Solidus: An Incentive-compatible Cryptocurrency Based on Permissionless Byzantine Consensus

Ittai Abraham*, Dahlia Malkhi*, Kartik Nayak*†, Ling Ren*‡, Alexander Spiegelman*§

*VMware Research, †UMD, ‡MIT, §Technion

Abstract—The decentralized cryptocurrency Bitcoin has experienced great success but also encountered many challenges. One of the challenges has been the long confirmation time and low transaction throughput. Another challenge is the lack of incentives at certain steps of the protocol, raising concerns for transaction withholding, selfish mining, etc. To address these challenges, we propose Solidus, a decentralized cryptocurrency based on permissionless Byzantine consensus. A core technique in Solidus is to use proof of work for leader election to adapt the Practical Byzantine Fault Tolerance (PBFT) protocol to a permissionless setting. We also design Solidus to be incentive compatible and to mitigate selfish mining. Solidus improves on Bitcoin in confirmation time, and provides safety and liveness assuming Byzantine players and the largest coalition of rational players collectively control less than one-third of the computation power.

I. INTRODUCTION

Bitcoin is the most successful decentralized cryptocurrency to date. Conceptually, what a decentralized cryptocurrency needs is consensus in a *permissionless* setting: Participants should agree on the history of transactions, and anyone on the network can join or leave at any time. Bitcoin achieves permissionless consensus using what's now known as Nakamoto consensus. In Nakamoto consensus, participants accept the longest proof-of-work (PoW) chain as the history of transactions, and also contribute to the longest chain.

While enjoying great success, Bitcoin does have several drawbacks. The most severe one is perhaps its inherently limited throughput and long confirmation time of transactions. For instance, presently a block can be added every ten minutes on average and it is suggested that one waits for the transaction to be six blocks deep. This implies a confirmation time of about an hour. Another drawback of Bitcoin is the lack of incentives at several protocol steps. A rational miner is motivated to withhold multiple types of information from other miners. A miner may increase its revenue by engaging in selfish mining [12], [21], [25], a strategy that selectively hides recent blocks from the network. A rational miner should keep the knowledge of transactions to itself to avoid competition from other miners [3], especially when the fixed mining reward (12.5 BTC at the time of writing) falls off or runs out in the future. Carlsten *et al.* [5] show how incorrect incentives such as reliance on transaction fees (when the fixed mining reward falls off) can cause, among other things, a backlog of transactions whose size grows indefinitely with time.

A number of attempts were made to improve the throughput and confirmation time of Bitcoin. These fall into two broad categories. One category [11], [18], [26] tries to improve Nakamoto consensus. The other category [8], [16], [19] hopes to replace Nakamoto consensus with classical Byzantine consensus. These proposals envision a rolling committee that approves transactions efficiently using Byzantine consensus protocols like the Practical Byzantine Fault Tolerance (PBFT) protocol [6]. This second category presents a new approach to cryptocurrency designs and has potential for significant improvements in performance and scalability. However, this new framework also introduces new challenges, which prior works largely overlooked or even exacerbated.

A central challenge of this framework is to reconcile the *permissioned* nature of Byzantine consensus protocols and the *permissionless* requirement of decentralized cryptocurrencies. Byzantine consensus protocols like PBFT assume a static group of committee members, but to be permissionless and decentralized, committee members must change over time — a step commonly referred to as *reconfiguration*. Before reconfiguration can happen, some mechanism is usually needed to stop the old committee from approving more transactions — a step commonly referred to as *wedging*. However, existing works [8], [16], [19] hardly provide any detail on how wedging and reconfiguration are performed. In particular, it was confirmed in a recent post [15] that ByzCoin [16] can deadlock and permanently lose liveness during reconfiguration. In fact, to the best of our knowledge, prior to this work, there is no clear way to adapt Byzantine consensus protocols to the permissionless setting.

Another major challenge is incentive-compatibility. Previous works [8], [16], [19], [24] assume (sometimes implicitly) that a supermajority of participants are "altruistic", i.e., they faithfully follow the protocol even if they get higher utility by deviating from it. This assumption is hardly justifiable in a cryptocurrency setting in which most participants are *rational* and hope to maximize their own gains. The situation is made worse as these works give participants even larger incentives (than Bitcoin) to deviate from the protocol. For instance, committee members may wish to stay in power forever to keep collecting fees, and miners benefit even more from withholding information.

Contributions. This work, which we call Solidus[1], addresses the above challenges. We were indeed inspired by recent prior

---

[1] Solidus was a gold coin used in the Byzantine Empire.

works, in particular, ByzCoin [16] and Hybrid consensus [24]. In a nutshell, Solidus is a decentralized cryptocurrency based on a novel permissionless Byzantine consensus protocol. We design the Solidus protocol to be incentive compatible, and more formally to satisfy the robust property defined in [2]. We elaborate below.

First, we adapt PBFT protocol to the permissionless setting (Section IV). Importantly, PoW is still at the heart of our consensus protocol, but is used in a novel way. Solidus does not build a totally-ordered PoW chain. Instead, PoW serves as a leader's rank in our protocol, which not only prevents Sybil attacks [10], but also ensures an eventually unique leader to drive consensus. The conceptual difference between Solidus and Bitcoin can be summarized as follows:

Bitcoin uses PoW to build a blockchain, which then provides permissonless consensus;
Solidus uses PoW to solve permissionless consensus, which then produces a blockchain.

As suggested in [16], using (minor variants of) PBFT has potential to improve transaction throughput and scalability of a cryptocurrency. Moreover, as decisions in Byzantine consensus are final once committed, transaction confirmation is much faster than Bitcoin.

Second, we design each component of Solidus to be incentive compatible (Section V). We incentivize committee members to validate transactions and to reconfigure as soon as successors emerge. With these incentives, Solidus provides safety and liveness under the assumption that a supermajority of participants are rational (as opposed to altruistic) and no large coalition exists.

Third, an important consequence of not building a totally-ordered PoW chain is that independent PoWs from different miners can all be accepted without creating forks. This helps mitigate selfish mining and reduce wasted PoW (Section V-D).

Lastly, our techniques for incentivizing information propagation may be of independent interest (Section VI). We reward propagators and describe cryptographic protocols to guarantee propagators get their rewards. As mentioned, Bitcoin or Nakamoto-style cryptocurrencies in general lack incentives for information propagation and may adopt our techniques.

A. Overview of the Solidus Protocol

For ease of exposition, we assume all non-Byzantine participants are "altruistic" in this overview. At a high level, Solidus runs a Byzantine consensus protocol among a set of participants that dynamically change over time. At any moment in time, the current set is called a committee. Once in a while, a new member is elected onto the committee and the oldest committee member leaves. We denote the i-th member in the chronological order as M_i, and the committee size is n. Then the i-th committee $C_i = (M_i, M_{i+1}, M_{i+2}, \cdots, M_{i+n-1})$. The first committee C_1 is known as the Genesis committee, and its n members are hard coded. After that, each new member M_{i+n} $(i \geq 1)$ is elected by the acting committee C_i using permissioned Byzantine consensus.

Let C_i be the current committee without loss of generality. To be elected onto the committee, a miner M needs to present the committee a PoW, i.e., a solution to a hard computational puzzle (more details on the puzzle in Section IV-B). M then acts as the *proposer* (or leader) in a traditional permissioned Byzantine consensus protocol. Through a multi-phase consensus protocol, M proposes: 1) M itself be elected as the next committee member 2) a set of transactions be committed. Transaction format does not have to change from Bitcoin, and transactions are propagated through gossip.

Upon receiving the proposal from M, current committee members validate the transactions in it and, if correct, accept/endorse M (through multiple phases). If M gets enough endorsements from current committee members, its proposal is committed: its set of transactions are recorded and the system reconfigures to C_{i+1} with M becoming M_{i+n}. At this point, we say M's proposal becomes a *decision*. The decision, which is also the next puzzle, then needs to be propagated to all miners. Incentivizing propagation is a challenging task, and we present our solution in Section VI.

It is possible that before M can drive consensus on its proposal, another miner M' also finds a PoW. How we handle this type of leader contention is one of our key differentiators from prior works. Among prior works, PeerCensus [8] did not mention this case, ByzCoin [16] hopes it never happens and may deadlock in case it does [15], and Hybrid consensus [24] relies on Nakamoto consensus to resolve it. Solidus resolves this contention through a Paxos-style leader election [17] using leaders' PoW solutions and auxiliary "epoch numbers" as leaders' ranks. Only a higher ranked leader M' can interrupt lower ranked ones, and to ensure safety M' may have to honor the proposal of the highest ranked leader so far, if there exists one.

One advantage of our approach over Nakamoto consensus is the elimination of forks. A first immediate improvement is shorter confirmation time. In Bitcoin or Nakamoto consensus, a decision is considered committed only if it is "buried" sufficiently deep in the chain. This results in the long transaction confirmation time (six blocks) in Bitcoin. In contrast, we use Nakamoto consensus for neither transactions nor committee election. Once a decision is made by a committee, the transactions in it "go through". The decision will not be reversed unless 1/3 of committee members collude to equivocate. Assuming that no coalition controls 1/3 of the total computation power, the probability that the above happens decreases exponentially with the committee size n. The elimination of forks also reduces wasted work. In Bitcoin, if two PoWs (blocks) emerge at approximately the same time, miners will be temporarily divided and any work done on the losing fork is wasted. In contrast, if two PoWs emerge together in Solidus, a choice will be made by the committee through Byzantine consensus and honored by all miners. Since the chain is never forked, no further PoW will be attempted on top of the losing PoW. We present additional techniques in Section V-D to ensure that even the losing PoW is not wasted in most cases.

