FruitChains: A Fair Blockchain

Rafael Pass
CornellTech
New York, NY
rafael@cs.cornell.edu

Elaine Shi
Cornell
Ithaca, NY
elaine@cs.cornell.edu

ABSTRACT

Nakamoto's famous *blockchain* protocol enables achieving consensus in a so-called *permissionless setting*—anyone can join (or leave) the protocol execution, and the protocol instructions do not depend on the identities of the players. His ingenious protocol prevents "sybil attacks" (where an adversary spawns any number of new players) by relying on *computational puzzles* (a.k.a. "moderately hard functions") introduced by Dwork and Naor (Crypto'92). Recent work by Garay et al (EuroCrypt'15) and Pass et al (manuscript, 2016) demonstrate that this protocol provably achieves *consistency* and *liveness* assuming a) honest players control a majority of the computational power in the network, b) the puzzle-hardness is appropriately set as a function of the maximum network delay and the total computational power of the network, and c) the computational puzzle is modeled as a random oracle. Assuming honest participation, however, is a strong assumption, especially in a setting where honest players are expected to perform a lot of work (to solve the computational puzzles). In Nakamoto's Bitcoin application of the blockchain protocol, players are *incentivized* to solve these puzzles by receiving rewards for every "block" (of transactions) they contribute to the blockchain. An elegant work by Eyal and Sirer (FinancialCrypt'14), strengthening and formalizing an earlier attack discussed on the Bitcoin forum, demonstrates that a *coalition* controlling even a minority fraction of the computational power in the network can gain (close to) 2 times its "fair share" of the rewards (and transaction fees) by deviating from the protocol instructions. In contrast, in a *fair* protocol, one would expect that players controlling a ϕ fraction of the computational resources to reap a ϕ fraction of the rewards.

We present a new blockchain protocol—the FruitChain protocol—which satisfies the same consistency and liveness properties as Nakamoto's protocol (assuming an honest majority of the computing power), and additionally is δ-*approximately fair*: with overwhelming probability, any honest set of players controlling a ϕ fraction of computational power is guaranteed to get at least a fraction $(1-\delta)\phi$ of the blocks (and thus rewards) in *any* $\Omega(\frac{\kappa}{\delta})$ length segment of the chain (where κ is the security parameter). Consequently, if this blockchain protocol is used as the ledger underlying a cryptocurrency system, where rewards and transaction fees are *evenly distributed* among the miners of blocks in a length κ segment of the chain, no coalition controlling less than a majority of the computing power can gain more than a factor $(1+3\delta)$ by deviating from the protocol (i.e., honest participation is an $\frac{n}{2}$-*coalition-safe* 3δ-*Nash equilibrium*). Finally, the FruitChain protocol enables decreasing the *variance* of mining rewards and as such significantly lessens (or even obliterates) the need for mining pools.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**;

KEYWORDS

Distributed consensus; blockchains; fairness; Nash equilibrium

1 INTRODUCTION

Distributed systems have been historically analyzed in a *closed* setting—a.k.a. the *permissioned setting*—in which the number of participants in the system, as well as their identities, are common knowledge. In 2008, Nakamoto [16] proposed his celebrated "blockchain protocol" which attempts to achieve consensus in a *permissionless* setting: anyone can join (or leave) the protocol execution (without getting permission from a centralized or distributed authority), and the protocol instructions do not depend on the identities of the players. The core blockchain protocol (a.k.a. "Nakamoto consensus", or the "Bare-bones blockchain protocol"), roughly speaking, is a method for maintaining a *public*, *immutable* and *ordered* ledger of records (for instance, in the Bitcoin application, these records are simply transactions); that is, records can be added to the *end* of the ledger at any time (but only to the end of it); additionally, we are guaranteed that records previously added cannot be removed or reordered and that all honest users have a *consistent view* of the ledger—we refer to this as *consistency*. Additionally, the protocol should satify a *liveness* property: transactions submitted by an honest user get incorporated into the ledger sufficiently fast.

The key challenge with the permissionless setting is that an attacker can trivially mount a so-called "sybil attack"—it simply spawns lots of players (that it controls) and can thus easily ensure that it controls a majority of all the players. Indeed, Barak et al [3] proved that this is a fundamental problem with the permissionless model. Nakamoto blockchain protocol overcomes this issue by relying on "computational puzzles"—a.k.a. *moderately hard functions* or *proofs of work*—put forth by Dwork and Naor [6]: roughly speaking, the participants are required to solve the computational puzzle of some well-defined diffulty in order to confirm transactions—this is referred to as *mining*. Next, rather than attempting to provide robustness whenever the majority of the participants are honest (since participants can be easily spawned in the permissionless setting), Nakamoto's goal was to provide robustness of the protocol under the assumption that a *majority of the computing power* is held

PODC '17, July 25-27, 2017, Washington, DC, USA

ACM ISBN 978-1-4503-4992-5/17/07…$15.00
https://doi.org/10.1145/3087801.3087809

by honest participants. Indeed, recent works by Garay et al. [8] and Pass et al. [18] formally proved that Nakamoto's blockchain protocol satisfies the above-mentioned consistency and liveness under different network assumptions, as long as the puzzle difficulty (referred to as the *mining hardness*) is apporpriately set as a function of the maxiumum delay in the network.

Nakamoto's blockchain represents an exciting breakthrough: it demonstrated that distributed consensus is possible on an Internet scale. The above analyses, however, assume that a majority of the computing power is controlled by honest players, and that honest players correctly execute the protocol. Assuming such honest participation is a strong assumption, especially in a setting where honest players are expected to perform a lot of work (to solve the computational puzzles)—why would we expect players to want to participate if it is costly! (This can be formalized in the Game-Theory with Costly computation framework of Halpern and Pass [9]). In Nakamoto's ingenious Bitcoin application of the blockchain protocol, players are thus *incentivized* to solve these puzzles by receiving, so-called, *block rewards* for every "blocks" (of transactions) they contribute to the blockchain; additionally, the miners also receive *transaction fees* for all the transactions that are confirmed in the block. The hope is that these reward mechanism (block rewards and transaction fees) properly incentivize honest participation. Unfortunately, as shown by several recent work, this is not the case:

- *Selfish mining undermines incentive compatibility.* Nakamoto's blockchain suffers from a so-called *selfish-mining* attack, where even a minority coalition that controls network delivery can manage to reap close to twice its fair share of block rewards [4, 7, 15, 17, 21] — in particular, if the adversary wields close to a half of the computational power, it can reap almost all of the rewards, thus denying honest players of (almost) any reward! (More specifically, whenever the adversary mines a new block, it simply *withholds it* (not sharing it with the honest players), and only releases it when some honest player mines a new block—if the adversary controls the network it can ensure that all honest players receive the adversarial block before the block mined by the honest players, and as such, it effectively "erases" the honest player's block replacing it with its own block.) This selfish mining attack was first observed in discussions on the Bitcoin forum [15]; the first analytical study provided by Eyal and Sirer [7], and subsequently improved by Sapirshtein et al. [21] and Nayak et al. [17].
- *Transaction fees exacerbate instability.* Due to Bitcoin's particular coin minting schedule, the block reward is scheduled to decrease over time and miners are expected to obtain rewards increasingly more from transaction fees. A recent work by Carlsten et al. [4] (concurrent to ours) demonstrates that the situation gets even worse once we take into account the transaction fees: as a simplest example, if a block contains transactions with large fees, miners will be incentivized to create a "fork" and attempt to confirm the transaction themselves.
- *Mining pools harm decentralization.* Finally, to maintain consistency of the blockchain, the puzzle difficulty is (and by the analysis of [18] need to be set) so that the whole world combined mines a block (i.e., solves a computational puzzle) roughly every 10 minutes. Consequently, an individual "solo miner" with state-of-the-art equiment must wait on average, roughly, 2 years before it gets any rewards [2]. This has led to the formation of "mining pools" where miners are coordinated by a pool operator and share the rewards to reduce the variance of their gains. In essence, the decentralized nature of the blockchain is lost.

