodes together in unanimous conspiracy of dishonest behavior. How much could such a conspiracy steal and distribute to its members? This depends on how users would react to blockchain validated by 100% dishonest nodes. Would users simply walk away from their accounts, or would the tokens have some residual value?

18 One might wonder what happens if a dishonest node is chosen as hub. Why would a dishonest node put audit transactions against itself in the HTB it is supposed to send to nodes if this results in the confiscation of its own GBB and being tossed out of the ANL? For now, we appeal to the PNN assumption. If the hub is dishonest, this fact is detectable. Honest nodes are allowed under protocol to ignore provably dishonest nodes and so would use the hub choosing procedure to identify a new hub. If the new choice was provably dishonest, the process is applied again. This continues until an honest hub is found or we run out of alternatives. If there is only one honest node, it ends up being chosen as its own hub, and so the audit transaction still gets written into the honest fork that it maintains.

The smaller the residual value, the less benefit there is from forming such a conspiracy. To reduce this value, we add another element to the CDM called the Catastrophic Recovery Procedure (CRP). There are a number of ways that this might be implemented, so we only give a sketch of one possibility here:

Catastrophic Recovery Procedure

1. Any agent can restart the blockchain from the last honest block with himself as the only node: If no honest fork exists, then any agent (whether or not he was a user or node on the blockchain in question) is authorized by protocol to set up a new version of the blockchain. This new version starts from the last honest block and CLS, includes audit transactions that show that all the nodes in the current ANL are dishonest as well as a new ANL including only the agent invoking the CRP.

2. Restarted blockchains merge: These new chains search for each other and attempt to merge. That is, coalitions of agents who invoked the CRP abandon the new chains they created by themselves and create similar chains with all the CRP invokers included in the ANL.

3. The merged chain with the largest ANL is declared authoritative: After a prescribed time has passed, no further mergers are allowed. At this point, the new chain with the largest number of nodes in its ANL is declared authoritative.

4. Users transact on the authoritative chain: Users start to write transactions as before to the authoritative chain.

5. GBBs for nodes in the ANL are rebuilt: Transaction fees are increased temporarily with half going to nodes in the ANL, and half going to a system account to rebuild the GBB for nodes that participated in the CRP.

The CRP is costly, confusing, and extremely undesirable. However, it prevents a coalition of all nodes from being able to force users to choose between accepting their dishonesty or walking away from their accounts. More importantly, it changes how a dishonest coalition of the whole calculates the value of its conspiracy.

Given the CRP, tokens on the dishonest chain created by the conspiracy of the whole have no residual value. Eventually, a new honest fork will come into existence, and the accounts on the dishonest fork will be orphaned. This seems to imply that the only way to profit from dishonest coordination is to somehow move the stolen tokens off the chain before this happens. Of course, this is literally impossible. Tokens can only exist on the chain. Thus, dishonest nodes would need to find a way to *move the value* of the stolen tokens off-chain if they hope to profit.

Moving value off-chain would require that some gullible agent or agents agree to give up something of value in the real world (or on a different blockchain) to the dishonest nodes in exchange for stolen tokens. For example, a gullible agent might agree to "buy" stolen tokens and transfer dollars to the bank accounts of dishonest agents, or transfer the title of a car to dishonest agents in exchange for tokens written to the gullible agent's account on the dishonest fork.

This is rather far-fetched, of course. To make this scam work, the conspiracy would need to find gullible agents who are willing to give up significant value despite the fact they could easily prove that the tokens are stolen and will have no value when the CRP is complete. How much might be stealable is unclear, but the CRP should limit the amount significantly. We formalize this in the next section.

12. Strategically Provable Security

Byzantine Fault Tolerance is the standard metric of blockchain security. In the sections above, we showed how PoH could achieve 99% BFT. Nevertheless, BFT is ultimately not a good way of thinking about how nodes behave. Nodes on a validation network are not automata that innately work correctly or incorrectly, honestly or dishonestly. Nodes are the agents of the human beings who program and run them. Thus, any meaningful security measure must encompass the motivations and optimal responses of these humans given whatever they might believe about the behavior of other users and validators on the system. In other words, to be meaningful, a security measure should have a game theoretic foundation. With this in mind, we offer the following:

Strategically Provable Security (SPS): A blockchain has Strategically Provable Security if truth-telling and faithful execution of the protocol by all the validating nodes is the only coalition-proof equilibrium for any reasonable belief structure.[19]

Less formally, following protocol is one of many strategies that validating nodes could follow. There are an infinity of ways that a single node, a coalition of some fraction of the nodes, or even a coalition of all the nodes working together, could violate protocol. For example, we have already discussed the possibility that all nodes might collectively agree to steal tokens from agents by making transfers without properly signed transaction requests from account owners and also decline to submit audit transactions to report any of this dishonest behavior. To prove that such behavior is not a coalition-proof equilibrium, we will need some notation to formally define the game.

V – The total value that dishonest nodes think they can move off-chain.

N – The number of validating nodes.

A – The number of independent agents running validating nodes.

H – The number of nodes who decide to honestly report the bad behavior.

D – The number of nodes that follow the conspiracy and behave dishonestly. $(D+H=N)$

G – The amount held in the GBB of each node.

19 An equilibrium is coalition-proof if there does not exist an alternative joint strategy profile for the coalition of the whole, any subcoalition, or a single agent that yields a higher payoff to all members of the coalition. The notion of what constitutes "reasonable beliefs" will be discussed below. Also note that if a protocol satisfies SPS, it is also 100% BFT.