II. RELATED WORK

Satoshi Nakamoto introduced Bitcoin, which achieves consensus in a permissionless setting using PoW and the longest-chain-win rule [20]. Bitcoin was formally analyzed by Garay *et al.* [14] and Pass *et al.* [22]. They show that Bitcoin achieves interesting properties such as chain consistency, chain quality and chain growth. Pass *et al.* [22] also show that the average time to solve a PoW puzzle should be much larger than the network propagation time for the security guarantees of Bitcoin to hold.

Proposals in the Nakamoto framework. Some "altcoins" (e.g., [1]) improve Bitcoin by simply increasing the block creation rate without other major changes to the Bitcoin protocol. As shown by [22] and [9], this approach will quickly hit the barrier of network propagation delay: the average block interval needs to be significantly longer than the propagation delay to avoid frequent forks and wasted PoW. Another two proposals in this category relax the "chain" design and instead utilize other graph structures to allow for faster block creation rate. The GHOST protocol [26], partially adopted by Ethereum [4], asks miners to accept and extend the heaviest sub-tree instead of the longest chain. Lewenberg *et al.* [18] allows blocks to refer to pruned branches to form a directed acyclic graph.

Lastly, the most relevant work to us in this category is Bitcoin-NG [11]. We discuss it together with other related designs below.

Table I reflects this with .

A comparison between Solidus and related designs. Table I compares how Solidus and related designs achieve consensus for transactions and leader/committee elections respectively, whether they provide instantaneous confirmation and whether they mitigate selfish mining. To start, we have Bitcoin that commits blocks using Nakamoto consensus and does not separate leader election and transaction processing.

The key idea of Bitcoin-NG is to decouple transactions and leader election [11]. A miner is elected as a leader if it mines a (key) block in a PoW chain. This miner/leader is then responsible for approving transactions in small batches (microblocks) until a new leader emerges. Since (key) blocks in the PoW chain are small, Bitcoin-NG reduces the likelihood of forks. One can think of each leader to be a single-member committee. Since a single leader cannot be trusted, any transactions it approves still have to be committed using Nakamoto consensus (i.e., buried in the PoW chain). Thus, after decoupling leader election and transactions, Bitcoin-NG still uses Nakamoto consensus for both.

PeerCensus [8] and ByzCoin [16] both envision a rolling committee to approve transactions using PBFT. Moving from Nakamoto to Byzantine consensus for transactions brings the benefit of instantaneous confirmation. The committee is also responsible for reconfiguring itself. PeerCensus did not seem to describe its reconfiguration process whereas ByzCoin employs a one-phase commit for reconfiguration.[2] Additionally, since reconfiguration and transaction processing happen concurrently, some wedging mechanism is needed, but neither works describe the wedging process. Partly citing unclear designs in the above two works, Pass and Shi [24] adopt a hybrid approach. They rely on Nakamoto consensus (a PoW chain) to elect rolling committees which then process transactions efficiently using Byzantine consensus. This hybrid approach does circumvent the difficulty of reconfiguration in the altruistic model.

Solidus achieves permissionless consensus solely using Byzantine consensus (augmented by PoW). We recombine committee election and transaction processing in Solidus. This effectively solves the wedging challenge even with rational participants. We remark that we could keep the two separate using additional mechanisms, but doing so does not bring clear benefits in Solidus and only complicates the design.

Another differentiating feature of Solidus is incentive compatibility. PBFT [6] assumes that non-Byzantine participants are completely honest. Previous works [8], [16], [24] inherited this assumption as they adopted PBFT as a black box. Ignoring other problems, it is still not straightforward to extend their protocols to work with a rational supermajority. In this work, we open the PBFT black box and build incentive mechanisms into it. We also identify and design incentives for other important steps in Solidus, most notably for information propagation.

Information propagation. Babaioff *et al.* [3] first proposed incentives for transaction propagation and proposed a reward scheme in Bitcoin. We give a detailed comparison to their work in Section VI-C.

III. MODEL

Network assumptions. We consider a *permissionless* setting in which participants use their public keys as pseudonyms and participants can join or leave the system at any time. Participants are connected to each other in a peer-to-peer network with a small diameter. Our protocol requires participants to solve computational puzzles (i.e., find PoWs). Following Bitcoin and Pass *et al.* [22], we assume the average network communication delay is significantly shorter than the expected time to find PoWs. We assume participants can measure time at the same rate except for very small clock drifts.

Types of players. Players in the system can be either *rational* or *Byzantine*. A rational player maximizes its expected utility, and its utility function is Solidus coins. A Byzantine player has an unknown utility function and thus behaves arbitrarily. For ease of exposition, in Section IV we temporarily consider a third type of players — *altruistic* (or honest) players who follow the protocol regardless of their utilities. We use participants and players interchangeably.

All Byzantine players are controlled by a single adversary and can thus coordinate their actions. Rational players can also

---

[2] This can deadlock as confirmed in a blog post [15]. This happens because of the absence of an eventually stable leader. Another blog post suggests adopting Nakamoto for leader election [13] following Hybrid consensus [24].

Comparison of consensus protocols in Solidus and related designs. Bitcoin and Solidus do not separate committee election and transactions. Other protocols in the table separate the two, which we reflect with two separate columns for committee (leader) election and transactions. We mention partial incentives for reconfiguration in case of Bitcoin (and Bitcoin-NG). This is because, as shown in Carlsten *et al.* [5], when mining reward reduces to only transaction fees, miners are incentivized to not follow the prescribed protocol.

Design	Consensus for		Incentives for	Instantaneous	Mitigate
	Committee election	Transactions	reconfiguration	confirmation	selfish mining
Bitcoin [20]	Nakamoto		Partial	No	No
Bitcoin-NG [11]	Nakamoto	Nakamoto	Partial	No	No
PeerCensus [8]	Unclear	Byzantine	No	Yes	No
ByzCoin [16]	One-phase commit[2]	Byzantine	No	Yes	No
Hybrid consensus [23], [24]	Nakamoto	Byzantine	No	Yes	Yes
Solidus	**Byzantine**		**Yes**	**Yes**	**Yes**

form coalitions but we assume no large coalition exists. More precisely, we require that at any point in time, Byzantine players and the largest coalition of rational players collectively control $\rho < 1/3$ of the network computation power. Additionally, we assume it takes time for rational players to form new coalitions, and it takes time for the adversary to corrupt a rational player to make it Byzantine. This is formalized as a delayed adaptive adversary in Hybrid Consensus [24].

Initial committee. We assume the existence of an Gensis committee C_1 of n players known to all miners. For every committee C_i, we define f_i to be the number of Byzantine players and the size of the largest coalition of rational players on C_i. When clear from the context we sometimes omit the subscript i in f_i. We assume that $n \geq 3f_1 + 1$.

IV. THE BASIC PROTOCOL ASSUMING ALTRUISM

For simplicity of presentation, we start by presenting our protocol in the altruistic case, i.e., this section assumes that all non-Byzantine participants always follow the protocol.

Solidus uses a committee to run a sequence of instances of a permissioned Byzantine agreement protocol to approve and order transactions into a ledger. As mentioned in I-A, for decentralization, we need a mechanism to dynamically reconfigure the committee. We first give some background on Byzantine consensus and the PBFT protocol [6] in Section IV-A, and then explain in Section IV-B how we reconfigure the PBFT committee in a permissionless setting.

A. Background on Byzantine Consensus

Byzantine consensus is a classical problem in distributed computing in which a fixed set of participants, each with an input value, try to decide on one input value, despite some participants being faulty. An algorithm solving Byzantine consensus must guarantee *liveness* and *safety*. Roughly speaking, *liveness* requires that every honest participant eventually decides, and *safety* requires that all honest participants decide on the same value.

Our protocol adopts the PBFT protocol [6]. We describe here an outline of PBFT; for more details please see [6]. PBFT

assumes that the number of Byzantine participants f is less than one third of the total number of participants n, i.e., $n \geq 3f + 1$. (In Section V-D we show how our full protocol guarantees it in the permissionless setting.) For simplicity, we present the protocol assuming $n = 3f + 1$.

At a high level, participants first elect a leader among them (view change). The new leader stops previous leaders and learns what values were previously proposed (wedging). Finally, the new leader proposes a value for the other participants to accept.

Leader election. To ensure liveness, there must be an eventually unique leader. Otherwise, different leaders can interrupt each other from driving consensus. To this end, leaders in PBFT are selected in a round robin manner. Every honest participant P uses time outs to monitor the current leader, and endorses the next leader upon timing out. Since we later change the leader election mechanism, the precise details of PBFT leader election are not very important to this paper.

Wedging. A newly elected leader has to atomically (1) stop previous leaders by making sure a majority of non-faulty participants abandon them, and (2) learn the value proposed by previous leaders (if any). To guarantee safety, if a value may be (or has been) committed, then the new leader must propose the same value.

To this end, the new leader picks a rank r that is higher than the previous leader and sends a new-leader message to all participants. Upon receiving such a message, a participant P verifies that the previous leader has timed out and the sender is the next leader in round robin order, in which case P:

- replies with the value from the highest ranked previous leader that it has accepted, or $\perp$ if no value was accepted.
- promises not to accept any proposals with ranks lower than r.