1.1 Our Results

In this work, we introduce a notion of fairness for blockchain protocols: Roughly speaking, we say that a blockchain protocol that is *fair* if honest players that wield ϕ fraction of the computational resources will reap at least ϕ fraction of the blocks in any sufficiently long window of the chain. (This notion of fairness can be viewed as a strengthen form of the notion of "ideal chain quality" considered, but not acheived, in [8, 18]) More precisely, we say that a blockchain protocol is *δ-approximately fair* w.r.t. ρ attackers if, with overwhelming probability, any ϕ fraction coalition of *honest* users is guaranteed to get at least a $(1-\delta)\phi$ fraction of the blocks in every sufficiently long window of the chain, even in the presence of an adversary controlling up to a ρ fraction of the computing power. Our main theorem shows how to acheive a blockchain which satisfies the same consistency and liveness properties as Nakamoto's one, as well as fairness:

Theorem 1.1 (Informally stated). *Let $\rho < \frac{1}{2}$ be a constant. Then, for every constant $\delta > 0$, there exists a blockchain protocol that satisfies consistency, liveness and δ-approximate fairness.*

Note that approximate fairness directly implies that an attacker cannot get "much" more than its fair share of the block rewards (and thus it disincentivizes selfish mining). But the instability issue with transaction fees still remains, and so does the mining pool issue. We finally demonstrate that our protocol provides a solution to both these issues as well:

- Regarding transaction fees: we suggest a method for spreading out the transaction fees of a block over the miners of a *sequence* of blocks preceeding it. As we show, any fair blockchain protocol can be used to disincentivize deviation even in the presence of transaction fee under this new reward rule. More precisely, we show that no coalition controlling less than a majority of the computing power can gain more than a factor $(1 + 3\delta)$ of the block rewards and transaction fees by deviating from the protocol—that is, honest participation is an *$\frac{n}{2}$-coalition-safe 3δ-Nash equilibrium.*
- Regarding mining pools, we demonstrate that the block (i.e., fruit) mining difficulty in our protocol can be made almost arbitrarily small, and as a consequence, miners can get paid much more often. Indeed, experimental results implenting our new blockchain [1] show that with Bitcoin current block size of 1MB, by sacrificing 8% to 10% of the block to new meta data, we can ensure that miners get paid 1000x more often (and thus on average, roughly, twice per day). Consequently, there is no longer a need for pooled mining.

1.2 Protocol Overview

To explain our protocol, let us first recall Nakamoto's blockchain protocol as we will make use of it.

Nakamoto's protocol in a nutshell. Roughly speaking, players "confirm" records/transactions by "mining blocks of transactions" through solving some computational puzzle that is a function of the transactions and the history so far. More precisely, each participant maintains its own local "chain" of "blocks" of records —called the *blockchain*. Each block consists of a triple $(h_{-1}, \eta, \mathsf{m})$ where h_{-1} is a pointer to the previous block in chain, m is the record component of the block, and η is a "proof-of-work"—a solution to a computational puzzle that is derived from the pair (h_{-1}, m). The proof of work can be thought of as a "key-less digital signature" on the whole blockchain up until this point.

Concretely, Nakamoto's protocol is parametrized by a parameter p—which we refer to as the *mining hardness parameter*, and a proof-of-work is deemed valid if η is a string such that $\mathsf{H}(h_{-1}, \eta, \mathsf{m}) < D_p$, where H is a hash function (modeled as a random oracle) and D_p is set so that the probability that an input satisfies the relation is less than p. At any point of the protocol execution, each participant attempts to increase the length of its own chain by "mining" for a new block: upon receiving some record m, it picks a random η and checks whether η is a valid proof of work w.r.t. m and h_{-1}, where h_{-1} is a pointer to the last block of its current chain; if so, it extends is own local chain and broadcast it to the all the other participants. Whenever a participant receives a chain that is longer than its own local chain, it replaces its own chain with the longer one.

The FruitChain *protocol.* Roughly speaking, our FruitChain protocol will be running an instance of Nakamoto's blockchain protocol, but instead of directly storing the records m inside the blockchain, the records are put inside "fruits" denoted f; these fruits themselves require solving some proof of work, with a *different hardness parameter* p_f; additionally, we require the fruits to "hang" from a block in the chain which is not too "far" from the block which records the fruit—more specifically, the fruit needs to "point" to an earlier block in the chain which is not too far from the block containing it (and thus, the fruit could not have been mined "too" long ago)—we refer to such a fruit as being *recent*. In this new protocol, the fruits play the roles of "blocks"—i.e., *"orange is the new block"*[1]—and chain quality is thus defined in terms of fruits.

In each round, honest players simultaneously mine for a fruit and a block (for Nakamoto's blockchain) by making one invocation of the hash function—this follows the 2-for-1 trick of [8] where, say, the prefix of the output of H determines whether fruit mining is successful, and the suffix is used to determine whether block mining is successful. Whenever a player successfully mines a fruit it broadcasts it to all other players; fruits that have not yet been recorded in the blockchain (and are still recent) are stored in a buffer and all honest players next attempt to add them to the blockchain.

Intuitively, the reason why "selfish mining" fails is that even if an adversary tries to "erase" some block mined by an honest player (which contains some honest fruits), by the chain growth and chain quality properties of the underlying blockchain, eventually an honest player will mine a new block which is stable and this honest player will include the fruits in it—in fact, the time before such an "honest block" arrives is short enough for the fruit to still be "recent" at the time of the honest block arriving.

Intuitively, the reason why we require fruits to be recent is to prevent a different kind of attack: without it, an attacker could *withhold fruits*, and suddenly release lots of them at the same time, thereby creating an very high fraction of adversarial fruits in some segment of the (fruit) chain. By requiring the fruits to be recent, we prevent the adversary from squirreling away (too many of) its fruits: since the underlying blockchain has a guaranteed liveness, we can upperbound the extra amount of time the attacker can withhold fruits and thus upperbound the number of extra fruits it can release in any window.

1.3 Related Work

Comparison with GHOST, the Inclusive Blockchain, and [8]. Although our approach of including fruits in a main blockchain take inspiration from to the earlier elegant works on GHOST [22] and inclusive blockchains [14], we stress that these earlier works do not attain our goals of providing a provably secure, fair blockchain. GHOST [22] is a mechanism such that forking blocks not on the main chain will affect the chain selection rule— however, as the subsequent work by Kiayias and Panagiotakos [12] shows, GHOST actually worsens "chain quality" (i.e., the fraction of honest blocks in the chain) whereas our goal is to improve chain quality and fairness. The inclusive blockchain work proposes to maintain a direct acyclic graph rather than a chain, such that forking subtrees may be included in the linearized transaction log — despite the superficial resemblance at first sight, the mechanisms employed by the inclusive blockchain is actually quit different from how we include fruits in the main blockchain.

As mentionned above, our protocol borrows the 2-for-1 trick from the work Garay et al [8] which also relied on a separate "mining process" to acheive a different goal (namely, to implement a broadcast channel from a blockchain).

