But Why Does it Work? A Rational Protocol Design Treatment of Bitcoin[⋆]

Christian Badertscher[1], Juan Garay[2], Ueli Maurer[1], Daniel Tschudi[3⋆⋆], and Vassilis Zikas[4⋆⋆⋆]

[1] ETH Zurich, {christian.badertscher,maurer}@inf.ethz.ch
[2] Texas A&M University, garay@tamu.edu
[3] Aarhus University, tschudi@cs.au.dk
[4] University of Edinburgh & IOHK, vassilis.zikas@ed.ac.uk

February 14, 2018

Abstract. An exciting recent line of work has focused on formally investigating the core cryptographic assumptions underlying the security of Bitcoin. In a nutshell, these works conclude that Bitcoin is secure if and only if the majority of the mining power is honest. Despite their great impact, however, these works do not address an incisive question asked by positivists and Bitcoin critics, which is fuelled by the fact that Bitcoin indeed works in reality: Why should the real-world system adhere to these assumptions?
In this work we employ the machinery from the Rational Protocol Design (RPD) framework by Garay *et al.* [FOCS'13] to analyze Bitcoin and address questions such as the above. We show that under the natural class of incentives for the miners' behavior—i.e., rewarding them for adding blocks to the blockchain but having them pay for mining—we can reserve the honest majority assumption as a fallback, or even, depending on the application, completely replace it by the assumption that the miners aim to maximize their revenue.
Our results underscore the appropriateness of RPD as a "rational cryptography" framework for analyzing Bitcoin. Along the way, we devise significant extensions to the original RPD machinery that broaden its applicability to cryptocurrencies, which may be of independent interest.

1 Introduction

Following a number of informal and/or *ad hoc* attempts to address the security of Bitcoin, an exciting recent line of work has focused on devising a rigorous cryptographic analysis of the system [12, 13, 26, 1]. At a high level, these works start by describing an appropriate model of execution, and, within it, an abstraction of the original Bitcoin protocol [22] along with a specification of its security goals in terms of a set of intuitive desirable properties [12, 13, 26], or in terms of a functionality in a simulation-based composable framework [1]. They then prove that (their abstraction of) the Bitcoin protocol meets the proposed specification under the assumption that the majority of the computing power invested in mining bitcoins is by devices which mine according to the Bitcoin protocol, i.e., *honestly.* This assumption of *honest majority* of computing power—which had been a folklore within the Bitcoin community for years underlying the system's security—is captured by considering the parties who are not mining honestly as controlled by a central adversary who coordinates them trying to disrupt the protocol's outcome.

Meanwhile, motivated by the fact that Bitcoin is an "economic good" (i.e., BTCs are exchangeable for national currencies and goods) a number of works have focused on a rational analysis of the system [28, 6, 8, 7, 30, 29, 21, 31, 23, 27, 14]. In a nutshell, these works treat Bitcoin as a game between the (competing) rational miners, trying to maximize a set of utilities that are postulated as a natural incentive structure for the system. The goal of such an analysis is to investigate whether or not, or under which assumptions on

[⋆] This is the full version of the article due to appear at Eurocrypt 2018. The final publication will be available at link.springer.com.
[⋆⋆] Work done while author was at ETH Zurich.
[⋆⋆⋆] Work done in part while the author was at RPI.

the incentives and/or the level of collaboration of the parties, Bitcoin achieves a stable state, i.e., a game-theoretic equilibrium. However, despite several enlightening conclusions, more often than not the prediction of such analyses is rather pessimistic. Indeed, these results typically conclude that, unless assumptions on the amount of honest computing power—sometimes even stronger than just majority—are made, the induced incentives result in plausibility of an attack to the Bitcoin mining protocol, which yields undesired outcomes such as forks on the blockchain, or a considerable slowdown.