The leader collects $2f + 1$ replies, and then proposes a *safe* value with a proof attached. Such a proof contains $2f + 1$ participants' replies to the new-leader message. A value is *safe* to propose if it is the same as the value proposed by the highest ranked leader among those accepted by the $2f + 1$

replying participants. If all $2f + 1$ replies say that no proposal was previously accepted, then any value is safe.

Reaching agreement. After completing the wedging phase, the new leader proposes a safe value to the other participants, and they agree on the proposed values via three phases of communication. All messages are signed.

- **Pre-prepare:** The leader sends a pre-prepare message to all other participants, which includes the proposed value, and the leader's rank.
- **Prepare:** Upon receiving a pre-prepare message from the current leader, a participant P validates that the leader did not equivocate, i.e., has not sent him a different value with the same rank before. If the leader did not equivocate, P sends a prepare message, with the proposed value and the rank, to all other participants.
- **Accept:** When a participant P receives $2f + 1$ prepare messages with the same value and rank r, it can be sure that no other participant will accept a different value with the same rank. Now if P has not promised to not accept proposals with rank r, it accepts the proposal, sends an accept message to all other participants, and promises not to accept any proposals with a rank lower than r.
- **Local commit:** Upon receiving $2f + 1$ accept messages for the same value and rank, a participant can safely commit the value.

The above local commit is safe because if $2f+1$ participants have accepted a proposal, then at most $2f$ participants (the remaining f and the f Byzantine) may accept a different one, which does not meet the $2f + 1$ threshold. In fact, the moment that the $(f + 1)$th honest participant accepts a proposal, it becomes *globally committed* since no other proposal can be accepted (even though no participant has locally committed it yet).

Steady state. In PBFT, once a leader is elected it can propose many consecutive proposals and drive consensus on them one by one without performing wedging after every decision. This is called the *steady state* in PBFT, and such a leader is called a *stable* leader. When a new leader is elected, it performs wedging to move the system from the previous steady state to the next one, in which he is the stable leader. We say that every consensus decision fills a *slot*, and a *ledger* (also called *blockchain*) is a sequence of slots filled with consensus decisions.

B. Adapting PBFT to the permissionless setting

We now explain how we adapt PBFT to a permissionless and decentralized environment. We identify three main challenges: how to prevent Sybil attacks, how to wedge and how to reconfigure the committee.

PoW to prevent Sybil attacks. To prevent Sybil attacks, we follow Bitcoin's proof-of-work (PoW) paradigm. In order to join the committee, a new member must show the committee a PoW, i.e., a solution to a hard computational puzzle. In particular, we adopt the conventional puzzle based on a random oracle $\mathcal{H}$. Solutions to this puzzle are in the form of $(\mathsf{pk}, \mathsf{data}, \mathsf{nonce})$ where pk is a miner's public key, data is some arbitrary data (a miner's proposal in our case) and nonce is field for trial-and-error. A solution is valid if $h = \mathcal{H}(\mathsf{puzzle}, \mathsf{pk}, \mathsf{data}, \mathsf{nonce})$ is larger than some threshold. The threshold is also called the *difficulty* and is periodically adjusted based on the total network mining power. Note that Bitcoin and most prior works define a valid solution to be one with h smaller than some threshold. We define them in the opposite way for reasons that will become clear soon.

The second modification we make to PBFT is to abandon its stable leader approach. Such a stable leader can potentially manipulate reconfiguration, for example, by waiting for another malicious miner M' to solve the puzzle and then nominate M onto the committee. This way, malicious miners may gradually take over the committee. Instead, our protocol uses *external* leaders. A leader in our protocol is not a committee member (though it can be a Sybil of a committee member as long as it has a valid PoW). Any miner who solves the puzzle can try to become a leader by showing a valid PoW.

High-level protocol. An illustration of our high-level protocol is in Figure 1. Miners work on puzzles, which are determined by previous consensus decisions (details in Section V-D). After finding a valid solution (PoW), a miner M tries to become a leader and drive consensus on its proposal among current committee members. We allow M to make only a single proposal. The caption of Figure 1 lists all the fields in a proposal. For now, readers can ignore the two optional fields (4) and (5). If the committee members decide on and commit M's proposal, the set of transactions are added to the ledger and M is accepted onto the committee, replacing the oldest member. At this point, M's proposal becomes the latest decision, which serves as a puzzle for subsequent slots.

Combining reconfiguration and transactions. To solve the wedging challenge, we bind transactions to reconfiguration. Transactions cannot be accepted without a reconfiguration event, so there is no need for wedging. (Recall that wedging is needed only if an old committee is making decisions concurrent to a reconfiguration event.)

Using PoW for leader election. By giving all external miners opportunities to become leaders we introduce a new contention problem if two miners find two PoWs approximately at the same time. If not carefully dealt with, this type of contention can lead to deadlocks [15]. To resolve this type of contention, we need a Paxos-style [17] leader election with increasing ranks. When a new leader with a higher rank arrives, committee members abandon the previous leader (with a lower rank) and continues the protocol with the new leader.

The key challenge is to figure out what constitutes a rank in our protocol. Since our protocol is permissionless and allows external leaders, we obviously cannot use PBFT's round robin order for leader election. One naïve way is to let leaders pick their ranks freely. This will not work since two Byzantine (or even rational) leaders may repeatedly pick higher ranks and
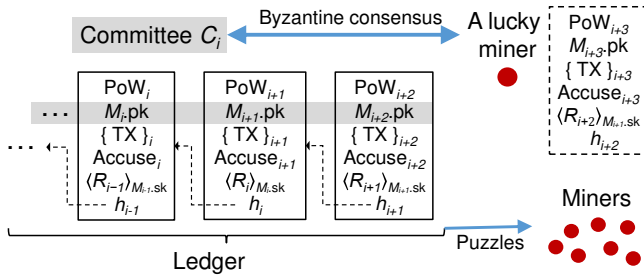
Fig. 1. **Overview of Solidus protocol.** When a miner finds a PoW, it makes a proposal to the current committee members through a Byzantine consensus protocol to be the jth member joining the committee.. Each proposal contains the following fields, (1) a proof of work meeting the required difficulty (PoW), (2) the identity of the lucky miner ($M_{i+3}.\mathsf{pk}_j$), (3) a set of transactions ($\{\mathsf{TX}\}_j$), (4) (optional) accusation of malicious actions against previous members (Accuse_j), (5) (optional) reward division for the previous consensus slot signed by the previous miner ($\langle R_{j-1}\rangle_{M_{j-2}.\mathsf{sk}}$), and (6) a hash digest of the previous block ($)h_{j-1}$). The entire block j is signed by the successful miner M_j (not shown in the figure).

keep interrupting each other. Instead, we need a way to elect an eventually unique leader. In particular, we use h, the hash output of a leader's PoW as part of its rank.

A new "get-epoch" phase. One disadvantage of using PoW as ranks is that a lucky Byzantine leader can stop progress for a long time if it finds an extremely high PoW (no progress will be made until an honest miner finds an even higher PoW). To demote stalling Byzantine leaders we propose a supplemental way to increase ranks. Instead of using just the PoW, the rank is now represented by a tuple consisting of an *epoch* number and a PoW. Ranks are ordered first by epoch numbers, and then by PoW hashes in the case of equal epoch numbers. Every committee member stores a local epoch number e, and increases e after a fixed period of time T. To obtain an epoch number for the rank, a leader P now has to perform an extra *get-epoch* phase at the beginning of the agreement protocol, in which P asks all n members for their local epoch numbers. Then, P waits for $2f + 1$ signed replies and picks the $f + 1$ replies with the highest epoch numbers. The epoch number of P's rank is determined to be the lowest epoch number among the $f + 1$ picked replies. On one hand, by setting T smaller, Byzantine leaders can prevent progress for at most T. On the other hand, T must be large enough such that an honest leader has enough time to drive consensus before an epoch ends. A reasonable value for T may be twice the expected reconfiguration time. In Section V-A we incentivize members to tell the truth about their current local epoch number. We illustrate the communication phases of our consensus protocol in Figure 2.

A subtle point regarding safety. A subtle feature we inherit from PBFT is that a leader cannot always propose its own proposal. Recall that in PBFT there can be a point when a proposal becomes globally committed but has not been locally committed by any committee member yet. Now consider a case in which a leader l_1's proposal has been globally committed,
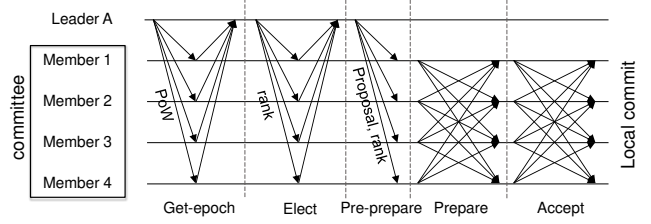

Fig. 2. **Messages in the consensus protocol in the altruistic case.** The last three steps are the same as in PBFT.

but before any member locally commits it, a new leader l_2 arrives with a higher rank. For liveness, committee members must follow l_2. But for safety, l_2 must propose a safe proposal (with a proof attached, cf. Section IV-A), and at this point, only l_1's proposal is safe to propose, which lets l_1 join the committee.

In this section, we assume the new leader l_2 is altruistic. It will re-propose l_1's proposal even though l_2 benefits nothing from it. This is clearly an unreasonable assumption since a rational l_2 may simply walk away and leave the system stalling. In Section V-C, we modify our protocol to create incentives for l_2.