T – The present value of transactions processing fees to nodes who behave honestly.

Belief structures of nodes are what determine V, the total value that dishonest nodes think they can move off-chain. Users are able to detect dishonesty and know that even if all nodes are dishonest, a new, honest version of the blockchain will appear in short order. Thus, it would seem that "reasonable beliefs" about how much could be moved off-chain should be quite low. Theorem 1, below, allows for V to take any value at all and so does not build in any particular limitation on beliefs. However, the larger V is, the larger the GBB required to guarantee SPS.

PoH/CDM Blockchain Game

Agents:

$n \in \{1 \ldots N\} \equiv \mathcal{N}$ (validating nodes)

Strategies:

$v_n \in \mathbb{R}_+^1 \quad \forall n \in \{1 \ldots N\}$

Payoff Function:

$F_n : \mathbb{R}_+^N \rightarrow \mathbb{R}^1 \quad \forall n \in \{1 \ldots N\}$

We interpret v_n as the amount that agent n plans to steal and move off-chain for its personal gain.

If $v_n = 0$, then node n is behaving honestly and also reports any thefts by other nodes.[20]

If $\sum_n v_n > V$, then we interpret the strategy profile as coordination failure which results in all dishonest nodes being unsuccessful in their attempts to steal tokens.

This implies the following payoff function:

$F_n(v_1, \ldots, v_n, \ldots v_N) =$

T if $v_n = 0 \quad \forall n \in \{1 \ldots N\}$ (all nodes honest)

$T + \frac{GD}{H}$ if $v_n = 0$ and $\exists m \in \mathcal{N}$ such that $v_m > 0$ (honest with dishonest nodes)

$-G$ if $v_n > 0$ and $\exists m \in \mathcal{N}$ such that $v_m = 0$ (dishonest with honest nodes)

20 It is possible that a node might steal zero but be dishonest in other ways, or steal zero and not report dishonest nodes. We ignore these possibilities as they are dominated strategies and including them would needlessly complicate the description of the game.

$-G$ if $v_n>0 \ \forall n\in\{1\dots N\}$ and $\sum_n v_n>V$ (dishonest coordination failure)

v_n if $v_n>0 \ \forall n\in\{1\dots N\}$ and $\sum_n v_n\leq V$ (all nodes dishonest)

The reader might notice that this is a one-shot game while blockchain validation is really a repeated one-shot game. Our reason for this approach is that it is more straight-forward to truncate the infinite sequential continuation game into a single period. That is, nodes may think that it will take many periods for all agents to figure out that the chain has been compromised and plan complex strategies of obfuscation to move V off-chain over a number of periods. The real strategic choice, however, is whether to validate the chain honestly in all periods or follow the profit-maximizing, multi-period path of optimally collusive dishonest behavior. Thus, we can truncate this decision to a one-shot game.

Claim 5: *Consider a blockchain with HaS architecture validated by PoH backed up by the CDM and CRP and assume PNN. Suppose that all the nodes join a conspiracy to steal and move off-chain the maximum value they believe to be feasible. (That is,* $v=(v_1,\dots,v_N)$ *where* $\forall n\in\{1\dots N\}$, $v_n>0$ *and* $\sum_n v_n=V$. *) Then if* $G>\frac{V}{N(A-1)}$, *at least one agent running a node will be better off reporting the conspiracy than following through on his commitment to be dishonest.*

Claim 5 says that if the GBB is greater than $\frac{V}{N(A-1)}$, then dishonesty is not coalition-proof. To get a sense of what this means, suppose that nodes believed that they could move $V=\$10{,}000$ off-chain if they formed a dishonest collusive coalition of the whole. Suppose that there were 100 nodes and all were independent $(A=N=100)$. Then a good behavior bond of at least $\$10{,}000/100(99)$, would be enough to prevent the conspiracy. Suppose that the $GBB=\$1.10$, for example, and consider the incentives of a single node. If the node stays in the conspiracy, it gets an equal share of the proceeds, $(\$10{,}000/100=\$100)$. If he reports the conspiracy, he gets to collect the GBB of the other 99 agents who followed the conspiracy $(\$1.10 x\, 99=\$108.90)$. Thus, a GBB of approximately $\$1$ is enough to prevent the theft of $\$10{,}000$. If nodes thought they could steal $\$1{,}000{,}000$ before being found out, the GBB would only need to be approximately $\$100$.

Sybiling is of no direct benefit in this case. In fact, it makes it even easier to prevent dishonesty. Consider the network above. Suppose that one of the agents decides to create 900 extra nodes so that there were still 100 independent agents, 99 controlling one node each, and one controlling 901 nodes. The using the formula above, a GBB of $10{,}000/1000(99)$, or just over $\$.10$ would be enough to prevent dishonesty. On the other hand, reducing the number of independent agents running nodes would require that the GBB be larger. If, for some reason, there were only 3 independent validators running, say, 50 Sybliled nodes each, then the GBB would need be at least $10{,}000/150(2)$ or about $\$67.00$. If each agent ran only one node, the necessary GBB would go up to $\$1667.00$.

In any event, this leads to our central Theorem:

Theorem 1: *Consider a blockchain with HaS architecture, validated by PoH backed up by the CDM and CRP and assume PNN. If* $G > \frac{V}{N(A-1)}$, *then the blockchain satisfies SPS.*

Two points are worth discussing here.