Yet, to our knowledge, no fork or substantial slowdown that is attributed to rational attacks has been observed to date, and the Bitcoin network keeps performing according to its specification, even though mining pools would, in principle, be able to launch collaborative attacks given the power they control.[5] In the game-theoretic setting, this mismatch between the predicted and observed behavior would be typically interpreted as an indication that the underlying assumptions about the utility of miners in existing analysis do not accurately capture the miners' rationale. Thus, two main questions still remain and are often asked by Bitcoin skeptics:

Q1. How come Bitcoin is not broken using such an attack?

Or, stated differently, why does it work and why do majorities not collude to break it?

Q2. Why do honest miners keep mining given the plausibility of such attacks?

In this work we use a rigorous cryptographic reasoning to address the above questions. In a nutshell, we devise a rational-cryptography framework for capturing the economic forces that underly the tension between honest miners and (possibly colluding) deviating miners, and explain how these forces affect the miners' behavior. Using this model, we show how natural incentives (that depend on the expected revenue of the miners) in combination with a high monetary value of Bitcoin, can explain the fact that Bitcoin is not being attacked in reality *even though* majority coalitions are in fact possible. In simple terms, we show how natural assumptions about the miners' incentives allow to substitute (either entirely or as a fallback assumption) the honest-majority assumption. To our knowledge, this is the first work that formally proves such rational statements that do not rely on assumptions about the adversary's computing power. We stress that the incentives we consider depend solely on costs and rewards for mining—i.e., mining (coinbase) and transaction fees—and, in particular, we make no assumption that implicitly or explicitly deters forming adversarial majority coalitions.

What enables us to address the above questions is utilizing the Rational Protocol Design (RPD) methodology by Garay *et al.* [10] to derive stability notions that closely capture the idiosyncrasies of coordinated incentive-driven attacks on the Bitcoin protocol. To better understand how our model employs RPD to address the above questions, we recall the basic ideas behind the framework.

Instead of considering the protocol participants—in our case, the Bitcoin miners—as rational agents, RPD considers a meta-game, called the *attack game*. The attack game in its basic form is a two-agent zero-sum extensive game of perfect information with a horizon of length two, i.e., two sequential moves.[6] It involves two players, called the *protocol designer* $\mathtt{D}$—who is trying to come up with the best possible protocol for a given (multi-party) task—and the *attacker* $\mathtt{A}$—who is trying to come up with the (polynomial-time) strategy/adversary that optimally attacks the protocol. The game proceeds in two steps: First, (only) $\mathtt{D}$ plays by choosing a protocol for the (honest) players to execute; $\mathtt{A}$ is informed about $\mathtt{D}$'s move and it is now his term to produce his move. The attacker's strategy is, in fact, a cryptographic adversary that attacks the protocol proposed by the designer.

The incentives of both $\mathtt{A}$ and $\mathtt{D}$ are described by utility functions, and their respective moves are carried out with the goal of maximizing these utilities.[7] In a nutshell, the attacker's utility function rewards the adversary proportionally to how often he succeeds in provoking his intended breach, and depending on its severity. Since the game is zero-sum, the designer's utility is the opposite of the attacker's; this captures the standard goal of cryptographic protocols, namely, "taming" the adversary in the best possible manner.

[5] We refer to forks of the Bitcoin chain itself, not to forks that spin-off a new currency.

[6] This is often referred to as a *Stackelberg game* in the game theory literature [25].

[7] Notice, however, the asymmetry: The designer needs to come up with a protocol based on speculation of what the adversary's move will be, whereas the attacker plays after being informed about the actual designer's move, i.e., about the protocol.

Based on the above game, the RPD framework introduces the following natural security notion, termed *attack-payoff security*, that captures the quality of a protocol Π for a given specification when facing incentive-driven attacks aiming to maximize the attacker's utility. Informally, attack-payoff security ensures that the adversary is not willing to attack the protocol Π in any way that would make it deviate from its ideal specification. In other words, the protocol is secure against the class of strategies that maximize the attacker's utility. In this incentive-driven setting, this is the natural analogue of security against malicious adversaries.[8] For cases where attack payoff security is not feasible, RPD proposes the notion of *attack-payoff optimality*, which ensures that the protocol Π is a best response to the best attack.