Subsequent works. In both Nakamoto's blockchain and ours, the time needed to confirm transactions grows with the *worst-case upper-bound* on the network delay [18, 20]. In contrast, in a *responsive* protocol, we require the confirmation time to only be a function of the *actual* network delay, which may be a lot smaller than the worst-case one. In a companion paper called *hybrid consensus* [20], we show how to combine any blockchain protocol with classical asynchronous consensus to improve the latency of the blockchain protocol and achieve responsiveness. Roughly speaking, hybrid consensus makes use of a blockchain to elect a committee—more specifically, the miners of blocks in a sufficiently long segment of the chain are elected as the committee—and then this committee executes the classical consensus protocol. The chain quality of the blockchain determines the fraction of honest players in the committee: if we employ Nakamoto's blockchain, we would need to require that $\frac{3}{4}$ of the computing power is controlled by honest player to ensure a chain quality of $\frac{2}{3}$ and thus a fraction $\frac{2}{3}$ honest committee members (which is required by the consensus protocol). In contrast, by relying on our new FruitChain protocol, it suffices to assume that $\frac{2}{3}$ of the computing power is controlled by honest players. We highlight that, as shown in [20], achieving a responsive protocol

[1] We thank Hugo Krawczyk for this phrase!

also *requires* assuming that $\frac{2}{3}$ of the computing is held by honest parties, and as such relying on our FruitChain protocol enables achieving an *optimal resilience* for low-latency blockchains.

Besides hybrid consensus, other subsequent works have also employed ideas from FruitChain to achieve incentive compatibility in blockchain style protocols. Notably, recent provably secure proof-of-stake protocols, including Snow White [5] and Ouroboros [13], argue that the idea from FruitChain is applicable to non-proof-of-work blockchains as well.

Other related works. Kiayias et al. [10] model Bitcoin mining as a game, where nodes decide on which blocks to extend and whether to release a mined block. They show that for small players controlling less than $\frac{1}{3}$ of the resources, following Bitcoin's protocol specification is a Nash equilibrium. Their results, however, only apply to a rather constrained idealistic model where all honest miners can communicate with 0 latency, and the adversary cannot perform form any network level attacks (such as rushing). As we mentionned in the introduction, in our model where the adversary can control the delivery of messages, the bitcoin protocol is not incentive compatible even for players controlling less than a $\frac{1}{3}$ of the computational resources—there is a selfish mining attack which enables an attacker to gain $\frac{1}{2}$ of the block rewards.

2 PRELIMINARIES AND DEFINITIONS

2.1 Protocol Execution Model and Notations

A protocol refers to an algorithm for a set of interactive Turing Machines (also called nodes) to interact with each other. The execution of a protocol Π that is directed by an environment $Z(1^\kappa)$ (where κ is a security parameter), which activates a number of parties $1, 2, \ldots, n$ as either "honest" or corrupted parties. Honest parties would faithfully follow the protocol's prescription, whereas corrupt parties are controlled by an adversary A which reads all their inputs/message and sets their outputs/messages to be sent.

The environment Z is a terminology often used in protocol composition in the cryptography literature — one can regard the environment Z a catch-all term that encompasses everything that lives outside the "box" defined by the protocol. For example, as mentioned later, part of the environment Z's job is to provide inputs to honest nodes and receive outputs from them. This models the fact that the inputs to the protocol may originate from external applications and the protocol's outputs can be consumed by external applications where any external application or other protocols running in the system are viewed as part of Z.

- A protocol's execution proceeds in *rounds* that model atomic time steps. At the beginning of every round, honest nodes receive inputs from an environment Z; at the end of every round, honest nodes send outputs to the environment Z.
- A is responsible for delivering all messages sent by parties (honest or corrupted) to *all* other parties. A cannot modify the content of messages broadcast by honest players, *but it may delay or reorder the delivery of a message* as long as it eventually delivers all messages. (Later, we shall consider restrictions on the delivery time.) The identity of the sender is not known to the recipient.[2]
- At any point, Z can *corrupt* an honest party j which means that A gets access to its local state and subsequently, A controls party j. (In particular, this means we consider a model with "erasures"; random coin tosses that are no longer stored in the local state of j are not visible to A.)[3]
- At any point, Z can *uncorrupt* a corrupted player j, which means that A no longer controls j. A player that becomes uncorrupt is treated in the same way as a newly spawning player, i.e., the player's internal state is re-initialized and then the player starts executing the honest protocol no longer controlled by A.

Notations for randomized execution. A protocol's execution is randomized, where the randomness comes from honest players as well as the adversary denoted A that controls all corrupt nodes, and the environment Z that sends inputs to honest nodes during the protocol execution. We use the notation $\mathsf{view} \xleftarrow{\$} \mathsf{EXEC}^{\Pi}(A, Z, \kappa)$ to denote a randomly sampled execution trace, and $|\mathsf{view}|$ denotes the number of rounds in the execution trace view. More specifically, view is a random variable denoting the joint view of all parties (i.e., all their inputs, random coins and messages received, including those from the random oracle) in the above execution; note that this joint view fully determines the execution.

Constraints on (A, Z). The environment Z and the adversary A must respect certain constraints. We say that a p.p.t. pair (A, Z) is (n, ρ, Δ)-*respecting* w.r.t. Π, iff for every $\kappa \in N$, every view view in the support of $\mathsf{EXEC}^{\Pi}(A, Z, \kappa)$, the following holds:

(1) Z activates n parties in view;
(2) For any message broadcast by an honest player at any time t in view, any player that is honest at time $t + \Delta$ or later must have received the message (including those that might have newly spawned). As long as this Δ contraint is respected, A is allowed to delay or reorder honest players' messages arbitrarily.
(3) at any round r in view, A controls at most $\rho \cdot n$ parties; and

Let $\Gamma(\cdot, \cdot, \cdot)$ be a boolean predicate. We say that a p.p.t. pair (A, Z) is Γ-*compliant* w.r.t. protocol π iff

- (A, Z) is (n, ρ, Δ)-respecting w.r.t. π; and
- $\Gamma(n, \rho, \Delta) = 1$.

In other words Γ is a predicate that places constraints on additional constraints on the parameter (n, ρ, Δ) that (A, Z) must respect. When the context is clear, we often say that (A, Z) is Γ-compliant while omitting to specify w.r.t. which protocol.

2.2 Conventions

Variables that are functions of the security parameter. Unless otherwise noted, by default we assume that all variables are a function of the security parameter κ. If any variable is not a function of κ, we shall explicitly note that the variable is a *constant*. Variables may also be functions of each other as defined later by relations

[2] We could also consider a seemingly weaker model where messages sent by corrupted parties need not be delivered to all honest players. We can easily convert the weaker model to the stronger model by having honest parties "gossip" all messages they receive.
[3] Our proof actually extends also to the model "without erasures".

that (A, Z) must additionally satisfy for our blockchain protocol to be secure.

For two variables that by default are functions of κ, we say that $\mathsf{var}_1 < \mathsf{var}_2$ iff for every $\kappa \in \mathbb{N}$, $\mathsf{var}_1(\kappa) < \mathsf{var}_2(\kappa)$. Similarly, if we say that var is positive, we mean that $\mathsf{var}(\kappa)$ is positive for any $\kappa \in \mathbb{N}$.

Other conventions. Throughout this paper, whenever we refer to p.p.t. machines, we mean that the machine is *non-uniform* probabilistic polynomial-time.

2.3 Blockchain Protocols

In this section, we recall the abstract model for blockchain protocols from [18] and a provide a description of Nakamoto's original blockchain protocol which we will heavily make use of.

In a blockchain protocol Π, nodes receive input records from an environment Z, and nodes interact with each other to agree on a linearly ordered log of transactions in a way that achieves consistency and liveness.

Inputs and outputs of a blockchain protocol. At the beginning of each time step, the environment Z inputs a record m to each honest player. At the end of each time step, each honest player outputs a chain to the environment Z, where chain denotes an ordered sequence of records (also referred to as blocks). Each record (or block) may in turn contain an ordered sequence of transactions. Henceforth we use the notation

$$\text{output of node } i \text{ in round } t: \quad \mathsf{chain}_i^t(\text{view})$$

to denote the output of node i in round t to Z in a given execution trace view.