First, the CDM and CRP are really designed to take care of edge cases that would completely destroy the integrity of blockchains validated by any other protocol in existence. No existing protocol would survive a network which is more than 50% dishonest, while the CDM and CRP allows survival even when 100% of the network is dishonest.

Second, PoH by itself provides validation with 99% BFT. An easy way to guarantee that an honest node will always exist is to make sure that there is one trustworthy agent in each validation network. Doing so would ensure that the edge-case that the CDM and CRP are meant to deal with never occurs. This could be accomplished by having one or several known and trustworthy agents (banks, accounting or law firms, for example) publicly join the validation network. Such nodes would not be privileged and would have no special power compared to other nodes. However, it is extremely unlikely that such nodes would join in any conspiracy to compromise the blockchain. The fact that they are not anonymous and might therefore be subject to pressure from state actors is not of great concern. Even if these trustworthy nodes were forced to behave dishonestly or violate protocol by court order, they would simply be audited out of the validation network with no harm done to the blockchain. Thus, in a practical sense, for a blockchain with a few trustworthy nodes to fail, all of the trustworthy nodes would have to compromised by state actors at the same time that all the rest of the nodes decided to come together as a dishonest coalition of the whole. Even then, we still would have the CDM and CRP to rescue the blockchain.

13. Conclusion

In this paper, we propose a new approach to blockchain validation based on economic mechanism design.

We begin by building a random hub and spoke network architecture that minimizes bandwidth requirements and latency while preserving the anonymous, open membership advantages of more traditional P2P approaches. We avoid central points of failure by keeping the list of validating nodes in each of the distributed ledgers maintained by the network of validators.

Next, we construct a protocol called Proof of Honesty that leverages the auditability of properly constructed blockchains to provide a solution with 99% Byzantine Fault Tolerance. Proof of Honesty returns authority to users who have tokens on the chain instead of allowing nodes, who have the potential to steal tokens and falsify transactions, to be the ultimate arbiters of truth. Users run client software that does continuous due diligence on the blockchain of interest and is equipped with the same protocol software that nodes use. This allows users to verify for themselves whether a node, fork, or chain is honest and then to choose to transact only with honest nodes.

While this is a huge advance over PoW, PoS, DAG and other blockchain protocols that offer BFT of 50% or less, we add an economic mechanism called the Catastrophic Dissent Mechanism

that makes the protocol 100% BFT. The CDM creates a system of self-enforcing, incentive compatible audits. The result is blockchain validation protocol that implements truth-telling and honest chain building in coalition-proof equilibrium. This means that not only are users and their token accounts safe in that even if 100% of the nodes are "dishonest" in the sense used by algorithmic game theorists, but there also is a behavioral foundation and incentive structure that provides Strategically Provable Security.

All the advantages that distributed ledger technology offer depend on honest transaction verification and block-writing by the network of nodes. A BFT of 33%, 50% or even 99% is simply not good enough if blockchains are to be trusted with critical infrastructure, personal and financial data, stock exchanges, public records, and so on. The more important the data on a blockchain is, the more worthwhile it is to invest computational and other resources to compromise it. It may not even be important to the attacker that he be able to take over a blockchain and steal tokens or the data it contains. It may be reward enough simply to be able to disrupt the economy and create chaos.

Our hope is that the protocol developed here provides a level of security on a trustless decentralized platform that makes it possible for blockchain to live up to its enormous and transformative potential.

Appendix 1: Proofs

Claim 1: *Assume that all nodes in the block B-1 ANL have identical CLSs. Then assuming PNN, either all the nodes will produce identical PTBs and FVBs and end up with identical CLSs for block B, or it will be provable that a node failed to follow protocol.*

Proof: We prove this claim in four steps.

First, we show that any failure of nodes or hubs to transmit messages as protocol requires can be proven. Messages are sent at three points in the HaS work-flow.

1. Users send UUTs to nodes.
2. Nodes send NTBs to the hub.
3. Hubs send HTB to all nodes.

- PNN directly implies that if a node fails to include a UUT in its NTB or send an NTB to the hub, the user or hub, respectively, will be able to prove it. In addition, if the hub fails to include the NTB sent from any node in the HTB or send the HTB to any node, the node in question will be able to prove it. Finally, no node that receives an HTB from the hub will be able to claim otherwise since the hub can prove the message was sent.

Second, we show that node and hub messages must be correctly signed.

- Suppose some node $n \in \mathcal{N}$ sends an NTB to the current hub that is not properly signed. By protocol, the hub should not include this in the HTB it sends to all the nodes. If it does, it can be detected by all nodes who can then prove the hub behaved dishonestly. If the hub

does not include the improper NTB, Node n will only be able to prove that it sent an improperly signed NTB and this will prove that node n is behaving dishonestly.

- Suppose that hub for block B sends an HTB to the nodes that is not properly signed. Then as above, the nodes can detect and prove this. Honest nodes will submit audit transactions. If, despite this, any node includes the improper HTB in the PTB or FVB for block B, it will be visible proof that both the hub that failed to sign the HTB correctly and the node that used it anyway are dishonest.