Safety and liveness. The consensus protocol we use is similar to PBFT except for the way we choose a new leader. Thus safety of our protocol follows from safety of PBFT. Our protocol provides liveness by ensuring an eventually unique leader. Note that the higher a PoW is, the harder it is to find a higher one. Eventually some miner will find a high enough PoW (rank) and have sufficient time as a unique leader to drive consensus. More formally, this implements an Ω oracle, which is known to be the weakest oracle needed for solving agreement [7].

Instant confirmation of transactions. As soon as a leader joins a committee, the transactions in its proposal become committed. Given our analysis in Section V-D and our standing assumption that it takes some time for an adversary to corrupt participants, no more than f members of a committee will equivocate. Thus, the transactions are confirmed as soon as consensus is achieved.

V. THE FULL PROTOCOL WITH INCENTIVES

In this section we would like to incentivize current committee members to not only participate in the consensus protocol but also follow the protocol as specified.

Committee members take actions by sending messages to other members or external leaders. We first classify their actions into three broad categories:

1) useful actions, those that follow the protocol,
2) vacuous actions, those that do not violate safety but do not help make progress either, and
3) malicious actions, those that may violate safety if taken by $> f$ members.

There exist many vacuous actions such as sending ill-formatted messages, electing/preparing abandoned low ranked leaders, just to name a few. All of these are equivalent to simply stalling.

Algorithm 1 Protocol for leader P_l.

1: **On** finding proof-of-work PoW **do**
   ▷ Phase 1: get epoch number
2:     Send $\langle$get-epoch, $PoW\rangle$ to all committee members
3:     **wait** for $2f + 1$ $\langle$epoch-reply, $epoch_{num}, \dots\rangle$ replies
4:     $Epoch_{proof} \leftarrow f + 1$ replies with the highest epochs
5:     $epoch \leftarrow$ the lowest epoch in $Epoch_{proof}$
6:     $rank \leftarrow \langle epoch, Epoch_{proof}, PoW\rangle$

   ▷ Phase 2: get elected and wedge
7:     Send $\langle$elect, $rank\rangle$ to all committee members
8:     **wait** for $2f + 1$ $\langle$elect-reply, $last\text{-}accept, \dots\rangle$ replies
9:     let $Prop_{proof}$ be the set of the replies
10:     **if** all replies in $Prop_{proof}$ have $last\text{-}accept = \perp$ **then**
11:         $proposal \leftarrow (\{TX\}, P_l)$
12:     **else**
13:         $proposal \leftarrow$ proposal in $last\text{-}accept$ with the highest rank in $Prop_{proof}$
14:     $proposal \leftarrow (proposal, Prop_{proof})$
15:

   ▷ Phase 3: pre-prepare
16:     Send $\langle$pre-prepare, $proposal, rank\rangle$ to all members
17:     **wait** for $2f + 1$ $\langle$"prepare-reply", $proposal, \dots\rangle$ replies
18:     let $Prepares$ be the set of the replies
19:

   ▷ Phases 4 and 5: prepare and accept
20:     Send $\langle$prepare, $proposal, Prepares, rank\rangle$ to all members
21:     **wait** for a valid $\langle$commit, $Accepts, proposal\rangle$ reply form a committee member   ▷ P_l is a new member
22:     $local\text{-}epoch \leftarrow epoch$ ▷ required for a member

---

Algorithm 2 Protocol for a committee member P_m.

23: **Every** T units of time:
24:     $local\text{-}epoch \leftarrow local\text{-}epoch + 1$

25: **On receive** $\langle$get-epoch, $PoW\rangle$ from a leader P_l **do**
26:     Validate PoW
27:     Send $\langle$epoch-reply, $local\text{-}epoch, P_m\rangle$ to P_l

28: **On receive** $\langle$elect, $rank\rangle$ from a leader P_l **do**
29:     Validate $rank$ ▷ PoW and $Epoch_{proof}$
30:     Let $last\text{-}accept$ denote $\langle proposal, Prop_{proof}, prev\text{-}rank\rangle$ with the highest rank that was previously accepted
31:     Send $\langle$elect-reply, $rank, last\text{-}accept, P_m\rangle$ to P_l

32: **On receive** $\langle$pre-prepare, $proposal, rank\rangle$ from a leader P_l **do**
33:     **if** P_l has sent different proposal with the same PoW before **then**   ▷ PoW is part of the $rank$
34:         Ignore message
35:     **else**
36:         Validate $proposal$ ▷ $Prop_{proof}$
37:         Send $\langle$prepare-reply, $proposal, rank, P_m\rangle$ to P_l

38: **On receive** $\langle$prepare, $proposal, Prepares, rank\rangle$ from a leader P_l **do**
39:     Validate $Prepares$
40:     **if** have not elected a leader with higher rank **then**
41:         send $\langle$accept, $proposal, rank, P_m\rangle$ to all members

42: **On receive** $\langle$accept, $proposal, rank\rangle$ from $2f + 1$ members **do**
43:     let $Accepts$ be the set of the $2f + 1$ recieved messages
44:     locally commit $proposal$
45:     Send $\langle$commit, $Accepts, proposal\rangle$ to the leader whose proposal is committed   ▷ new member

---

We take a two-step approach towards our goal. In Section V-A, we design incentives for useful actions over stalling (vacuous actions), *assuming* rational members do not take malicious actions. We then introduce penalties to deter malicious actions in Section V-B. The pseudocode for our consensus protocol is shown in Algorithms 1 and 2.

A. Incentives for Member Participation

Even after assuming members do not take malicious actions, incentivizing useful actions is particularly challenging. Without specific mechanisms, members actually have incentives to stall reconfiguration in order to stay on the committee and collect more fees. For example, prior works [16], [24] simply divide rewards equally among committee members. Then, every member participating is not an equilibrium. A rational member will stop participating (which means doing no work): he/she gets the same amount of reward regardless. If more and more members stop participating, reconfiguration will not happen any more. Now members suddenly find themselves in an even better (for themselves) equilibrium than before: they stay on the committee forever. This is a serious problem that prior works did not attempt to address.

Our basic idea to address this challenge is to reward the committee members who respond fastest. This creates competition among committee members, forcing them to endorse reconfiguration as soon as possible.

High-level reward structure. The block reward from each committed decision consists of a fixed mining reward and some varying transaction fees. In Solidus, all transaction fees go to the external leader. This ensures that committee members do not have an incentive to delay reconfiguration due to a possible high transaction fee. The fixed mining reward is split between the external leader and participating committee members e.g., half-half (the concrete ratio is not important). The part for committee members is further divided into four shares, one for each phase. Figure 3 shows red bold arrows for messages that get rewarded. The external leader, call it A, is in the best position to divide every share.

Reward for the get-epoch phase. For the get-epoch phase, we want committee members to not just participate, but also reply with their true epoch numbers. In this phase, recall that A picks a set S of $f + 1$ replies, and her epoch number is determined by the lowest local epoch number(s) in S. The reward for this phase is equally divided among these replies with the lowest local epoch number(s) in S. In an equilibrium, every member replies with its actual epoch number, which
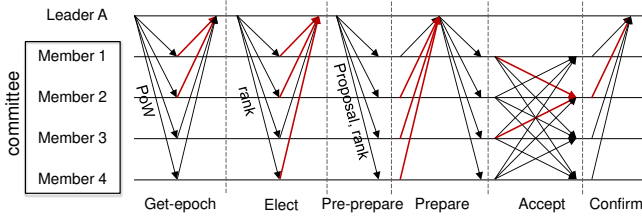
Fig. 3. **Messages in the consensus protocol with reward mechanisms.** The red arrows represent messages that get rewarded. The prepare message now uses all-to-one communication to allow the leader to reward participating members.

ideally should be equal or at least close across committee members. Note that A wants a high epoch number, so she picks the $f + 1$ replies with the highest local epoch numbers. Replies with lower-than-actual epoch numbers will be excluded from S, gaining no reward. Replies with higher-than-actual epoch numbers will get into S, but will not be the lowest in S, again gaining no reward.

Reward for the elect phase. In this phase, A simply broadcasts her PoW and sees which members are fastest to respond (elect her as the leader). The $2f + 1$ fastest to respond divide the reward (details later). In an equilibrium, every member responds as fast as it can. Obviously, delaying responses in this equilibrium only decreases a member's potential gain, which no rational player would do.

Reward for the prepare phase. To accommodate reward division, we slightly modify the prepare phase in PBFT. As shown in Figure 3, we replace the all-to-all communication among committee members with a leader convergecast and broadcast. The situation then becomes the same as in the elect phase. A simply picks and rewards the $2f + 1$ members that are fastest to prepare her proposal.

Reward for the accept phase. For the accept phase, we still keep the all-to-all communication.[3] We let each member sent a commit confirmation to the leader. The confirmation from member i contains $2f + 1$ accept messages including i's own. The leader rewards the $2f + 1$ members involved in the first commit confirmation it receives. Now each member is incentivized to quickly send accept messages to every other member to make sure (increase the chance) that his message gets included no matter which member manages to send the confirmation first.