Modeling proofs-of-work. To model Nakamoto's blockchain protocol, we need to extend the protocol execution model with a random oracle. In an execution with security parameter κ, we assume all parties have access to a random function $H : \{0,1\}^* \to \{0,1\}^\kappa$ which they can access through two oracles: $\mathsf{H}(x)$ simply outputs $H(x)$ and $\mathsf{H.ver}(x, y)$ output 1 iff $H(x) = y$ and 0 otherwise. In any round r, the players (as well as A) may make *any* number of queries to H.ver. On the other hand, in each round r, honest players can make only a *single* query to H, and an adversary A controlling q parties, can make q *sequential* queries to H. (This modeling is meant to capture the assumption that we only "charge" for the effort of finding a solution to a "proof of work" [6], but checking the validity of a solution is cheap. We discuss this further after introducing Nakamoto's protocol.) We emphasize that the environment Z does not get direct access to the random oracle (but can instruct A to make queries).

2.4 Nakamoto's Blockchain Protocol

We describe Nakamoto's protocol [16] referred to as $\Pi_{\mathrm{nak}}(p)$. $\Pi_{\mathrm{nak}}(p)$ takes in a puzzle difficulty parameter p that denotes the probability that each player mines a block in a single round.

Protocol overview. In Π_{nak}, each honest node maintains an internal state *chain* at any point of time. Each $chain[i]$ is referred to as a (mined) block and is of the format $chain[i] := (h_{-1}, \eta, \mathsf{m}, h)$, containing the hash of the previous block denoted h_{-1}, a nonce η, a record m, and a hash h [4] Let $\mathsf{chain} := \mathsf{extract}(chain)$ be the sequence of records contained in the sequence of blocks *chain*. chain is the version that honest nodes output to the environment.

The Π_{nak} works as follows:

- Nodes that are newly spawned or that have been become uncorrupt start with initial chain containing only a special genesis block: $chain := (0, 0, \bot, \mathsf{H}(0, 0, \bot))$.
- In every round: a node reads all incoming messages (delivered by A). If any incoming message $chain'$ is a valid sequence of blocks that is longer than its local state *chain*, replace *chain* by $chain'$. We define what it means for a chain to be valid later. Checking the validity of $chain'$ can be done using only H.ver queries.
- Read an input record m from the environment Z. Now parse $\mathsf{chain}[-1] := (\_, \_, \_, h_{-1})$, pick a random nonce $\eta \in \{0,1\}^\kappa$, and issue query $h = \mathsf{H}(h_{-1}, \eta, \mathsf{m})$. If $h < D_p$, then append the *newly mined* block (h_{-1}, η, b, h) to *chain* and broadcasts the updated *chain*. More specifically, $D_p = p(\kappa) \cdot 2^\kappa$ such that for all (h, m), $\Pr_n[\mathsf{H}(h, \eta, \mathsf{m}) < D_p] = p$. In other words, D_p is appropriately parameterized such that the probability that any player mines a block in a round is p.
- Output $\mathsf{chain} := \mathsf{extract}(chain)$ to the environment Z. Note that the notation chain extracts only the sequence of records from *chain* removing all other metadata that are not needed by external applications.

Valid chain. We say a block $chain[i] = (h_{-1}, \eta, \mathsf{m}, h)$ is *valid with respect to (a predecessor block)* $chain[i-1] = (h'_{-1}, n', m', h')$ if two conditions hold: $h_{-1} = h'$, $h = \mathsf{H}(h_{-1}, \eta, \mathsf{m})$, and $h < D_p$. A chain of blocks *chain* is *valid* if a) $chain[0] = (0, 0, \bot, \mathsf{H}(0, 0, \bot))$ is the genesis block, and b) for all $i \in [\ell]$, $chain[i]$ is valid with respect to $chain[i-1]$.

Remark: on our use of the random oracle. Recall that in our model, we restrict players to a single evaluation query H per round, but allow them any number of verification queries H.ver in the same round. We do this to model the fact that checking the validity of mined blocks is "cheap" whereas the mining process is expensive. (To enable this, we have included a pointer h to the current record in every mined block in the description of Nakamoto).

In practice, the cost of evaluating a hash function (which is used to instantiate the random oracle) is the same as verifying its outputs, but our modeling attempts to capture the phenomena that a miner typically use various heuristics (such as black lists of IP addresses that have sent invalid blocks) and different hardware to check the validity of a mined block versus to mine a new block.

2.5 Security of Blockchain Protocols

We recall the security properties of blockchains from [18], which in turn are based on earlier definitions from [8, 11] For our purposes, we slightly generalize the properties from [18] (see below for a discussion of this generalization), but point out that our generalized definitions suffice for all known applications of them; see [18] for more discussion (and historical remarks) on these definitions.

[4] In reality (as well as in the description in the introduction), h is not included in the block (as it can be easily determined from the remaining elements); we include it to ensure that we can verify validity of a block using only H.ver.

Negligible functions. A function $\epsilon(\cdot)$ is said to be *negligible* if for every polynomial $p(\cdot)$, there exists some κ_0 such that $\epsilon(\kappa) \leq \frac{1}{p(\kappa)}$ for all $\kappa \geq \kappa_0$.

We now define three useful properties, referred to as chain growth, chain quality, and consistency respectively. Note that all properties are defined over honest nodes' outputs to the environment that are visible at the abstraction level (rather than over nodes' internal states *chain*).

2.5.1 Chain Growth. The first desideratum is that the chain grows proportionally with the number of time steps. Let,

$$\text{min-chain-increase}_{t,t'}(\text{view}) = \min_{i,j} |\text{chain}_j^{t+t'}(\text{view})| - |\text{chain}_i^t(\text{view})|$$

$$\text{max-chain-increase}_{t,t'}(\text{view}) = \max_{i,j} |\text{chain}_j^{t+t'}(\text{view})| - |\text{chain}_i^t(\text{view})|$$

where we quantify over nodes i, j such that i is honest at round t and j is honest at round $t + t'$ in view.

Let $\text{growth}^{t_0,t_1}(\text{view}, \Delta, T) = 1$ iff the following properties hold:

- **(consistent length)** for all time steps $t \leq |\text{view}| - \Delta$, $t + \Delta \leq t' \leq |\text{view}|$, for every two players i, j such that in view i is honest at t and j is honest at t', we have that $|\text{chain}_j^{t'}(\text{view})| \geq |\text{chain}_i^t(\text{view})|$
- **(chain growth lower bound)** for every time step $t \leq |\text{view}| - t_0$, we have $\text{min-chain-increase}_{t,t_0}(\text{view}) \geq T$.
- **(chain growth upper bound)** for every time step $t \leq |\text{view}| - t_1$, we have $\text{max-chain-increase}_{t,t_1}(\text{view}) \leq T$.

In other words, growth^{t_0,t_1} is a predicate which tests that a) honest parties have chains of roughly the same length, and b) during any t_0 time steps in the execution, all honest parties' chains increase by at least T, and c) during any t_1 time steps in the execution, honest parties' chains increase by at most T.