- As a corollary, the HTBs the current hub sends to the nodes in the ANL must be identical. We know from the arguments above that the hub must send an HTB to each node and that it must include a properly singed NTB from each node. If each node only sends one signed NTB (as protocol requires) then this must be included in the HTB and so there is only one HTB the hub can send. If the node is dishonest and sends two signed NTBs to the hub, the hub could dishonestly create two HTBs using the two NTBs and so send different HTBs to different nodes in the ANL. But then nodes will end up with different FVB and CLSs. Hashes of these CLSs will be included in the next NTB sent to the next hub. Thus, nodes looking at the next HTB and users looking at the blockchains and CLSs of different nodes will all be able to detect and prove dishonesty by the nodes and hubs that failed to follow protocol .

Third, we show that UUTs, NTBs, and HTBs must be correctly constructed. Failure to do so is either provable or harmless.

- Suppose some node $n \in \mathcal{N}$ includes a false UUT that it simply made up (and so was not received from a user) in the NTB it sends to the hub. Node n 's NTB would be included in the HTB sent by the hub back to each node. The fictional UUT could not have a correct user signature since only the account owner has the private key associated with the account. Therefore, each node would either have to follow protocol and reject the false UUT in constructing the PTB and FVB or include it anyway. If any node included the UUT as a valid transaction, the falsity of the signature would be immediately detectable and so it would be provable that the node is behaving dishonestly. Thus, while a node could make up UUTs and add them to its NTB, if they made it into a verified block its dishonesty would be provable. On the other hand, if nodes followed protocol and rejected the false UUT, no harm is done.

- Suppose some some node $n \in \mathcal{N}$ sends an NTB to the current hub that is properly signed but improperly constructed. By protocol, the hub should include this in the HTB it sends to all the nodes (and will be provably dishonest if it fails to). The fact that the NTB is improperly constructed can be detected by all nodes who can then prove the node n is behaving dishonestly.

- Suppose that hub for block B sends an HTB to the nodes that is not properly constructed. All nodes can detect and prove this and should launch audits. If, despite this, any node includes it in the FVB for block B, it will be visible proof that both the hub that failed to construct the HTB correctly and the node that used it anyway are dishonest.

It only remains to show that if nodes or hubs fail to follow protocol in processing messages it can be proven. Putting together the three points above, we know all nodes must include all UUTs received from users in a single NTB which is correctly constructed and signed and sent to the correct hub, and that the hub must include the exact NTB received from each node in a correctly constructed and signed HTB, and this same HTB must be sent to all nodes in the block $B-1$ ANL. By assumption, all nodes in the block $B-1$ ANL start with the same CLS, and from the argument above receive the same HTB which must be included in the block B FVB it commits to the chain. Failure to do so is detectable and provable by PNN. As a result, if a node fails to follow protocol in correctly updating the CLS with the common HTB, it will end up with a block B CLS that is different and provably incorrect from the ones derived by honest nodes.

Thus, all nodes that start with the same block $B-1$ CLS must end up with the same block B CLS or be provably dishonest. ■

Claim 2: *Assuming PNN, all the nodes will produce identical PTBs and FVBs and CLSs for all blocks or it will be provable that a node failed to follow protocol.*

Proof: All chains begin at a common genesis block (block 0) which has an initial ANL and CLS which is correct and shared by all nodes by definition. By Claim 1, if all nodes in the ANL of block $B-1$ agree on the CLS, all the nodes will produce identical PTBs and FVBs and end up with identical CLSs for block B or it will be provable that a node failed to follow protocol. Therefore, by induction all the nodes will produce identical PTBs and FVBs and CLSs for all blocks or it will be provable that a node failed to follow protocol. ■

Claim 3: *Rational honest users will always choose to transact on an honest fork if one exists.*

Proof: First, note that the tokens can never be stolen on honest forks. Regardless of how many tokens are stolen or improperly added to accounts on dishonest forks, none of these dishonest transactions ever appear on honest forks. Thus, all agents, honest or not, always have the correct balances on honest forks.

Suppose that a rational agent were offered a choice between a transfer of tokens written to an account on an honest or dishonest fork of the chain. The agent will realize that there is at least a positive probability that the tokens offered on the dishonest fork are stolen. He will also realize that there is at least a positive probability that if he takes tokens on the dishonest fork, they might be stolen in the future. In addition, by protocol, the dishonest fork is per se invalid. On the other hand, tokens on an honest fork are not stolen, will not be stolen, and are in per se valid accounts. Thus, we claim that under any reasonable set of beliefs, tokens on an honest fork are worth more than tokens on a dishonest one.[21] Accepting tokens on a dishonest fork is risky since it is likely to

21 Properly speaking, this should be an assumption. There are Nash equilibrium in which agents coordinate on ignoring the honest chain and believe that tokens will never be stolen again. Of course, agents might also believe that if they don't cover themselves with tomato sauce immediately, they will be doomed to Pastafarian hell by the Giant Flying Spaghetti Monster (despite his noodley goodness). In this case, however, blockchain validation protocols lose most of their relevance. Put another way, we argue that any beliefs about the relative value of tokens on honest and dishonest forks other than those we outline are unreasonable.

be difficult to convince other users who can also prove that the fork is dishonest to accept them at face value in the future. Rational agents will therefore prefer tokens on honest forks to dishonest forks since they are worth more.