A useful feature of RPD (see below) is that all definitions build on Canetti's simulation-based framework (either the standalone framework [4] or the UC framework [5]), where they can be easily instantiated. In fact, there are several reasons, both at the intuitive and technical levels, that make RPD particularly appealing to analyze complex protocols that are already running, such as Bitcoin. First, RPD supports adaptive corruptions which captures the scenario of parties who are currently running their (mining) strategy changing their mind and deciding to attack. This is particularly useful when aiming to address the likelihood of insider attacks against a protocol which is already in operation. For the same reason, RPD is also suitable for capturing attacks induced by compromised hardware/software and/or bribing [3] (although we will not consider bribing here). Second, the use of a central adversary as the attacker's move ensures that, even though we are restricting to incentive-driven strategies, we allow full collaboration of cheaters. This allows, for example, to capture mining pools deciding to deviate from the protocol's specification.

At the technical level, using the attack-game to specify the incentives takes away many of the nasty complications of "rational cryptography" models. For example, it dispenses with the need to define cumbersome computational versions of equilibrium [9, 20, 18, 24, 16, 17], since the actual rational agents, i.e., $\mathtt{D}$ and $\mathtt{A}$, are not computationally bounded. (Only their actions need to be PPT machines.) Furthermore, as it builds on simulation-based security, RPD comes with a composition theorem allowing for regular cryptographic subroutine replacement. The latter implies that we can analyze protocols in simpler hybrid-worlds, as we usually do in cryptography, without worrying about whether or not their quality or stability will be affected once we replace their hybrids by corresponding cryptographic implementations.

Our contributions. In this work, we apply the RPD methodology to analyze the quality of Bitcoin against incentive-driven attacks, and address the existential questions posted above. As RPD is UC-based, we use the Bitcoin abstraction as a UC protocol and the corresponding Bitcoin ledger functionality from [1] to capture the goal/specification of Bitcoin. As argued in [1], this functionality captures all the properties that have been proposed in [12, 26].

We define a natural class of incentives for the attacker by specifying utilities which, on one hand, reward him according to Bitcoin's standard reward mechanisms (i.e., block rewards and transaction fees) for blocks permanently inserted in the blockchain by adversarial miners, and, on the other hand, penalize him for resources that he uses (e.g., use of mining equipment and electricity). In order to overcome the inconsistency of rewards being typically in Bitcoins and costs being in real money, we introduce the notion of a *conversion rate* $\mathtt{CR}$ converting reward units (such as BTC) into mining-cost units (such as US Dollar) This allows us to make statements about the quality of the protocol depending on its value measured in a national currency.

We then devise a similar incentive structure for the designer, where, again, the honest parties are (collectively) rewarded for blocks they permanently insert into the blockchain, but pay for the resources they use. What differentiates the incentives of the attacker from the designer's is that the latter is utmost interested in preserving the "health" of the blockchain, which we also reflect in its utility definition. Implicit in our formulation is the assumption that the attacker does not gain reward from attacking the system, unless this attack has a financial gain.[9]

Interestingly, in order to apply the RPD methodology to Bitcoin we need to extend it in non-trivial ways, to capture for example non-zero-sum games—as the utility of the designer and the attacker are not necessarily opposites—and to provide stronger notions of security and stability. In more detail, we introduce

[8] In fact, if we require this for any arbitrary utility function, then the two notions—attack-payoff security and malicious security—coincide.

[9] In particular, a fork might be provoked by the attacker only if it is expected to increase his revenue.

the notion of *strong attack payoff security*, which mandates that the attacker will stick to playing a passive strategy, i.e., stick to Bitcoin (but might abuse the adversary's power to delay messages in the network). We also introduce the natural notion of *incentive compatibility* (IC) which mandates that both the attacker and the designer will have their parties play the given protocol. Observe that incentive compatibility trivially implies strong attack payoff security, and the latter implies the standard attack payoff security from the original RPD framework assuming the protocol is at least correct when no party deviates. These extensions to RPD widen its applicability and might therefore be of independent interest. We note that although we focus on analysis of Bitcoin here, the developed methodology can be adapted to analyze other main-stream cryptocurrencies.