Other equilibria? The games in the latter three phases (elect, prepare and accept) have another equilibrium where not enough members participate, the protocol stalls (no single member can make the protocol proceed) and no one gets any reward. This should not be a concern for two reasons. First, this stalling

---

[3] This is to make sure that at least two non-colluding players know the puzzle, which is required by our propagation incentives in Section VI. If we were to use the convergecast approach like in the prepare phase, the leader would be the only one to know the puzzle and would have no incentive to propagate.

equilibrium is a weak one, in the sense that members do not lose anything by endorsing reconfiguration. Second, if the system starts in the good equilibrium, it should stay in the equilibrium under our safety condition. No other equilibria exist: If enough members participate, then those who respond too late can simply respond sooner to get rewarded.

Consensus on reward division. So far our protocol gives the leader a natural way to divide the reward, but we still need committee members to agree on the division. This is not straightforward to achieve since it seemingly requires another consensus, which again requires incentives. Our solution is to include the reward division in the next slot. As shown in Figure 1, each proposal/decision includes a reward division for the previous consensus slot, signed by the previous successful miner.

We need to be careful with some additional details to make this idea work. First, a leader M_i may not honor our guideline for reward division. For example, it may reward its allies on the committee even if they are late. We do not and cannot stop this. But other committee members should still respond as fast as they can, to account for the likelihood that M_i (at least partially) rewards its fastest supporters. Second, if M_i for any reason (e.g., it crashed) does not release a reward division plan, the protocol must move on. So we allow M_{i+1} to not include a reward division plan in its proposal, in which case, the reward for slot i is simply evenly divided among all committee members. But now that M_{i+1} has this option, we have to incentivize M_{i+1} to respect M_i's reward division plan if M_i releases one. To this end, we give M_{i+1} a small extra reward for including M_i's reward division.

B. Penalties for Malicious Actions

We list all malicious actions below:

– prepare without $2f + 1$ valid elect messages,
– accept without $2f + 1$ valid prepare messages,
– commit without $2f + 1$ valid accept messages,
– prepare an invalid proposal (e.g., one that contains a double spending transaction),
– prepare an unsafe proposal,
– prepare two proposals with the same PoW/rank,
– accept a proposal after having promised not to.

Note that $2f + 1$ valid elect (as also prepare, accept) must be properly signed by distinct committee members and correspond to the same slot, the same leader rank, and the same proposal.

All the malicious actions above are detectable and non-repudiatable. We design the protocol such that reward earned by a committee member can only be spent a certain number of slots after that member leaves the committee. If a member takes malicious actions, we allow any future leader to include an accusation in its proposal/decision (shown in Figure 1). If correct, such an accusation revokes all reward earned by the misbehaving committee member during its entire service.

Note that at any point, if $< f$ members take malicious actions, the safety of the protocol will not be affected, which means they not only risk losing all their rewards, but their
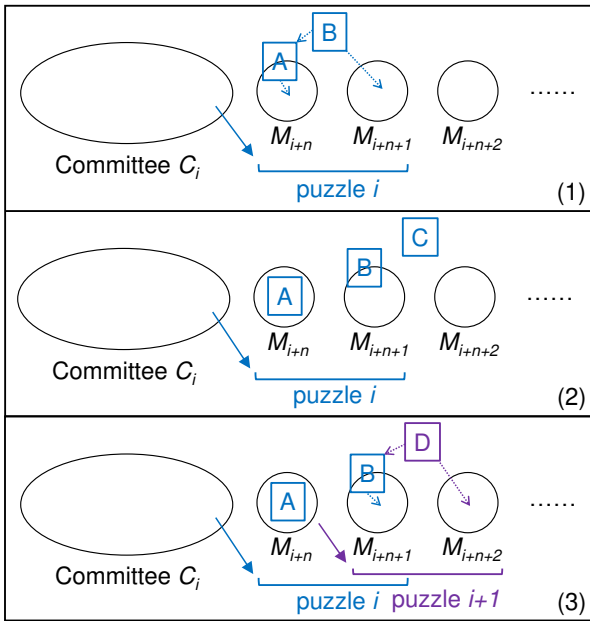
Fig. 4. Overlapping 2 puzzles creates incentives for higher ranked leaders to help a lower ranked one.

intent to double spend (or any other attack) is not going to succeed. This should deter any rational member from taking malicious actions.

A resilient, immune and robust equilibrium. More formally, combining the techniques in this subsection and the previous one, we achieve a *resilient*, *immune* and *robust* equilibrium defined in [2]. The equilibrium is f-resilient: a coalition of f rational players cannot increase their payoff by changing their strategy. The equilibrium is f-immune: if f players behave arbitrarily, the outcome of the protocol stays the same and the rest of the players are not worse off. Combining the two notions, the equilibrium is (k, t)-robust for any $k + t < f$.

C. Incentives for Leaders

There are clear incentives for a leader to drive consensus on its own proposal: it gets a significant portion (e.g., half) of the reward in its proposal and it joins the committee to claim more reward. Stalling is not in the interest of a rational leader.

However, Section IV-B has identified a strong need for incentives when a higher ranked leader interrupts a lower ranked leader whose proposal has already been globally committed. In this case, the higher ranked leader must "help" the lower leader commit while gaining nothing from doing so. In fact, as long as one of the $2f + 1$ replying members has accepted the lower ranked leader's proposal, the higher ranked leader has to respect that proposal. In this subsection we create incentives for higher ranked leaders using *overlapping* puzzles. Concretely, we make each puzzle valid for two slots. Every decision still releases a new puzzle. This means that each slot can now accept solutions from two puzzles.

Figure 4 walks through the possible cases with examples. Without loss of generality, assume the current committee is $C_i = (M_i, M_{i+1}, \cdots, M_{i+n-1})$. The small circles represent slots for future committee members, i.e., $M_{i+n}, M_{i+n+1}, M_{i+n+2}, \cdots$. The last decision by leader (member) M_{i+n-1} generates puzzle i, the blue puzzle in the figure.

Figure 4(1) depicts the case in which a leader A solves puzzle i, manages to globally commit her proposal (pictorially shown as "half in the slot"); but before she can finish, A gets interrupted by B who also solved puzzle i. Now B has an incentive to pick up and finish the work left by A, because B's solution is valid for slot M_{i+n+1} and only after helping A finish can B start driving consensus on his own proposal into slot M_{i+n+1}.

Figure 4(2) shows the case in which B gets interrupted by C after globally committing his proposal to slot M_{i+n+1}, the last valid slot for puzzle i. (B may or may not have helped A.) If C also solved puzzle i, C has no incentive to help B: C's solution is not valid for the next slot (M_{i+n+2}). So a rational C may simply "walk away".

Fortunately, as shown in Figure 4(3), A's proposal must have already been committed into slot M_{i_n} at this point. Its corresponding new puzzle $i + 1$ (the purple puzzle) has been released and propagated to all miners (refer to Section VI). Eventually some miner D will solve puzzle $i + 1$, and D is incentivized to help B finish so that D can take the next slot M_{i+n+2}.

We emphasize again that the reward division plan needs to be signed by the identity inside the decision, as shown in Figure 1. This is because members only reach consensus on the proposal, not on who helped to achieve consensus. For example, consider the case where M_{i+1} helps M_i finish consensus in slot i. M_{i+1} is the one that knows which members participated. M_{i+1} should relay that information to M_i and have M_i produce a signed reward division plan. Finally, M_{i+1} includes it in its own proposal for slot $i + 1$.

D. Mitigating Selfish Mining with a Good Puzzle Span

We say a puzzle valid for k slots has a *puzzle span* of k. Section V-C discussed how to incentivize a higher ranked leader using a puzzle span of 2. In this section, we show how a larger puzzle span can mitigate selfish mining.

A comparison with Nakamoto consensus. Before presenting detailed analysis, we remark that this idea of large puzzle span cannot be applied in the plain Nakamoto consensus. The reason goes back to the fundamental conceptual difference we pointed out at the beginning of the paper. Nakamoto consensus relies on PoWs to order blocks. Each block must have one and only one successor, meaning each puzzle must have one solution.[4] In contrast, Solidus uses permissionless Byzantine consensus to order decisions (blocks). PoWs in Solidus do

---

[4]We note that Fruitchain [23] achieves an effect similar to a large puzzle span, using nontrivial techniques involving two types of blocks.
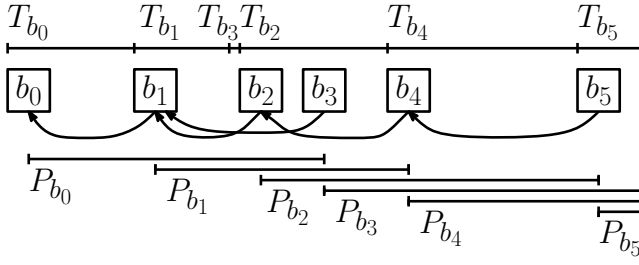
Fig. 5. **An example blockchain in Solidus.** The top portion of the figure shows the times at which blocks were found. The middle portion shows the blocks linking to the puzzles solved by them. The bottom portion shows the slots for which the puzzles are accepted.

not need a total order and thus a puzzle can have multiple solutions (successors).

Example. Figure 5 shows an example of puzzles and solutions with $k = 3$. The lower part of the figure shows the slots at which the solutions of a given puzzle are accepted The middle part of the figure shows the blockchain created and displays the corresponding puzzle of each solution. The ordering among the solutions is determined by the order of reconfiguration events. The upper part of the figure shows a possible timeline for the arrival of solutions. After block b_1 was mined, blocks b_2 and b_3 may have arrived at around the same time. In this example, block b_2 has a better rank and hence it gets accepted first followed by b_3. Block b_4 is a solution to the puzzle released by block b_2 and can still be accepted in one of the $k = 3$ slots after b_2.