By definition, honest agents would never take tokens on a dishonest fork if they were stolen. On the other hand, if tokens on a dishonest fork happen not to be stolen, they will also exist in the same account on an honest fork. Since tokens on an honest fork are worth more than tokens on a dishonest fork, it is in the interests of both the sending and receiving agents to transact on the honest fork. Thus, honest, rational agents will only transact on honest forks, if they exist. ■

***Claim 4**: Assuming PNN, a blockchain using the HaS architecture outlined in Section 8 verified using PoH is 99% BFT.*

Proof: Suppose that at least one honest node exists in the ANL. By Claim 2, we know that users are able to discover and verify the honesty of the fork these nodes are writing. Then by Claim 3, all honest, rational users send their transactions to one of the nodes constructing the honest fork. But then, all rational, honest users are transacting on a fork where no tokens are stolen and all protocols are followed. ■

Claim 5: *Consider a blockchain with HaS architecture validated by PoH backed up by the CDM and CRP and assume PNN. Suppose that all the nodes join a conspiracy to steal and move off-chain the maximum value they believe to be feasible. (That is, $v=(v_1,\ldots,v_N)$ where $\forall n\in\{1\ldots N\}$, $v_n>0$ and $\sum_n v_n=V$.) Then if $G>\frac{V}{N(A-1)}$, at least one agent running a node will be better off reporting the conspiracy by initiating an audit rather than following through on his commitment to be dishonest.*

Proof: Suppose that there are A independent agents running a total of N nodes in the ANL. Then the average payoff that dishonest colluding agents receive is $\frac{V}{A}$. If any agent gets more than this, another agent must get less. Suppose first that all agents run the same number of nodes: $\frac{N}{A}$. Then if one agent defects from the conspiracy, he collects the GBBs from all other nodes.[22] Since $\frac{N}{A}$. of these are nodes are owned by the defecting agent, the value of the GBBs collected from the nodes belonging to the other agents is $\frac{GN(A-1)}{A}$. Thus, if $\frac{GN(A-1)}{A}>\frac{V}{A}$, the agent is better off defecting. This is true if and only if $G>\frac{V}{N(A-1)}$.

Now suppose that agents run different numbers of nodes. Then some agent must run fewer than $\frac{N}{A}$ nodes. If one of these smaller than average agents (in terms of the number of Sybils he creates

22 If one agent defects from a conspiracy, it does not matter whether one, some, or all of the nodes he runs call for the audit. To see this, suppose only one of his nodes calls for an audit. The agents collects the GBB from the "dishonest" nodes he runs, but this is his own money. The agent could have kept these tokens by simply writing a "resign" ANL UUT. The net audit reward is therefore the GBB's of the nodes of the other agents if they decide to behave dishonestly.

on the network) defects, he collects GBBs from more than $\frac{N(A-1)}{A}$ nodes. In other words, the total reward to defecting is even greater for agents running smaller numbers of nodes and so the agent with the smallest number of nodes will certainly defect if $G>\frac{V}{N(A-1)}$.

Suppose that the dishonest conspiracy tried to head this off by giving these smaller agents a proportionally greater share of V. In particular, if agent i runs N_i nodes $(\sum_i N_i=N)$, then agent i gets an audit payoff of $G(N-N_i)$. Suppose that the dishonest coalition of the whole tries to give each agent this payoff for participating in the conspiracy to steal V. In total, these payoffs would equal

$$\sum_i G(N-N_i)=G(AN-\sum_i N_i)=G(A-1)N.$$

Thus, if the conspiracy were able the share V in this way, all agents would be better off participating and not calling for audits. Unfortunately, this would require that $GN(A-1)N>V$. This in turn would require that $G<\frac{V}{N(A-1)}$, which contradicts the hypothesis of the claim.

Thus, regardless of the pattern of node ownership and the sharing of the proceeds of the theft, there is at least one agent who would be better off reporting the conspiracy if $G>\frac{V}{N(A-1)}$. ■

Theorem 1: *Consider a blockchain with HaS architecture, validated by PoH backed up by the CDM and CRP and assume PNN. If* $G>\frac{V}{N(A-1)}$. *then the blockchain satisfies SPS.*

Proof: Claim 5 demonstrated that, assuming $G>\frac{V}{N(A-1)}$. and PNN, if a blockchain with HaS architecture is validated by PoH backed up by the CDM and CRP, then no conspiracy to steal tokens by a coalition of 100% of the nodes in the validating network is sustainable. Regardless of how stolen tokens are shared, there will always exist a coalition of one or more agents who are better off defecting from the conspiracy and initiating an audit. These honest nodes will therefore continue to validate an honest fork on the blockchain, and honest, rational users will choose to transact only on this honest fork.

Dishonest nodes will end up holding tokens and GBBs on dishonest forks that no honest, rational user would ever choose for transactions. As a result, the dishonest fork will eventually be orphaned. Collectively, the dishonest nodes lose GBBs worth DG as a result. It is possible, however, that $DG<V$ and so the dishonest nodes may, in fact, profit. This would require that a gullible agent or agents make the real-world transfers without waiting for their user clients to confirm that their transactions went through on an honest chain (or that the gullible agents simply ignored their user clients). While this does, in fact, mean the gullible agents are robbed, it is a real-world robbery, not a theft of tokens. Moreover, had the gullible agents taken precautions that the PoH protocol requires, even this robbery would not have been possible.