Having laid out the model, we then use it to analyze Bitcoin. We start our analysis with the simpler case where the utilities do not depend on the messages—i.e., transactions—that are included into the blocks of the blockchain: when permanently inserting a block into the blockchain, a miner is just rewarded with a fixed block-reward value. This can be seen as corresponding to the Bitcoin backbone abstraction proposed in [12], but enriched with incentives to mine blocks. An interpretation of our results for this setting, listed below, is that they address blockchains that are not necessarily intended to be used as cryptocurrency ledgers. Although arguably this is not the case for Bitcoin, our analysis already reveals several surprising aspects, namely, that in this setting one does not need to rely on honest majority of computing power to ensure the quality of the system. Furthermore, these results offer intuition on what is needed to achieve stability in the more complete case, which also incorporates transaction fees. Summarizing, we prove the following statements for this backbone-like setting, where the contents of the blocks do not influence the player's strategies (but the rewards and costs do):

- Bitcoin is strongly attack-payoff secure, i.e., no coordinated coalition has an incentive to deviate from the protocol, provided that the rest of the parties play it. Further, this statement holds no matter how large the coalition (i.e., no matter how large the fraction of corrupt computing power) and no matter how high the conversion rate is. This means that in this backbone-like setting we can fully replace the assumption of honest majority of computing power by the above intuitive rational assumption.[10]
- If the reward for mining a block is high enough so that mining is on average profitable, then the Bitcoin protocol is even incentive-compatible with respect to local deviations. In other words, not only colluding parties (e.g., mining pools) do not have an incentive to deviate, but also the honest miners have a clear incentive to keep mining. Again, this makes no honest-majority assumption. Furthermore, as a sanity check, we also prove that this is not true if the conversion rate drops so that miners expect to be losing revenue by mining. The above confirms the intuition that after the initial bootstrapping phase where value is poured into the system (i.e., CR becomes large enough), such a ledger will keep working according to its specification for as long as the combination of conversion rate and block-reward is high enough.

With the intuition gained from the analysis in the above idealized setting, we next turn to the more realistic setting which closer captures Bitcoin, where block contents are messages that have an associated fee. We refer to these messages as *transactions*, and use the standard restrictions of Bitcoin on the transaction fees: every transaction has a maximum fee and the fee is a multiple of the minimum division.[11] We remark that in all formal analyses [12, 26, 1] the transactions are considered as provided as inputs by an explicit environment that is supposed to capture the application layer that sits on top of the blockchain and uses it. As such, the environment will also be responsible for the choice of transaction fees and the distribution of transactions to the miners. For most generality, we do not assume as in [12, 26] that all transactions are communicated by the environment to all parties via a broadcast-like mechanism, but rather that they are distributed (i.e., input) by the environment to the miners, individually, who might then forward them using the network (if they are honest) or not. This more realistic transaction-submission mechanism is already explicit in [1].

We call this model that incorporates both mining rewards and transaction fees into the reward of the miner for a block as the *full-reward* model. Interestingly, this model allows us to also make predictions about

[10] It should be noted though that our analysis considers, similarly to [12, 26, 1], a fixed difficulty parameter. The extension to variable difficulty is left as future research.

[11] For Bitcoin the minimum division is 1 satoshi = 10^{-8} BTC, and there is typically a cap on fees [2].

the Bitcoin era when the rewards for mining a block will be much smaller than the transaction fees (or even zero).