Now we analyze in detail how a large puzzle span k mitigates selfish mining. Recall that we assume the communication delay is much shorter than the average time to find a PoW (Section III). This combined with incentives for reconfiguration implies that the reconfiguration time is also much smaller than the average time to find a PoW. [5]

In this section, we use the terms "adversary" and "honest miners", but we do not mean Byzantine and altruistic players here. We still consider Byzantine and rational players. Recall the safety condition of our protocol: the number of Byzantine players plus the largest coalition of rational players on the committee is no more than $n/3$. Therefore, the "adversary" refers to all the Byzantine miners plus the largest coalition of rational miners, who collectively control ρ of the computation power. We need to make sure they do not succeed with selfish mining and do not get more than $n/3$ seats on the committee. "Honest miners" refer all the other (also rational) miners.

Lemma 1. *The probability an adversary occupies k consecutive slots decreases exponentially with k.*

Proof. We prove by induction. In the base case, the genesis puzzle is known to all miners and valid for k slots. The probability that the adversary finds k consecutive solutions

[5]We leave to future work the analysis when the expected reconfiguration time is only slightly shorter than expected time to find a PoW.

for slot 1 to k is ρ^k, exponentially small in k. When the context is clear, we say "negligible probability" for short.

Assuming the lemma holds up to slot j, we now prove it holds up to slot $j + 1$ where j is polynomial in k. We only need to analyze the probability that the adversary occupies the consecutive k slots from $j - k + 2$ to $j + 1$. To occupy these k slots, the adversary must use solutions to puzzles $j - 2k + 2$ to j, because a puzzle is valid for k slots. We consider the first $2k$ solutions that the miners obtained for these blocks. As the reconfiguration time is much smaller than the time to find a solution, the k blocks accepted in consecutive k slots from $j - k + 2$ to $j + 1$ must come from these $2k$ blocks. Thus, the adversary at least needs to obtain k out of these $2k$ solutions. Note that this is a necessary but not sufficient condition for occupying k consecutive slots.

We denote X_i ($1 \le i \le 2k$) to be an indicator random variable where $X_i = 1$ if the adversary obtains the ith solution in these $2k$ possible solutions, and $X_i = 0$ otherwise. Let $X = \sum_{i=1}^{2k} X_i$. We need to find the probability that $X \ge k$.

We only need to consider the case where slot $j - k + 1$ is occupied by an honest miner; otherwise the adversary has k consecutive slots up to j, which happens with negligible probability. Slot $j - k + 1$ thus provides all miners a puzzle to work on up to slot $j + 1$. Given all miners have a puzzle to work on, $E[X_i] = \rho$ and $E[X] = \sum_{i=1}^{2k} E[X_i] = 2k\rho$.

Using a Chernoff bound,

$$Pr[X \ge (1+\delta)2\rho k] \le e^{-[(1+\delta)\ln(1+\delta)-\delta]2\rho k}$$

Choose $\delta = \frac{1}{2\rho} - 1$, we have

$$Pr[X \ge k] \le e^{-[2\rho-1-\ln(2\rho)]k}$$

where ρ is a constant smaller than $1/3$. Thus the probability that an adversary occupies k consecutive blocks decreases exponentially with k. □

How large can k be? While a large puzzle span k mitigates selfish mining, k cannot be too large. Otherwise, it is vulnerable to a *rushing attack*. By a rushing attack, we mean an adversary withholds mined blocks and "rushes" them into consecutive slots in an attempt to take over the committee. Let us examine an extreme case where $k \gg n$ or even $k = \infty$ (meaning there is a fixed puzzle that never changes). The adversary can accumulate n solutions and then present them to the committee one after another. With high probability it may completely take control over a committee of size n, violating the safety condition $f < n/3$.

We will now find safe values of k relative to the committee size n and the adversary's computation power ρ. A value of k is safe if the probability that an adversary has more than $n/3$ blocks in any consecutive n slots is negligible.

Lemma 2. *The probability that there are more than $n/3$ byzantine players in a committee through a rushing attack is bounded by $e^{-(3\rho(n+k)-n(1-\ln(\frac{n}{3\rho(n+k)})))}$.*

Proof. Consider any such consecutive set of n slots. We need to determine the number of solutions that the adversary can obtain in these n positions. Since a puzzle is valid for k slots, the adversary can withhold solutions to the k puzzles generated before the first slot. Let X be a random variable that denotes the number of solutions produced by the adversary in the $n+k$ solutions found by all players in this time span. Again, assuming time for reconfiguration is much shorter than the average time to find a solution, the $n+k$ solutions would be the next $n+k$ solutions accepted into the chain. Thus, the adversary can have a maximum of X solutions in the n possible slots. Using Lemma 1, a puzzle is always available to all miners except with probability exponentially small in k. Assuming this bad case does not happen, $E[X] = \rho(n+k)$.

We want to find the probability that $X > n/3$. By a Chernoff bound,

$$Pr[X > (1+\delta)\rho(n+k)] < e^{[\delta-(1+\delta)\ln(1+\delta)]\rho(n+k)}$$

Choose $\delta = \frac{n}{3\rho(n+k)} - 1$, and let $c = n/k$, we have

$$Pr[X > n/3] < e^{-k(3\rho(c+1)-c(1-\ln(\frac{c}{3\rho(c+1)})))}$$

For a fixed ρ, by setting c to an appropriate value > 1, the probability of a successful rushing attack can be made negligible. $\square$

In theory, one can set $k = O(\lambda)$ where λ is a security parameter, and then choose committee size n to be large enough such that a rushing attack succeeds with negligible probability. In practice, one would fix the success probability of rushing attacks to a small value and then find an appropriate value for c to get k.

Mitigating selfish mining. Recall that selfish mining [12] in Bitcoin works by hiding the latest block, which is also the latest puzzle, from the network. At a high level, a large puzzle span k mitigates selfish mining because there are k puzzles to work on for each slot. Thus, to carry out selfish mining with a k puzzle span, an adversary needs to be k solutions ahead of the network. As shown in Lemma 1, the probability that an adversary occupies any k consecutive slots decreases exponentially with k. If the puzzle span k is $O(\lambda)$, where λ is the security parameter, this will happen with negligible probability.

Reduce wasted PoW. For a solution to be wasted, it is necessary for all miners to collectively submit more than k solutions in a short span of time. Note that the probability for the adversary to submit most of them in this time is very small due to Lemma 1. Also given that the network propagation time (and hence, reconfiguration time) is much smaller than the average time to solve a puzzle, the probability for a block to not be accepted is extremely small.

Producer (P)   puzzle

Alice (A) $\text{puzzle}_{P \to A} :=$ puzzle A.pk x *Signed by P.sk*

Bob (B) $\text{puzzle}_{P \to A \to B} :=$ $\text{puzzle}_{P \to A}$ B.pk x' *Signed by A.sk*
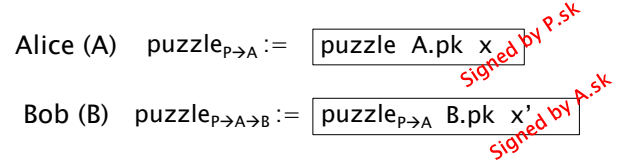
Fig. 6. **Miners propagate the puzzle in a chain of signatures.** Each link is signed by the sender, and contains the receiver's public key and a propagation fee charged by the sender. The sender in each link must be the receiver from the previous link, or the producer of the puzzle if it is the first link. In this example, if B solves the puzzle, P is paid x, A is paid x'.

VI. INFORMATION PROPAGATION

In our basic protocol, there are two places where rational miners are incentivized to withhold information. First, when a miner Alice learns a puzzle, she has no incentive to propagate the puzzle to other miners. Doing so only introduces competition and decreases her own chance of winning (solving the puzzle first). Second, once Alice learns a fee-carrying transaction, it is in her best interest to keep this transaction secret to herself. This way, she is the only miner who can claim the fee of this transaction (when she finally succeeds in solving a puzzle).

Both issues exist in Bitcoin. For transactions, the situation is exactly the same in Bitcoin, and it has been noted by Babaioff *et al.* [3]. For puzzles, the issue is less obvious and less serious in Bitcoin. When a miner Bob finds a latest block and sends it to his neighbor Alice, Alice has nothing to gain by further propagating Bob's block. On the contrary, she has a small incentive *not* to propagate this block. This latest block at this moment is the latest puzzle, and Alice is among the few miners who have learned it. This gives Alice an advantage over most miners for a short period of time (until a competing block is found). This issue is related to but different from the selfish mining attack [12]. In selfish mining, it is the successful miner that tries to hide his/her latest block, while here it is its neighbors that lack incentives to help propagate its latest block.

At a high level, puzzle propagation and transaction propagation are the same problem. In both cases, every miner who learns the information has a chance to win a lottery, i.e., by solving the puzzle and committing the transaction. This model may have applications outside the cryptocurrency setting [3].

A. Our Solution: Signed Propagation Chain

We use puzzle propagation as an example to describe our scheme. Obviously, we need to reward Alice for propagating the puzzle. If Bob learns the puzzle from Alice and Bob succeeds in solving the puzzle, Alice should be paid. The first task is to ensure that Bob cannot deny the fact that he learned the puzzle from Alice. Here it is insufficient for Alice sign the puzzle, because once Bob learns the puzzle, he can easily create a Sybil identity, call it Robert, to sign the puzzle and claims the puzzle came from Robert. We also cannot encrypt the puzzle

because we need to protect the receiver as well: Bob needs to be able to verify the authenticity of the puzzle, i.e., it carries $2f + 1$ signatures from distinct committee members, or the fee-carrying transaction, i.e., it carries the spender's signature.