We conclude that the CDM guarantees that an honest chain will always exist and, given the audit rewards, that the CRP will never need to be invoked. All nodes know that any dishonest behavior

will be reported by at least one other node, which will result in the confiscation of their GBBs. This means that there is no possibility of holding on to tokens stolen as a result of any type of conspiracy. In practice, this should make it unprofitable for any node to violate protocol and try to steal tokens. The presence of an honest chain should make it extremely difficult to move value off-chain (that is, V should be small). Thus, it is likely to be a best response for all nodes to behave honestly. However, it is possible that V is large enough to make some nodes choose to try to move V off-chain and sacrifice their GBBs. Moving sufficient value offline requires that enough gullible users do not follow protocol and behave contrary to their own interests. Regardless, the only coalition proof equilibrium results in the existence of an honest chain and no tokens being stolen from rational, honest users. Therefore, the blockchain satisfies SPS. ■

Appendix 2: Hashing, Encryption, and Blockchain

In this appendix we give a very brief explanation of some of the key cryptographic elements of blockchain.

Hashing

Hashing is a method used to verify the integrity of a message or file. The **Hash Algorithm** itself is public knowledge and does not need any kind of encryption key to work. When a message or file is run through a hash algorithm it produces a fixed-length output string. The Secure Hash Algorithm-256 (SHA-256) is one widely used approach and has the following properties:

- Running any file, regardless of length, through SHA-256 returns a 256 bit binary string.
- Very similar files produce quite different hashes in an unpredictable way. In fact, SHA-256 is designed so that hashes of different files are, in effect, randomly and uniformly distributed[23] over the set of all possible 256 binary strings of which there are 2^{256} or about 10^{77}.
- The hash of any file is unique. The same input always produces the same output. For this reason, the hash is sometimes referred to as a file's "fingerprint".
- Although hashing a file always gives the same result, two files may have the same hash. This is called a "collision". In practice, however, collisions are extremely unlikely to occur.[24]
- Hash algorithms are noninvertible. It is impossible to recover a file from its hash.

23 Okay, not randomly; hash functions are deterministic. However, if you took a file, modified it one bit at a time, and then looked at the distribution of the resulting hashes, they would be approximately uniformly distributed between $0\ldots0$ and $1\ldots1$, where these are each 256 bit long binary strings.

24 Collisions must occur in theory. A typical MP3 file is tens of millions of bits long. Obviously, it is impossible to do a one-to-one mapping between the set of every possible 10M bit string and every possible 256 bit string. However, if I took a hash of any file and then started looking for another file with the same hash, I could test $10^{77}/2$ files and still have only a 50% chance of finding one. Thus, it is extremely unlikely that you will ever actually see a collision.

One of the most important uses of hashing is proving that documents and other digital objects have not been altered. Suppose you sign a partnership agreement and later on have a dispute. You and your partner both present copies of the contract to a judge, but they say different things. How can the judge determine which is genuine? Suppose the judge had access to a hash of the contract as it was on the day it was signed. He could then compare it to hashes of the contracts presented to him by you and your partner. The genuine contract will produce the same hash, and any altered contract will produce something else. You may ask how the judge might have access to a hash of the contract he knows was taken on the day it was signed. Read on about digital signatures and blockchain below.

Encryption

Symmetric Encryption

At the most basic level, encryption subjects a **plaintext** file to a series of substitutions and permutations determined by an encryption key to produce **ciphertext**. An **encryption key** is simply a string of bits of specified length. For example, **AES 256 (Advanced Encryption Standard 256)** uses a 256 bit key which means that there are $1.2x10^{77}$ different ways that the substitutions and permutations might be executed on plaintext to produce ciphertext. AES-256 is seen as a completely secure encryption system given current computer technology.[25] AES-256 also has the advantage of being computationally efficient, meaning that relatively few computer clock cycles are needed to decrypt each byte of the ciphertext if the key is known.

Although AES is a secure system, it depends on the sender and receiver sharing knowledge of a common key (sometimes called a "shared secret"). This is called **private key encryption** and is a form of **symmetric key cryptography**. But how can users agree on a key while keeping it secret from everyone else? Obviously, sending the key in an unencrypted form exposes it to interception. On the other hand, the sender cannot send the key to the receiver in an encrypted form unless the receiver has the key needed to decrypt encrypted key. We have a chicken and egg problem.

Public Private Keys (PKK)

Where it is impractical for users to meet securely and agree upon a shared secret, **public private key (PPK) encryption,** which is a form of **asymmetric key cryptography,** can be used instead. The drawback is that decrypting text without a shared key takes more computational effort. As a result, public key encryption is often used only to begin a secure communication session in order to send a symmetric encryption key to be used by both sides for the remainder of session.

---

25 The only way to decrypt a file is to know the right key. Breaking encryption therefore requires that the key be discovered by brute-force guessing. Suppose that a hacker had enough computational power to make 10^{12} (one quadrillion) guesses per second. This means that the hacker could make about 3×10^{19} guesses per year. To have a 10% chance of guessing which of the 1.2×10^{77} possible keys was correct, the hacker would have to test 1.2×10^{76} keys. This would take approximately 4×10^{56} years. As far as we know, no one has this much computational power. In any event, the universe is only about 13.7×10^{9} years old. Thus, while it is not impossible to break AES encryption in theory, it would take more computing power than is likely to exist in the foreseeable future and require many orders of magnitude longer than the universe has existed to do do. This is why the National Institute of Standards and Technology (NIST) says that it is computationally impractical to break AES-256 encryption, although this may change as new quantum computing based decryption techniques are developed.