We stress that transactions in our work are dealt with as messages that have an explicit fee associated with them, rather than actions which result in transferring BTCs from one miner to another. This means that other than its associated fee, the contents of a transaction does not affect the strategies of the players in the attack game. This corresponds to the assumption that the miners, who are responsible for maintaining the ledger, are different than the users, which, for example, translate the contents of the blocks as exchanges of cryptocurrency value, and which are part of the application/environment. We refer to this assumption as *the miners/users separation principle.* This assumption is explicit in all existing works, and offers a good abstraction to study the incentives for maintaining the ledger—which is the scope of our work—separately from the incentives of users to actually use it. Note that this neither excludes nor trivially deters “forking” by a sufficiently powerful (e.g., 2/3 majority) attacker; indeed, if some transaction fees are much higher than all others, then such an attacker might fork the network by extending both the highest and the second highest chain with the same block containing these high-fee transactions, and keep it forked for sufficiently long until he cashes out his rewards from both forks.

In this full-reward model, we prove the following statements:

- First, we look at the worst-case environment, i.e., the one that helps the adversary maximize its expected revenue. We prove that in this model Bitcoin is still incentive compatible, hence also strongly attack payoff secure. In fact, the same is true if the environment makes sure that there is a sufficient supply of transactions to the honest miners and to the adversary, such that the fees are high enough to build blocks that reach exactly the maximal rewarding value (note that not necessarily the same set of transactions have to be known to the participants). For example, as long as many users submit transactions with the heaviest possible fee (so-called *full-fee transactions*), then the system is guaranteed to work without relying on an honest majority of miners. In a sense, the users can control the stability of the system through transaction fees.
- Next, we investigate the question of whether or not the above is true for arbitrary transaction-fee distributions. Not surprisingly, the answer here is negative, and the protocol is not even attack-payoff secure (i.e, does not even achieve its specification). The proof of this statement makes use of the above sketched forking argument. On the positive side, our proof suggests that in the honest-majority setting where forking is not possible (except with negligible probability), the only way the adversary is incentivized to deviate from the standard protocol is to withhold the transactions he is mining on to avoid risking to lose the fees to honest parties.

Interpreting the above statements, we can relax the assumption for security of Bitcoin from requiring an honest majority to requiring long-enough presence of sufficiently many full-fee transactions, with a fallback to honest majority.

- Finally, observing that the typically large pool of transactions awaiting validation justifies the realistic assumption that there is enough supply to the network (and given the high adoption, this pool will not become small too fast), we can directly use our analysis, to propose a possible modification which would help Bitcoin, or other cryptocurrencies, to ensure incentive compatibility (hence also strong attack-payoff security) in the full-reward model in the long run: The idea is to define an exact cumulative amount on fees (or overall reward) to be allowed for each block. If there are enough high-fee transactions, then the blocks are filled up with transactions until this amount is reached. As suggested by our first analysis with a simple incentive structure, ensuring that this cap is non-decreasing would be sufficient to argue about stability; however, it is well conceivable that such a bound could be formally based on supply-and-demand in a more complex and economy-driven incentive structure and an interesting future research direction is to precisely define such a proposal together with the (economical) assumptions on which the security statements are based. We note that the introduction of such a rule would typically only induce a “soft fork,” and would, for a high-enough combination of conversion rate and reward bound, ensure incentive compatibility even when the flat reward per block tends to zero and the main source of rewards would be transaction fees, as it is the plan for the future of Bitcoin.

2 Preliminaries

In this section we introduce some notation and review the basic concepts and definitions from the literature, in particular from [10] and [1] that form the basis of our treatment. For completeness, an expanded version of this review can be found in Appendix A. Our definitions use and build on the simulation-based security definition by Canetti [5]; we assume some familiarity with its basic principles.