Definition 2.1 (Chain growth). A blockchain protocol Π satisfies (T_0, g_0, g_1)-chain growth in Γ-environments, if for all Γ-compliant p.p.t. pair (A, Z), there exists some negligible function negl such that for every $\kappa \in \mathbb{N}$, $T \geq T_0$, $t_0 \geq \frac{T}{g_0}$ and $t_1 \leq \frac{T}{g_1}$,

$$\Pr\left[\text{view} \leftarrow \text{EXEC}^{\Pi}(A, Z, \kappa) : \text{growth}^{t_0,t_1}(\text{view}, \Delta, \kappa) = 1\right] \geq 1 - \text{negl}(\kappa)$$

2.5.2 Chain Quality. The second desideratum is that the number of blocks contributed by the adversary is not too large.

Given a chain, we say that a block $B := \text{chain}[j]$ is honest w.r.t. view and prefix $\text{chain}[:j']$ where $j' < j$ if in view there exists some node i honest at some time $t \leq |\text{view}|$, such that 1) $\text{chain}[:j'] \prec \text{chain}_i^t(\text{view})$ where $\prec$ denotes "is a prefix of" and 2) Z input B to node i at time t. Informally, for an honest node's chain denoted chain, a block $B := \text{chain}[j]$ is honest w.r.t. a prefix $\text{chain}[:j']$ where $j' < j$, if earlier there is some honest node who received B as input when its local chain contains the prefix $\text{chain}[:j']$.

Let $\text{quality}^T(\text{view}, \mu) = 1$ iff for every time t and every player i such that i is honest at t in view, among any consecutive sequence of T blocks $\text{chain}[j+1..j+T] \subseteq \text{chain}_i^t(\text{view})$, the fraction of blocks that are honest w.r.t. view and $\text{chain}[:j]$ is at least μ.

Definition 2.2 (Chain quality). A blockchain protocol Π satisfies (T_0, μ)−*chain quality*, in Γ-environments if for all Γ-compliant p.p.t. pair (A, Z), there exists some negligible function negl such that for every $\kappa \in \mathbb{N}$ and every $T \geq T_0$ the following holds:

$$\Pr\left[\text{view} \leftarrow \text{EXEC}^{\Pi}(A, Z, \kappa) : \text{quality}^T(\text{view}, \mu) = 1\right] \geq 1 - \text{negl}(\kappa)$$

2.5.3 Consistency. Roughly speaking, consistency stipulates common prefix and future self-consistency. Common prefix requires that all honest nodes' chains, except for roughly $O(\kappa)$ number of trailing blocks that have not stabilized, must all agree. Future self-consistency requires that an honest node's present chain, except for roughly $O(\kappa)$ number of trailing blocks that have not stabilized, should persist into its own future. These properties can be unified in the following formal definition (which additionally requires that at any time, two honest nodes' chains must be of similar length).

Let $\text{consistent}^T(\text{view}) = 1$ iff for all times $t \leq t'$, and all players i, j (potentially the same) such that i is honest at t and j is honest at t' in view, we have that the prefixes of $\text{chain}_i^t(\text{view})$ and $\text{chain}_j^{t'}(\text{view})$ consisting of the first $\ell = |\text{chain}_i^t(\text{view})| - T$ records are identical — this also implies that the following must be true: $\text{chain}_j^{t'}(\text{view}) > \ell$, i.e., $\text{chain}_j^{t'}(\text{view})$ cannot be too much shorter than $\text{chain}_i^t(\text{view})$ given that $t' \geq t$.

Definition 2.3 (Consistency). A blockchain protocol Π satisfies T_0-*consistency*, in Γ-environments if for all Γ-compliant p.p.t. pair (A, Z), there exists some negligible function negl such that for every $\kappa \in \mathbb{N}$ and every $T \geq T_0$ the following holds:

$$\Pr\left[\text{view} \leftarrow \text{EXEC}^{\Pi}(A, Z, \kappa) : \text{consistent}^T(\text{view}) = 1\right] \geq 1 - \text{negl}(\kappa)$$

Note that a direct consequence of consistency is that at any time, the chain *lengths* of any two honest players can differ by at most T (except with negligible probability).

2.6 Security of Nakamoto's Blockchain

The results from [18] (and as we shall shortly see, also ours) are parametrized by the following quantities (which are defined for some fixed mining hardness function $p(\cdot)$; recall that Nakamoto's protocol is parametrized a *single* hardness parameter p):

- let $\alpha := 1 - (1-p)^{(1-\rho)n}$. That is, α is the probability that *some* honest player succeeds in mining a block in a round;
- let $\beta := \rho np$. That is β is the expected number blocks that an attacker can mine in a round.
- let $\gamma := \frac{\alpha}{1+\Delta\alpha}$. γ is a "discounted" version of α which takes into account the fact that messages sent by honest parties can be delayed by Δ rounds and this may lead to honest players "redoing work"; γ corresponds to their "effective" mining power.

In essence, the quantities capture the per round expected "chain length increase" by the honest parties and the adversary; the reason that α, β are defined differently is that we assume that the adversary can sequentialize its queries in a round, whereas honest players make a single parallel query (they each act independently), and thus even if they manage to mine several blocks, the longest chain held by honest players can increase by at most 1. Note, however, that when p is small (in comparison to $1/n$), which is case for the Bitcoin protocol, α is well approximated by $(1-\rho)np$ and thus $\frac{\alpha}{\beta} \approx \frac{1-\rho}{\rho}$, so this difference is minor; additionally, when p is small, $\gamma \approx \alpha$ and thus $\frac{\gamma}{\beta} \approx \frac{1-\rho}{\rho}$.

Compliant executions for Nakamoto's blockchain. We now specify the compliance predicate $\Gamma^{p}_{\text{nak}}(\cdot,\cdot,\cdot)$ for the Nakamoto blockchain. We say that $\Gamma^{p}_{\text{nak}}(\cdot,\cdot,\cdot) = 1$ iff there is a constant $\lambda > 1$ such that

$$\alpha(1 - 2(\Delta + 1)\alpha) \geq \lambda\beta$$

where α and β are functions of the parameters n, ρ, Δ and κ as defined above.

Formal guarantees of Nakamoto's blockchain. The following theorem was proven in [18].

Theorem 2.4 (Security of Nakamoto [18]). *For any constant $\delta > 0$, any $0 < p < 1$, any superlogarithmic function $T_0 = \omega(\log \kappa)$ Nakamoto's blockchain protocol $\Pi_{\text{nak}}(p)$ satisfies the following properties in Γ^{p}_{nak}-environments:*

- *T_0-consistency;*
- *chain growth rate (T_0, g_0, g_1) where $g_0 = (1-\delta)\gamma$, $g_1 = (1+\delta)np$*
- *chain quality (T_0, μ) where $\mu = 1 - (1+\delta)\frac{\beta}{\gamma}$*

Remark 2.5 (Blockchain quality and consistency). *The consistency property proven in [18] is actually a bit stronger than stated. Not only it shows that players agree on the* records *contained in their blockchains, but also that the actual* blockchains *agree except for potentially the last κ blocks. We refer to this property as* blockchain consistency*, and will rely on it in the sequel.*

Additionally, the chain quality property is also stronger in that not only the records of honest blocks are contributed by honest players, but also the actual blocks are mined by honest players. We refer to this property as blockchain quality*, and will rely on it in the sequel.*

2.6.1 Liveness. The liveness property from [18] (which generalized the one from [8]), stipulates that from any given round r, if a sufficiently long period of time t elapses—we refer to this time as the *wait-time* of the blockchain—*every* honest player will have a record m sufficiently "deep" in their chain (technically, κ blocks from the end of the chain), where m was provided as an input to some honest player between rounds r and $r + t$[5] More precisely, let $\mathsf{live}(\mathsf{view}, t) = 1$ iff for any t consecutive rounds $r, \ldots, r + t$ in view there exists some round r' s.t. $r \leq r' \leq r + t$ and player i such that in view, 1) i is honest at r', 2) i received a record m as input at round r', and 3) for every player j that is honest at $r + t$ in view, $\mathsf{m} \in \mathsf{chain}^{r+t}_{j}(\mathsf{view})[:-\kappa]$.