Our solution, shown in Figure 6 is to keep a chain of signatures, where each sender must be the legitimate receiver from the previous link. In the beginning, the puzzle is known only to its producer(s). In our example, any member on the committee that generated this puzzle is a producer. In the case of a fee-carrying transaction, the spender is the producer. Let P be the producer, A be Alice and B be Bob. We use A.pk and A.sk to denote A's public key and secret key respectively (similar for P and B).

In order for P to propagate the puzzle to A, P first asks for A's public key. P then signs the puzzle together with A's public key A.pk and a propagation charge x. Lastly P sends A the following signed message

$$\text{puzzle}_{P \to A} = \langle \text{puzzle}, \text{A.pk}, x \rangle_{\text{P.sk}}$$

A is only allowed to work on this signed puzzle from P, but not puzzle itself. Also note that this puzzle is signed specifically for A (it contains A.pk), and no other miner is allowed work on it.

Without loss of generality, we assume the reward for solving the puzzle is 1. If A solves this puzzle and gets elected onto the committee, it is clear that she learned the puzzle from P. In this case, P gets x and A gets the remaining $1 - x$ of the reward. It is possible that A receives the puzzle (signed for her) from multiple legitimate producers. In that case, she is free to pick any, and should just pick the one with the lowest charge.

We allow A to further propagate and endorse the puzzle to B. To do so, A adds her own signature layer to what she got from P ($\text{puzzle}_{P \to A}$) together with B's public key B.pk and an amount x' she wishes to charge B, i.e.,

$$\text{puzzle}_{P \to A \to B} = \langle \text{puzzle}_{P \to A}, \text{B.pk}, x' \rangle_{\text{A.sk}}.$$

B and only B is allowed to work on this doubly signed puzzle. Now if B solves this puzzle, it is clear that B learned the puzzle from A, who in turn learned from P. In this case, P gets x, A gets x' and B gets $1 - x - x'$. Note that A cannot omit P's signature. The puzzle is valid only if it originates from a producer and the sender (propagator) at each link is the receiver of the previous link. Of course, B may also get his puzzle from multiple valid paths — they may have originated from different producers and/or have gone through different intermediate hops — and B should just pick the one with the lowest total fee. Similarly, B can further propagate the puzzle to Carol, denoted by C, by extending the signature chain, i.e., $\langle \text{puzzle}_{P \to A \to B}, \text{C.pk}, x'' \rangle_{\text{B.sk}}$.

In summary, a miner can work on the original puzzle if and only if he/she is a producer of that puzzle. A non-producer miner must work on a puzzle with an endorsement (possibly through multiple hops) specifically for him/her. His/her solution on any other version of the puzzle (original, improperly chained,

or endorsed for someone else) will be considered invalid by the committee.

B. Game Theoretic Analysis

We show that with our protocol and a reasonably well connected network of miners, a miner's best strategy is to propagate the puzzle and charge a small fee.

Game 1. To give some intuition, we start with an over-simplified model and focus on only three miners. P_1 and P_2 are two producers of a puzzle, and B is a non-producer miner who has yet to learn the puzzle. P_1 and P_2 are non-cooperative. Each of them can choose to either propagate the puzzle to B for a fee, or not to propagate. For now, we assume propagating the puzzle has no cost, and that every miner will work on the puzzle upon receiving it.

Let λ_1, λ_2 and λ_3 be the computational power of P_1, P_2 and B, respectively. Let λ_0 be the combined computational power of all the other miners who know the puzzle (excluding P_1, P_2 and B). A miner's chance of winning the race is proportional to its computational power. The payoff of P_1 and P_2 in this game look like the following:

- **Case 1:** If P_1 and P_2 both choose not to propagate, then P_1 gets an expected earning of $\frac{\lambda_1}{\lambda_0 + \lambda_1 + \lambda_2}$, and P_2 gets an expected earning of $\frac{\lambda_2}{\lambda_0 + \lambda_1 + \lambda_2}$.
- **Case 2:** If P_1 propagates to B and charges x_1 while P_2 propagates and charges $x_2 > x_1$ or P_2 does not propagate, then B adopts P_1's puzzle and pays x_1 to P_1 upon winning the race. In this case, P_1 gets an expected earning of $\frac{\lambda_1 + x_1 \lambda_3}{\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3}$, and P_2 gets an expected earning of $\frac{\lambda_2}{\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3}$.
- **Case 3:** If P_2 propagates with a lower fee than P_1 (or P_1 does not propagate), the situation is similar to the previous case with B adopting P_2's puzzle.
- **Case 4:** If P_1 and P_2 both propagate and charge the same fee $x_1 = x_2 = x > 0$, we assume B picks P_i's puzzle with probability q_i ($q_1 + q_2 = 1$, $i = 1, 2$). In this case, P_1 gets an expected earning of $\frac{\lambda_1 + x q_1 \lambda_3}{\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3}$, and P_2 gets an expected earning of $\frac{\lambda_2 + x q_2 \lambda_3}{\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3}$.

For reasons that will become clear later, we impose two restrictions on the propagation fees x_1 and x_2. First, there is a smallest unit of currency ϵ (similar to 1 Satoshi in Bitcoin). x_1 and x_2 must be multiples of ϵ. Second, there is a minimum amount $x_{\min} \gg \epsilon$ one has to charge, i.e., $x_1 \geq x_{\min}$, $x_2 \geq x_{\min}$. A miner is not allowed to charge less than $x_{\min}$, and this will be enforced by the protocol. With these two restrictions, we analyze the equilibrium of the game.

- Case 1 is not an equilibrium. P_1 can increase its payoff by propagating to B and charging $x_1 > \frac{\lambda_1}{\lambda_0 + \lambda_1 + \lambda_2}$.
- Case 2 is not an equilibrium. P_2 can increase its payoff by propagating to B and charging $x_2 \leq x_1$.
- Case 3 is not an equilibrium, similarly to Case 2.
- Case 4 is not an equilibrium if $x > x_{\min}$. Between q_1 and q_2, one of them is ≤ 0.5. Without loss of generality, assume $q_1 \leq 0.5$. Then, P_1 can increase its payoff by charging x_1 such that $q_1 x < x_1 < x$. $x_1 = x - \epsilon$ is a viable choice.

- Case 4 with $x = x_{\mathsf{min}}$ is the only equilibrium. Neither P_1 nor P_2 has an incentive to deviate. Charging $> x_{\mathsf{min}}$ or not propagating only decreases one's payoff, while charging $< x_{\mathsf{min}}$ is not allowed.

It is important to note that the equilibrium of the above game does not depend on any player's computational power. Intuitively, both P_1 and P_2 want to "sell" the puzzle B (regardless of their respective computational power), so they enter a "price war" and end up both charging the minimum allowed amount x_{min}. The restriction of $x_{\mathsf{min}} \gg \epsilon$ justifies our "zero propagation cost" assumption: the cost of sending a few KBytes is insignificant compared to the potential earning from selling the puzzle. If $x_{\mathsf{min}} = 0$, the equilibrium would be $x_1 = x_2 = 0$. But this equilibrium is an artifact of an unrealistic game setup. The propagators do not gain anything, so the propagation cost can no longer be ignored.

Now we extend the above simple game in several ways, which we will use in our final analysis.

Game 2a. This game is basically the same as Game 1, but instead of considering producer propagators, we consider two non-producer propagators A_1 and A_2, and whether they should propagate the puzzle to B.

Based on our scheme, the puzzles that A_1 and A_2 receive already carry some fees imposed by their respective predecessors. Suppose the imposed fees are y_1 and y_2, i.e., if A_i ($i = 1, 2$) wins the race, she gets $(1 - y_i)$ reward; a total of y_i is paid to the chain of predecessors that propagate the puzzle to A_i. If B adopts the puzzle from A_i and B wins the race, then B gets $1 - y_i - x_i$, A_i gets x_i and predecessors of A_i combined get y_i.

In this game, let us consider the simple case where $y_1 = y_2$. Through a very similar analysis, it is not hard to show that the only equilibrium is that A_1 and A_2 both propagate and both charge the minimum amount x_{min}. Note that Game 1 is a special case of Game 1a where $y_1 = y_2 = 0$.

Game 2b. This game is the same as Game 2a, except that $y_1 \neq y_2$. Without loss of generality, we assume $y_1 < y_2$. Through a very similar analysis, we can show that A_1 and A_2 will again enter a price war, and that in an equilibrium, A_2 will propagate and charge $x_2 = x_{\mathsf{min}}$. The difference is that A_1 can now easily win the price war if she chooses to do so. Indeed, A_1's best strategy is to win it. It's not in A_1's interest to lose B to A_2. A_1 can choose to enter a tie by charging $x_1' = x_{\mathsf{min}} + y_2 - y_1$, but unless her chance of winning the tie q_1 is close to 1, this is not her best strategy either. The best strategy for A_1 is to charge $x_1 = x_{\mathsf{min}} + y_2 - y_1 - \epsilon$ to win the price war (A_2 cannot match this price), and get an expected earning of $\frac{(1-y_1)\lambda_1 + x_1\lambda_3}{\lambda_0 + \lambda_1 + \lambda_2 + \lambda_3}$.