The real magic of public private key encryption is that the public and private keys have a special mathematical relationship. Not only is the private key the one and only way to decrypt a message encrypted with its complementary public key, but the *public key is the one and only way to decrypt message encrypted by the complementary private key*. This symmetry turns out to be essential to blockchain, SSL certificates, and many other applications.

At the highest level, public key encryption for secure communication works like this:

1. The receiver generates two large cryptographically entangled numbers. One is called a **Public Key** and is made openly available. The other is called a **Private Key** and is kept secret by the receiver.

2. The sender uses the receiver's public key to encrypt his message.

3. The receiver uses his private key to decrypt the message.

Cryptographic Signatures

Signing paper documents is the traditional way of indicating agreement or acknowledging receipt. Signatures can be forged, so banks and notary publics require that people show identification documents such as passports or driver's licenses. These documents often include photographs, signatures, physical descriptions, and even fingerprints. In effect, these documents are an **attestation** by a government agency or whoever issued the documents that it believes that the person named on the ID document is the same one in the photograph, uses a signature that looks like the one on the document, has a certain fingerprint, and so on.

There are several weaknesses to this approach. First, we have to trust in the truthfulness and due diligence of the document issuer if we are to believe its attestation. This is why banks ask for official government IDs instead of your work ID or AAA card. Second, the document could be forged. Third, the ID could be stolen. These last two problems might be solved if we could request a copy or image of the document from the issuing agency. This would make it immediately apparent if the document being presented is forged, altered, stolen, or revoked.

In the digital world, public private key (PPK) pairs combined with hashes can be used to sign documents, messages, transactions, and any other type of digital object. Signing and verifying signatures works as follows:

1. The signer produces a hash of the document.

2. The signer encrypts the hash with his *private key*.

3. The signer attaches this encrypted hash to the unencrypted (cleartext) document

4. To verify the signature, the reader decrypts the hash using the signer's *public* key. The decrypted hash could only have been encrypted in this exact way by the holder of the complementary private key (that is, the signer). Thus, the reader knows that this is the correct hash of the document as it was signed by the private key holder.

5. Finally, the reader produces his own hash of the unencrypted document. If the hashes match, then he knows the document is exactly what was signed and that it has not been altered in any way.

For this to work, the reader must have access to the public key that can decrypt the hash and believe that this public key belongs to a specific person or entity. The reader also has to believe that the owner of the public key had control of the corresponding private key when the document was signed. If a hacker managed to get his hands on the private key, he could use it to sign documents just as easily as the true owner. Thus, if we think the key has not been stolen or compromised, and we are confident that we know the real world person or entity who owns and controls the key, then we can be equally confident that that person or entity signed the document verified by matching hashes.

Blockchain

PPK Account Addresses

PPK pairs are at the heart of blockchain technology. Accounts kept on the ledger effectively belong to a PPK pair. Accounts are numbered or at least include a public key to indicate ownership. There is no record within the ledger of what human or other entity is associated with the account. Transactions from an account require a transaction request signed by the corresponding private key. Nodes use the public key in the account address to attempt to decrypt the transaction request, and if they are successful, they know that the requester has access to required private key. If an account holder loses his private key, it is impossible to access his account. The tokens are frozen forever and are effectively removed from the coinbase. If another agent gains access to the private key, he can use it to steal the tokens. From the standpoint of the validating nodes, anyone who can produce the private key owns the tokens in the account.

Merkle Trees

Blocks are chained together using a kind of recursive hashing technique called a Merkle tree. The idea is ingenious, but very straight-forward. Blockchains begin with a genesis block (block 0) that contains a ledger of accounts defining the initial distribution of tokens. The first block could be constructed in any number of ways depending on the protocols the blockchain uses. However it is constructed, a set of valid transaction requests are grouped together as block 1 to be appended to the genesis block, block 0. This is done by taking a hash of block 0 and including it in block 1. Block 2 is then built and a hash of block 1 included. In general, block B includes a hash of block $B-1$.

Now suppose that a chain has 1000 blocks and I want to go back and alter a transaction in block 600. Obviously, this changes the data in block 600 and so also changes the hash of block 600. But the hash of block 600 is included in block 601. Anyone who wanted to could take the hash of the altered block 600, compare it to the hash included in block 601, and thereby prove that the data in block 600 had been changed. This is called “failing the Merkle proof”. The only fix would be to include the new hash of block 600 in block 601. But then the hash of block 601 contained in block 602 would be wrong. Thus, for the blockchain not to fail the Merkle proof, all the hashes from block 600 to block 1000 would have to be recalculated.

What this buys us is two things. First, any attempt to alter data in the a blockchain is visible and provable since the Merkle proof would fail. Second, it makes the blockchain "append only". That is, it is impossible to insert blocks or data into the middle of a blockchain without causing the Merkle proof to fail. Data can only be append the last block.

Both of these features depend on the knowledge that the Merkle tree has not been recalculated after data has been altered. If someone takes the trouble to recalculate all the hashes from block 600 onward, for example, he could erase evidence of his tampering. How can such thing be prevented?

Proof of Work

The cleverest thing in Nakamoto's Bitcoin protocol is the cryptographic puzzle he developed which is the foundation of Proof of Work. The idea is the following:

Suppose we took a hash of a block we wished to append to a chain. Recall that a hash is a 256 bit binary string that is effectively random. Thus, the probability that the first digits in the hash of any object is 0 (instead of 1) is 50%. The odds that the first two digits are 0 is 25%. In general, the odds that the first n digits of hash are 0 is $\frac{1}{2^n}$. For example, the odds that a random data object would have a hash of 00000??????... (five leading zeros) is one in 32.