Definition 2.6. We say that blockchain (Π, chain) satisfies *liveness with wait-time w* in Γ-environments if for all Γ-compliant p.p.t. pair (A, Z), there exists a negligible function ϵ in the security parameter $\kappa \in \mathbb{N}$, such that

$$\Pr\left[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(A, Z, \kappa) : \mathsf{live}(\mathsf{view}, w) = 1\right] \geq 1 - \epsilon(\kappa)$$

The following theorem was shown in the prior work [18].

Theorem 2.7 ([18]). *For any boolean predicate $\Gamma(\cdot,\cdot,\cdot)$, let Π be a blockchain protocol satisfying chain growth (T_0, g_0, g_1), chain quality (T_0, μ) and T_0-chain consistency in Γ-environments where μ and g_0 are strictly positive, and T_0 is sublinear.*[6] *Then, Π satisfies liveness with wait-time $w = (1+\delta)\frac{\kappa}{g_0}$ against in Γ-environments*

As a direct corollary of 2.7 and 2.4, we get:

Corollary 2.8 ([18]). *For any constant $\delta > 0$, and any $0 < p < 1$, $\Pi_{\text{nak}}(p)$ satisfies liveness with wait-time*

$$w = (1+\delta)\frac{\kappa}{\gamma}$$

in Γ^{p}_{nak}-environments.

3 DEFINING FAIRNESS

We turn to defining our notion of fairness. Roughly speaking, a blockchain protocol is *δ-approximately fair* w.r.t. ρ attackers if, with overwhelming probability, any honest subset controlling ϕ fraction of the compute power is guaranteed to get at least a $(1-\delta)\phi$ fraction of the blocks in every sufficiently long window, even in the presence of an adversary controlling a ρ fraction of the computation power. Note that this condition trivially implies $(1-\delta)(1-\rho)$ chain quality (by considering $\phi = 1 - \rho$, that is, the full set of honest players). Consequently, to formally define this notion, we first generalize the definition of quality (used in the definition of chain quality, see Definition 2.2) to consider "quality" w.r.t to any subset S of the honest players.

Warmup: fairness definition for static corruption. As a warmup, let us consider how to define (approximate) fairness in a static corruption model where the adversary must declare corrupt nodes upfront — once we show how to do this, we then discuss how to extend the definition to an adaptive corruption model. Under a static corruption model, we say that a blockchain protocol satisfies (T, δ)-approximate fairness, iff the following holds except with negligible probability over the protocol's execution: for any honest node's chain during the protocol execution, for any constant $\phi > 0$, for any subset S of honest users such that $|S| = \phi \cdot n$ where n denotes the total number of users, for any T consecutive blocks $\mathsf{chain}[j+1..j+T]$ in chain, it holds that the fraction of blocks in $\mathsf{chain}[j+1..j+T]$ contributed by nodes in S is at least $(1-\delta)\phi$.

Fairness definition for adaptive corruption. In general, the corruptions can be declared in an adaptive fashion, therefore nodes in any subset S may become corrupt during the course of the window we care about. To define (approximate) fairness with adaptive corruptions, we need to allow the subset S to change over time. We formalize the definition below.

- Let a *player subset selection*, $S(\mathsf{view}, r)$, be a function that given (view, r) outputs a subset of the players that are honest at round r in view.
- We say that S is a *ϕ-fraction player subset selection* if $S(\mathsf{view}, r)$ always outputs a set of size ϕn (rounded upwards) where n is the number of players in view.
- Given a player subset selection S, we say that a *record m is S-compatible w.r.t. view and prefix* chain if there exists a player j

[5] The weaker liveness property from [8] only requires this is *all* honest players have m as their input; this weaker property is not enough for our purposes.

[6] [18] only explicitly considered the case when T_0 is some slightly super-logarithmic function, but their proof actually only assumes that T_0 is sublinear. We also remark that any blockchain protocol which satisfies the security properties w.r.t. to a polynomial T_0 which potentially is super-linear can always be modified to satisfy security w.r.t. a sublinear T_0 by redefining the security parameter.

and round r' such that j is in $S(\mathsf{view}, r')$, the environment provided m as an input to j at round r', and $\mathsf{chain} \prec \mathsf{chain}_i^{r'}(\mathsf{view})$ where $\prec$ denotes "is a prefix of";

- Let $\mathsf{quality}^{T,S}(\mathsf{view}, \mu) = 1$ iff for every round r and every player i such that i is honest in round r of view, we have that among any consecutive sequence of T records $\mathsf{chain}_i^r(\mathsf{view})[j+1 : j+T]$, the fraction of records that are S-compatible w.r.t. view and prefix $\mathsf{chain}_i^r(\mathsf{view})[:j]$ is at least μ.

We now define fairness analogously to chain quality.

Definition 3.1. A blockchain protocol Π has (approximate) *fairness* (T_0, δ) in Γ-environments, if for all Γ-compliant p.p.t. (A, Z), every positive constant $\phi \leq 1-\rho$, every ϕ-fraction subset selection S, there exists some negligible function ϵ such that for every $\kappa \in \mathbb{N}$ and every $T \geq T_0$ the following holds:

$$\Pr\left[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(A, Z, \kappa) : \mathsf{quality}^{T,S}(\mathsf{view}, (1-\delta)\phi)) = 1\right] \geq 1-\epsilon(\kappa)$$

As a sanity check, note that the definition of $\mathsf{quality}^{T,S}(\mathsf{view}, \mu)$ collapses down to $\mathsf{quality}^{T}(\mathsf{view}, \mu)$ if S is the full set of the honest players. As a consequence, (T_0, δ)-fairness trivially implies $(T_0, (1-\delta)(1-\rho))$-chain quality (by considering $\phi = 1-\rho$). Additionally, when $\rho \leq \frac{1}{2}$ which is the case we consider in this paper,

$$\begin{aligned}(1-\delta)(1-\rho) &= 1-\delta-\rho+\delta\rho = 1-[\delta+(1-\delta)\rho] \\ &\geq 1-[2\delta\rho+(1-\delta)\rho] = 1-(1+\delta)\rho\end{aligned}$$

Thus, no ρ-size coalition can get more than a factor $(1+\delta)$ more than its "fair" share of blocks.

Fact 3.2. *If a blockchain protocol Π satisfies (T_0, δ)-fairness in Γ-environments, then it satisfies (T_0, μ)-chain quality where $\mu = (1-\delta)(1-\rho) \geq 1-(1+\delta)\rho$ in Γ-environments.*

4 THE FRUITCHAIN PROTOCOL

We now turn to formally defining our FruitChain protocol. Roughly speaking, the FruitChain protocol will be running an instance of $\Pi_{\mathrm{nak}}(p)$ but instead of directly putting the records m inside the blockchain, the records are put inside "fruits" denoted f; these fruits themselves requires solving some proof of work—with a *different hardness parameter* p_f; additionally, we require a fruit to "hang" from a block which isn't too far from from the block which records the fruit—more specifically, the fruit needs to "point" to an earlier block in the chain which is not too far from the block containing it (and thus, the fruit could not have been mined "too" long ago); the *recency parameter* R will be used to specify how far back a fruit is allowed to hang.

4.1 Valid Blocks, Fruits, and Blockchain

Towards formalizing the protocol, we first introduce some notation:

- We assume that the random oracle H outputs strings of length at least 2κ. Let d be a collision-resistant hash-function (technically, it is a family of functions, and the instance from the family is selected as a public-parameter; in the sequel we ignore this selection and simply treat it as a single function (for instance, selected using randomness $H(0)$.)
- Our protocol is parametrized by two "hardness" parameters $p = p, p_f = p_f$, and a recency parameter R. (p is the mining hardness parameter for the underlying Nakamoto blockchain, and p_f is the "fruit mining" hardness parameter, as mentioned above, the recency parameter will specify how far back a fruit is allowed to "hang"); the quantity $q = \frac{p_f}{p}$ will be useful in our analysis.