We remark that the choice of x_1 here explains why we discretize the currency with a smallest unit ϵ. If x_1 were continuous, then A_1 would wish to charge as close to $x_{\mathsf{min}} + y_2 - y_1$ as possible, and formally the game would have no equilibrium.

Game 3a, 3b. We can generalize Game 2a and 2b to have possibly more than two competing propagators. (Note again that

Game 1 is a special case of Game 2a.) It is not hard to show the following equilibrium results. Define a lowest-cost propagator to be one that starts with the lowest imposed fee. There are two cases. If there are at least two lowest-cost propagators, then in an equilibrium, all the lowest-cost propagators charge x_{min}, and it does not matter what the rest of the propagators do. If there is a unique lowest-cost propagator A_i, let the propagator with the second lowest imposed fee be A_j. In an equilibrium, A_j charges x_{min}, A_i charges $x_{\mathsf{min}} + y_j - y_i - \epsilon$, and it does not matter what the rest of the propagators do. In either case, the puzzle that B receives will carry a fee of at most $x_{\mathsf{min}} + \bar{y}$, where $\bar{y}$ is the *second lowest* imposed fee among A_i's adopted puzzles. This is a crucial property that we will use to analyze a general peer-to-peer network below.

Game in a general network. We model a peer-to-peer network as a directed graph. Each vertex in the graph represents a rational and non-cooperative miner. Current committee members are sources in the graph. [6] We then label all nodes using the following rule:

1) Initially, sources are labeled with 0, and all the other nodes are unlabeled.
2) For $l = 1, 2, 3, \ldots$:
   - Find all vertices that have at least two predecessors with label $\leq l - 1$;
   - If such vertices exist, label them with l; else go to step 3).
3) Label all remaining nodes with ∞.

Proposition 1. *In a Nash equilibrium, a miner on a vertex with a label $l \neq \infty$ receives a puzzle that charges a total fee of $l \cdot x_{\mathsf{min}}$.*

Proof. By induction on l. The proposition trivially holds for sources, which have labels 0. Suppose the proposition holds for vertices with label l, we prove it holds for vertices with label $l + 1$.

Consider a vertex v that has label $l + 1$. All predecessors of v form an instance of Game 3a or 3b. In an equilibrium of the network, this local game must also be in an equilibrium. In an equilibrium of Game 3a or 3b, v receives a puzzle that carries a fee of at most $x_{\mathsf{min}} + \bar{y}$ where $\bar{y}$ is the second lowest imposed fee among v's predecessors' adopted puzzles. By our labeling rules, v has at least two predecessors with labels $\leq l$. By the inductive hypothesis, each of these predecessors adopts a puzzle with an imposed fee of $\leq l \cdot x_{\mathsf{min}}$. This means $\bar{y} \leq l \cdot x_{\mathsf{min}}$ and v receives a puzzle with a fee of at most $(l + 1)x_{\mathsf{min}}$. $\qquad\square$

Note that the above result only applies to $l \neq \infty$. If one or more nodes have $l = \infty$, that means they are either completely

---

[6]These assumptions are without loss of generality. We can remove all Byzantine nodes and consolidate all colluding nodes into a single vertex. Committee members surely have incoming connections, but they are irrelevant in the puzzle propagation graph.

eclipsed from the network or connected to only one node in the network. These have to accept any propagation fees the colluding nodes impose on them. If a node notices an abnormally high propagation fee, it should try to make more connections to get out of the eclipse.

C. Comparison to Prior Work

Babaioff *et al.* [3] first spotted Bitcoin's lack of incentives for transaction propagation and proposed a reward scheme. Their work has a few limitations. First, the effectiveness of their reward scheme was only proved in a quite restricted model, in which the peer-to-peer network is assumed to be a regular d-ary tree with height H. Second, they did not discuss how to enforce the proposed reward division. Third, their scheme seems to have high reward overheads. If a transaction gives one unit of fee to its successful miner, up to $\log H$ units of rewards have to be given to propagators. Considering that propagation is an easy and cheap task, this reward overhead is rather high. Our work addresses all three issues. We analyze our protocol in a general peer-to-peer network. We present a concrete cryptographic protocol that not only enforces the reward division, but also allows each miner to set a propagation surcharge of its own choice. This opens up the propagation surcharge to free market competition, which combined with the low cost of propagation will drive the surcharge to a very small amount ($x_{\min}$) in most cases.

VII. DISCUSSION

There have been many proposals for designing cryptocurrencies and blockchains. Fundamentally, different proposals represent different trade-off points between fault tolerance and efficiency. Bitcoin-NG [11] improves throughput over Bitcoin but its single leader approach creates concerns on fault tolerance. If a leader fails due to either a crash or a DoS attack, no transactions can be approved in that leader's epoch. If multiple consecutive leaders fail, the system temporarily loses liveness. Solidus along with other committee-based designs [8], [16], [24] offer better fault tolerance. It is significantly harder for an adversary to launch DoS attacks simultaneously on a one-third fraction of committee members. Of course, at the other end of the spectrum, Bitcoin's leaderless/committee-less design enjoys even better fault tolerance in theory.[7] As an informal analogy, one can think of the Bitcoin committee to be the set of all active miners. In this sense, Solidus and related work [8], [16], [24] down-sample miners into a medium-sized committee. The committee size can be thought of as a security parameter rather than the total number of miners in the world. This approach trades off some degree of fault tolerance for shorter confirmation time and better scalability.

For all committee based protocols, it is essential that the committee be sampled (almost) uniformly at random from all active miners even when a fraction of miners behave adversarially (performing selfish mining, for instance). Moreover,

it is imperative to assume that an adversary cannot corrupt a miner immediately. Otherwise, an adversary can corrupt a miner as soon as it joins the committee, and easily take over the committee that way. Delayed corruption [24] is a reasonable assumption in practice as it takes some time to either bribe a miner or infect a clean host.

Next, we discuss our choice for using PBFT. We first note that ideally (assuming no selfish mining, etc.) Bitcoin can resist an adversary with up to $\sim$49% computation power. By using PBFT, we can only resist a 33% adversary. The underlying difference is that PBFT itself offers security in an fully asynchronous setting, in which case, resistance against a 33% adversary is the best one can hope to achieve. However, the use of PoW fundamentally relies on a synchronicity assumption. If messages can be arbitrarily delayed in an adversarial way, PoW is trivially broken: Imagine whenever one honest participant finds a PoW, its message containing the PoW is arbitrarily delayed, then the adversary has complete control over the ledger. Solidus and related designs all use PoW, so PBFT's safety guarantee in an asynchronous setting is indeed overly strong for us. We could potentially choose a synchronous Byzantine consensus protocol and get back the 50% resistance (still assuming time to obtain a PoW is much larger than the worst case round trip delay of the network). However, these protocols proceed in rounds where a round is longer than than the worst case communication delay between any two participants. They will be inefficient for the Internet, where the worst-case message delay far exceeds the average delay. By using PBFT, the protocol proceeds at a speed of average network delay. As soon as enough participants receive messages, the protocol moves on to the next phase/slot.

Finally, incentive compatibility is imperative to any permissionless and decentralized cryptocurrency. To this end, we modified PBFT and our surrounding protocols to make Solidus incentive compatible at every step.

VIII. CONCLUSION

In summary, this work presents Solidus, a scalable and incentive compatible cryptocurrency based on a fault-tolerant committee and permissionless Byzantine consensus. It is secure against withholding attacks such as selfish mining and provides instantaneous confirmation of transactions. Perhaps a limitation of Solidus is that the protocol is currently rather complex. It remains interesting future work to simplify and improve the Solidus protocol.

REFERENCES

[1] Litecoin. https://litecoin.org/.
[2] Ittai Abraham, Lorenzo Alvisi, and Joseph Y. Halpern. Distributed computing meets game theory: combining insights from two fields. *Acm Sigact News*, 42(2):69–76, 2011.
[3] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On bitcoin and red balloons. In *Proceedings of the 13th ACM conference on electronic commerce*, pages 56–73. ACM, 2012.
[4] Vitalik Buterin. A next generation smart contract and decentralized application platform, 2013. https://github.com/ethereum/wiki/wiki/White-Paper.

---

[7] The fact that the top 5 mining pools control almost 90% of the mining power makes this argument less clear.

[5] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 154–167. ACM, 2016.

[6] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the third symposium on Operating systems design and implementation*, pages 173–186. USENIX Association, 1999.

[7] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[8] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, page 13. ACM, 2016.

[9] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[10] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[11] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.

[12] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[13] Bryan Ford. Untangling mining incentives in bitcoin and byzcoin, 2016. http://bford.github.io/2016/10/25/mining/.

[14] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[15] Philipp Jovanovic. Byzcoin: Securely scaling blockchains, 2016. http://hackingdistributed.com/2016/08/04/byzcoin/.

[16] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing, 2016. arXiv preprint.

[17] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[18] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[19] Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, and Prateek Saxena. SCP: a computationally-scalable byzantine consensus protocol for blockchains, 2015. Cryptology ePrint Archive, Report 2015/1168.

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

[21] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 305–320. IEEE, 2016.

[22] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016. http://eprint.iacr.org/2016/454.

[23] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. 2016. Cryptology ePrint Archive, Report 2016/916.

[24] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. 2016. Cryptology ePrint Archive, Report 2016/917.

[25] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.

[26] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.