Now suppose I took my block and I added some random data to it called a "nonce" and took the hash again. The hash of the block with the new data would also be a random 256 bit binary number. If I did this often enough, I would eventually find a hash that had three leading zeros. In fact, since the odds of this are one in eight, if I added random data four times and took the hash, I would have a 50% chance of finding a hash with three leading zeros.

In the case of Bitcoin, nodes engage in a "guess and check" procedure in which they add random data, take the hash, and see if they have found a nonce that generates a hash with n leading zeros.[26] For example, the odds of guessing at the nonce needed to generate a hash with 70 leading zeros is about one in 10^{21}. There is no way to find the nonce without going through repeated guessing, hashing, and checking. In other words, any node that finds the nonce must have expended considerable computational effort. There are no shortcuts.

As you can see, this makes recalculating the Merkle tree very costly. If data is changed, the nonce is no longer valid, and so must be recalculated as well. Thus, to change transactions buried B blocks deep in a PoW blockchain, B new nonces must be found before the Merkle tree can be recalculated and the Merkle proof made correct again.

Appendix 3: Glossary

Active Node List (ANL): A roster of all nodes that are currently part of the validation network which is maintained as an element of every block that is added to the chain. Membership to the validation network is open and agents can request to be added or removed from the ANL by submitting the proper request transaction to an existing node.

26 The actual Nakamoto PoW system is a bit more complicated, but this is the essential idea.

Byzantine Fault Tolerance (BFT): In algorithmic game theory, the BFT of a protocol or system is the percentage of nodes or components that can behave dishonestly, contrary to protocol, or simply fail, without compromising the objectives or purpose of the protocol or system.

Committed Block Header (CBH): A header needed to create a FVB that includes the block number (that is, the "height" of the block being added), a hash of the previous block, and a hash of the CLS for the previous block.

Current Ledger State (CLS): A list of all current accounts with corresponding balances. This is updated each time a new block of transactions is added to the chain.

Fork: A fork occurs when different blocks are added by different sets of nodes to a common root chain. After the fork, the two versions of the blockchain have different sets of transactions which may be in conflict. Forks may result from network latency or segmentation, or from strategic or manipulative behavior by dishonest nodes.

Full Verified Block (FVB): The actual block that is added to the chain consisting of the current CBH, PTB, and HTB.

Good Behavior Bond (GBB): Joining the ANL as a validating node requires that the future node post a Good Behavior Bond. The GBB is a certain quantity of the native token which can be confiscated if the node behaves dishonestly and also serves as a reward to incentivize the other nodes to initiate an audit of dishonest nodes.

Hub Transactions Bundle (HTB): A bundle of NTBs that the current hub receives from each node in the ANL intended for the current block that are formatted, signed individually and collectively by the current hub and then transmitted to each node in the ANL.

Node Transactions Bundle (NTB): A bundle of UTTs that have been sent to a node since the last block was committed that are formatted, signed individually and collectively by the node, and then transmitted to the current hub.

Nodes/Miners: Nodes are computers attached to the internet at some IP address that have been set up to run software that contains code to run a given blockchain validation protocol. Although nodes are machines, they are under the control of human agents who may wish to manipulate the network or ledger state. Depending on the specific protocol and the role that nodes play, nodes may be referred to as miners, forgers, stake-holders, delegates, or notaries, among other things.

Orphaning: Forks of blockchains can appear for a variety of reasons, some honest and some not. Only one fork can be definitive and authoritative and so consensus protocols must have mechanisms for choosing between them. Forks that end up not being chosen as authoritative are "orphaned" and all transactions added to a chain after the block where the fork occurred are considered invalid.

Proposed Transactions Block (PTB): In addition to VUTs, transaction fees, audits, and other transactions may also be added to the current block depending on the business logic of the blockchain in question. All of these transactions are ordered and formatted and put together as a PTB.

Sybil: Nodes that are controlled by the same agent. In certain protocols, there may be an advantage for an agent to assume many false identities in order to gain more power or voice in the consensus mechanism that determines which transactions are valid.

Unverified User Transactions (UUT): A properly formatted transactions request created by users and sent to any node in the ANL he chooses.

Verified User Transactions (VUT): The set of UUTs in a given HTB that are valid under protocol (correct signatures, no over or double spending) that will end up being including in the next block added to the chain.

References

John P. Conley, (2017) "Blockchain and the Economics of Crypto-tokens and Initial Coin Offerings", Vanderbilt University Department of Economics Working Papers, VUECON-17-00008. http://www.accessecon.com/Pubs/VUECON/VUECON-17-00008.pdf

Groves, Theodore and Ledyard, John, (1977), "Optimal Allocation of Public Goods: A Solution to the `Free Rider' Problem", *Econometrica*, 45, issue 4, pp. 783-809

Lamport, L., Shostak, R., Pease, M. (1982). "The Byzantine Generals Problem". *ACM Transactions on Programming Languages and Systems*. 4 (3): 387–389.

Nakamoto, Satoshi. (2009). "Bitcoin: A Peer-to-Peer Electronic Cash System". *Manuscript*, http://bitcoin.org/bitcoin.pdf

Vickrey, William (1961). "Counterspeculation, Auctions, and Competitive Sealed Tenders". *The Journal of Finance*. **16** (1), pp. 8–37.