Valid fruits. A fruit is of the format $f = (h_{-1}; h'; \eta, \mathsf{digest}; \mathsf{m}; h)$ where each entry means the following:

- h_{-1} points to the previous block's reference — this entry is an artifact of the fruit mining and block mining piggybacked on top of each other; a fruit actually does not care about this entry (but a block does). However the value still needs to be included for the fruit to be verified;
- h' points to a (recently stablized) block that the fruit is hanging from — we call h' the pointer of the fruit f;
- η is a random nonce denoting the puzzle solution;
- digest is the digest of some fruit-set F — this is an artifact since the fruit mining and block mining are piggybacked on top of each other. The block must contain a set of fruits denoted F, but the fruit does not care about the fruit-set, and therefore we include only its d that is necessary for checking that the fruit is correct;
- m is the record to be contained in the fruit; and
- h is the hash or reference of the fruit.

We say that a *fruit* denoted $f = (h_{-1}; h'; \eta, \mathsf{digest}; \mathsf{m}; h)$ is valid iff

(i) $H(h_{-1}; h'; \eta; \mathsf{digest}; \mathsf{m}) = h$;
(ii) $[h]_{-\kappa:} < D_{p_f}$ where $[h]_{-\kappa:}$ denotes the last κ bits of h.

We say that F *is a valid fruit-set* if either $F = \emptyset$ or F is a set of valid fruits.

Valid blocks. Since the block mining and the fruit mining are piggybacked on top of each other, a block looks very much like a fruit, except that a block must additionally include the actual fruit-set F. More specifically, a block is of the following format $\mathsf{b} = ((h_{-1}; h'; \eta; \mathsf{digest}; \mathsf{m}; h), F)$ where each entry means the following:

- h_{-1} points to the previous block's reference, this represents the chain that the block extends from;
- h' is an artifact of the fruit mining and block mining piggybacked atop each other; a block actually does not care about this field (but a fruit does), but it still needs to be included for block verification;
- η is a random nonce denoting the puzzle solution;
- digest is the digest of some fruit-set F to be included in the block later;
- m is a record — the block also does not care about this field, and this is an artifact of the two piggybacked mining processes;
- h is called the reference of the block, which is a hash of the previous fields; and
- F is a fruit-set to be included in the block.

We say that a *block* denoted $\mathsf{b} = ((h_{-1}; h'; \eta; \mathsf{digest}; \mathsf{m}; h), F)$, is valid iff

(i) $\mathsf{digest} = \mathsf{d}(F)$ where d is a collision-resistant hash function as mentioned earlier;
(ii) F is a valid fruit-set;
(iii) $H(h_{-1}; h'; \eta, \mathsf{d}(F); \mathsf{m}) = h$;
(iv) $[h]_{:\kappa} < D_{p_1}$ where $[h]_{:\kappa}$ denotes the first κ bits of h.

Valid blockchain. We say that a *chain* is valid iff

- $chain[0] = genesis$ where $genesis := ((0; 0; 0; 0; \perp; \mathsf{H}(0; 0; 0; 0, \perp)), \emptyset)$ is the "genesis" block;
- for all $i \in [\ell]$, $chain[i].h_{-1} = chain[i-1].h$, i.e., each block refers to the previous block's reference;
- for all $i \in [\ell]$, all $f \in chain[i].F$, there exists some $j \geq i - R\kappa$ such that the pointer of f is $chain[j].h$.

Recency of fruits. Finally, we say that the *fruit* f *is* recent *w.r.t. chain* if the pointer of f is the reference of a block in $chain[-R\kappa :]$ (i.e., one of the last $R\kappa$ blocks in *chain*).

4.2 The FruitChain Protocol and Main Theorem

The FruitChain protocol denoted Π_{fruit} is described in Figure 1. Henceforth, we say that $\Gamma_{\text{fruit}}^{p,p_f,R}(n,\rho,\Delta) = 1$ iff $\Gamma_{\text{nak}}^{p}(n,\rho,\Delta) = 1$. Moreover, we assume the following quantities are constants throughout this paper:

$$q := \frac{p_f}{p} = \Theta(1), \qquad R = \Theta(1)$$

We are now ready to state our main theorem.

Theorem 4.1 (Security of FruitChain). *For any constant* $0 < \delta < 1$, *and any* p, p_f, *let* $R = 17$, $\kappa_f = 2qR\kappa$, *and* $T_0 = 5\frac{\kappa_f}{\delta}$. *Then the* FruitChain *protocol denoted* $\Pi_{\text{fruit}}(p, p_f, R)$ *satisfies*

- κ_f*-consistency;*
- *chain growth rate* (T_0, g_0, g_1) *where* $g_0 = (1-\delta)(1-\rho)np_f$, *and* $g_1 = (1+\delta)np_f$.
- *fairness* (T_0, δ).

in $\Gamma_{\text{fruit}}^{p,p_f,R}$*-environments.*

Proof of the main theorem. In the interest of space, we provide the proof of our main theorem in our online full version [19].

5 FROM FAIRNESS TO INCENTIVE COMPATIBILITY

We remark that any secure blockchain protocol that satisfies δ-approximate fairness (where $\delta < 0.3$) w.r.t $T(\kappa)$ length windows can be used as the ledger underlying a cryptocurrency system while ensuring 3δ*-incentive compatibility* if players (i.e. miners) only care about how much money they receive—that is, a miner's utility is the sum of the rewards and transaction fees it receives (potentially times some constant).[7]

Consider a crypto-currency which uses a blockchain protocol as the underlying ledger; we omit a formalization of what this means, but have in mind a system such as Bitcoin where rewards and transaction fees are somehow distributed among the miners of blocks—for instance, recall that in Bitcoin, the miner of a block receives a mining reward as well as all the transaction fees contained in the block it mined.

We say that *honest mining is a* ρ*-coalition-safe* ϵ*-Nash equilibrium* if, with overwhelming probability, no $\rho' < \rho$ fraction coalition can gain more than a multiplicative factor $(1+\epsilon)$ in utility, no matter what transactions are being processed—formally, consider some environment providing transactions into the system. We restrict to a setting where the *total* rewards and transaction fees during the run of the system is some fixed constant V[8].

We now remark that if rewards and transaction fees are *evenly distributed* among the (miners of the) blocks in the $T(\kappa)$-length segment of the chain preceeding the block (and in the initial phase, before the chain is of length $T(\kappa)$, simply the first $T(\kappa)$ blocks) then it follows that honest mining is a ρ-coalition-safe 3δ-Nash equilibrium as long as the underlying blockchain satisfies δ-approximate fairness w.r.t. ρ attackers: as noted above, fairness implies that no matter what deviation the coalition performs, with overwhelming probability, the fraction of adversarial blocks in any $T(\kappa)$-length window of the chain is upperbounded by $(1+\delta)\rho$ and thus the total amount of compensation received by the attacker is bounded by $(1+\delta)\rho \cdot V$; in contrast, by fairness, if the coalition had been following the honest protocol, they are guaranteed to receive at least $(1-\delta)\rho \cdot V$; thus, the multiplicative increase in utility is $\frac{1+\delta}{1-\delta} \leq 1+3\delta$ when $\delta < 0.3$.[9]

To see why the "standard" bitcoin approach of giving all rewards and fees to the miner of the block does not work, consider an freshly mined (honest) block containing a transaction with a very high transaction fee. A coalition controlling a constant fraction of the computing power would have a huge incentive to "drop" this block and instead try to mine a new block which contains it. Fairness does not prevent such an attack, and indeed, even in our protocol such an attack will be successful with constant probability. (Indeed, it has been informally conjectured in the bitcoin community that ϵ-incentive compatibility is impossible to achieve in the presence of transaction fees, due to exactly this reason. Our method of distributing the fees over a segment overcomes this "barrier".)

6 DISINCENTIVIZING POOLED MINING

An issue with the Bitcoin protocol (which relies on Nakamoto's blockchain protocol) is that the mining hardness is set so that the world (combined) finds a new block every 10 minutes—as shown in [18], the mining hardness needs to be set in such a way to ensure consistency. This not only leads to a long latency (which can be remedied by the Hybrid Consensus approach discussed above), but also leads to the issue that it may take a very long time for an individual miner to be successful in mining a block and consequently reap a reward for its work. In other words, the payments received by miners has a very *high variance*. This has lead to the creation of mining pools, where miners come together and pool their work and then share the reward once someone in the pool mines a block—such pooling decreases the variance. To prevent free-riding, miners submit "partial proofs of work" (that is, "near" solutions to the mining puzzles) that are significantly easier to find, and rewards are distributed (according to some distribution rule) among the contributors of the partial proofs-of-work.

[7] This may not always be a realistic assumption. For instance, a miner can care about what transactions get added into the blockchain etc, but following earlier approaches to modeling incentives in blockchains (e.g., [7]), we focus only on miners' monetary rewards.

[8] The analysis directly extends to a setting where the total rewards and fees are only guaranteed to be withing some multiplicative factor $(1+\delta')$ of V at the cost of a degradation of the quality of the Nash equilibrium (i.e., increasing the ϵ).

[9] Let us remark that an alternative approach would be to give the whole mining reward to the miner of a block (as in Bitcoin) but still distribute the transaction fees among the group of miners in a $T(\kappa)$-segment of the chain. This approach works by the same analysis as long as mining rewards are *fixed* throughout the experiment (which is not the case for e.g., Bitcoin where mining rewards decrease over time).

$\Pi_{\text{fruit}}(p, p_f, R)$: FruitChain **protocol**

Initialize: $chain := genesis$, $F = \emptyset$

Upon receiving a valid *fruit*: let $F := F \cup \{fruit\}$

Upon receiving a valid $chain'$, if $|chain'| > |chain|$: let $chain := chain'$

Every time step, upon receiving input m from the environment:

- let F' be all fruits $f \in F$ that are recent w.r.t. *chain* and not already in *chain*;
- let h' be the reference of $chain[pos]$ where $pos = max(1, |chain| - \kappa)$;
- let h_{-1} be the reference of $chain[-1]$;
- Pick random $\eta \in \{0,1\}^\kappa$ and let $h := \mathsf{H}(h_{-1}; h'; \eta; \mathsf{d}(F'); \mathsf{m})$
- If $[h]_{-\kappa:} < D_{p_f}$ (i.e., we "mined a fruit"): let $fruit := (h_{-1}; h'; \eta; \mathsf{d}(F'); \mathsf{m}, h)$, $F := F \cup \{fruit\}$, and broadcast *fruit*
- If $[h]_{:\kappa} < D_p$ (i.e., we "mined a block"): let $chain := chain || ((h_{-1}; h'; \eta, \mathsf{d}(F'); \mathsf{m}, h), F')$, and broadcast *chain*
- Output $\text{extract}_{\text{fruit}}(chain)$ to Z, where $\text{extract}_{\text{fruit}}(\cdot)$ is defined as below:

$\text{extract}_{\text{fruit}}(chain)$:

On input a valid *chain*,

- first extract a sequence of *distinct* fruits from *chain*, where if the same fruit is included multiple times, only the first occurrence is included. The extracted fruits are ordered by the first block that contains the fruit; and for fruits in the same block, follow the order in which the fruits are serialized within the block.
- then, output the sequence of records m contained in the extracted sequence of fruits, where records contained in earlier fruits are extracted earlier.

Figure 1: The FruitChain **protocol. Nodes not only mine for blocks, but also fruits. Blocks confirm "recent" fruits; whereas fruits confirm transactions.**

An undesirable effect of such pools is that the pool operator effectively controls a large number of participants and potentially could get them to deviate; in a sense, the decentralized nature of the system gets lost.

We note that since the FruitChain protocol is parametrized by *two* mining hardnesses—the block hardness p, and the fruit hardness p_f—which are independent of each other, we can set p appropriately to ensure consistency, but p_f can be set to be much larger—for instance, as large as the probability of find a partial proof-of-work in mining pools—and consequently, we would reduce the variance of the rewards received by miners in exactly the same way as in mining pool, but now in a *fully decentralized* way.

Today, a solo miner (assuming one unit of typical commodity mining ASIC) would take 2 to 5 years to obtain its first reward [2]. With FruitChain, suppose we allocate space for 1000 fruits per block where each fruit is 80 bytes (same size as a Bitcoin puzzle solution), this would occupy roughly 8% of a 1MB block — however, this would allow a solo miner to get its first rewards 1000x faster, roughly on the order of a day (or days) rather than years.

Acknowledgments. This work is supported in part by NSF grants CNS-1217821, CNS-1314857, CNS-1514261, CNS-1544613, CNS-1561209, CNS-1601879, CNS-1617676, AFOSR Award FA9550-15-1-0262, an Office of Naval Research Young Investigator Program Award, a Microsoft Faculty Fellowship, a Packard Fellowship, a Sloan Fellowship, Google Faculty Research Awards, a VMWare Research Award, and a Baidu Research Award.

REFERENCES

[1] 2016. Personal Communication with Iddo Bentov and Yuncong Hu and Siqiu Yao. (2016).
[2] 2017. http://www.coinwarz.com/calculators/bitcoin-mining-calculator. (2017).
[3] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. 2005. Secure Computation Without Authentication. In *CRYPTO'05*.
[4] Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. 2016. On the Instability of Bitcoin Without the Block Reward. In *CCS*. 154–167.
[5] Phil Daian, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs of Stake. Cryptology ePrint Archive, Report 2016/919. (2016).
[6] Cynthia Dwork and Moni Naor. 1992. Pricing via processing or combatting junk mail. In *CRYPTO'92*. 139–147.
[7] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*. Springer, 436–454.
[8] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*.
[9] Joseph Y. Halpern and Rafael Pass. 2015. Algorithmic rationality: Game theory with costly computation. *J. Economic Theory* 156 (2015), 246–268.
[10] Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. 2016. Blockchain Mining Games. In *EC*. 365–382.
[11] Aggelos Kiayias and Giorgos Panagiotakos. 2015. Speed-Security Tradeoffs in Blockchain Protocols. (2015).
[12] Aggelos Kiayias and Giorgos Panagiotakos. 2016. On Trees, Chains and Fast Transactions in the Blockchain. *IACR Cryptology ePrint Archive* 2016 (2016), 545.
[13] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2016. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. Cryptology ePrint Archive, Report 2016/889. (2016). http://eprint.iacr.org/2016/889.
[14] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 2015. Inclusive Block Chain Protocols. In *Financial Crypto'15*.
[15] mtgox. 2010. https://bitcointalk.org/index.php?topic=2227. (2010).
[16] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[17] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. 2016. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In *EuroS&P*.
[18] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *Eurocrypt*.
[19] Rafael Pass and Elaine Shi. 2016. Fruitchains: A Fair Blockchain. Online Technical Report, https://eprint.iacr.org/2016/916.pdf. (2016).
[20] Rafael Pass and Elaine Shi. 2016. Hybrid Consensus. http://eprint.iacr.org/2016/917. (2016).
[21] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. 2016. Optimal Selfish Mining Strategies in Bitcoin. In *Financial Crypto'16*.
[22] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure High-Rate Transaction Processing in Bitcoin. In *FC*. 507–527.