Throughout this work we will assume an (at times implicit) security parameter κ. We use $\mathtt{ITM}$ to the denote the set of *probabilistic polynomial time (PPT)* interactive Turing machines (ITMs). We also use the standard notions of *negligible, noticeable*, and *overwhelming* (e.g., see [15]) were we denote negligible (in κ) functions as $\mathrm{negl}(\kappa)$. Finally, using standard UC notation we denote by $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ (resp. $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$) the random variable (ensemble if indexed by κ) corresponding to the output of the environment $\mathcal{Z}$ witnessing an execution of protocol Π against adversary $\mathcal{A}$ (resp. an ideal evaluation of functionality $\mathcal{F}$ with simulator $\mathcal{S}$).

2.1 The RPD Framework

The RPD framework [10] captures incentive-driven adversaries by casting attacks as a *meta-game* between two rational players, the protocol designer $\mathtt{D}$ and the attacker $\mathtt{A}$, which we now describe. The game is parameterized by a (multi-party) functionality $\mathcal{F}$ known to both agents $\mathtt{D}$ and $\mathtt{A}$ which corresponds to the ideal goal the designer is trying to achieve (and the attacker to break). Looking ahead, when we analyze Bitcoin, $\mathcal{F}$ will be a ledger functionality (cf. [1]). The designer $\mathtt{D}$ chooses a PPT protocol Π for realizing the functionality $\mathcal{F}$ from the set of all probabilistic and polynomial-time (PPT) computable protocols.[12] $\mathtt{D}$ sends Π to $\mathtt{A}$ who *then* chooses a PPT adversary $\mathcal{A}$ to attack protocol Π. The set of possible terminal histories is then the set of sequences of pairs $(\Pi, \mathcal{A})$ as above.

Consistently with [10], we denote the corresponding attack game by $\mathcal{G}_{\mathcal{M}}$, where $\mathcal{M}$ is referred to as the *attack model*, which specifies all the public parameters of the game, namely: (1) the functionality, (2) the description of the relevant action sets, and (3) the utilities assigned to certain actions (see below).

Stability in RPD corresponds to a refinement of a *subgame-perfect equilibrium* (cf. [25, Definition 97.2]), called *ϵ-subgame perfect equilibrium*, which considers as solutions profiles in which the parties' utilities are ϵ-close to their best-response utilities (see [10] for a formal definition). Throughout this paper, we will only consider $\epsilon = \mathrm{negl}(\kappa)$; in slight abuse of notation, we will refer to $\mathrm{negl}(\kappa)$-*subgame perfect equilibrium* simply as subgame perfect.

The utilities. The core novelty of RPD is in how utilities are defined. Since the underlying game is zero-sum, it suffices to define the attacker's utility. This utility depends on the goals of the attacker, more precisely, the security breaches which he succeeds to provoke, and is defined, using the simulation paradigm, via the following three-step process:

First, we modify the ideal functionality $\mathcal{F}$ to obtain a (possibly weaker) ideal functionality $\langle\mathcal{F}\rangle$, which explicitly allows the attacks we wish to model. For example, $\langle\mathcal{F}\rangle$ could give its simulator access to the parties' inputs. This allows to score attacks that aim at input-privacy breaches.

Second we describe a scoring mechanism for the different breaches that are of interest to the adversary. Specifically, we define a function $v_{\mathtt{A}}$ mapping the joint view of the relaxed functionality $\langle\mathcal{F}\rangle$ and the environment $\mathcal{Z}$ to a real-valued *payoff*. This mapping defines the random variable (ensemble) $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ as the result of applying $v_{\mathtt{A}}$ to the views of $\langle\mathcal{F}\rangle$ and $\mathcal{Z}$ in a random experiment describing an ideal evaluation with ideal-world adversary $\mathcal{S}$; in turn, $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ defines the *attacker's (ideal) expected payoff* for simulator $\mathcal{S}$ and environment $\mathcal{Z}$, denoted by $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})$, so the expected value of $v^{\mathtt{A}}_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$. The triple $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v^{\mathtt{A}})$ constitutes the *attack model.*

The third and final step is to use $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})$ to define the attackers utility, $u_{\mathtt{A}}(\Pi,\mathcal{A})$, for playing an adversary $\mathcal{A}$ against protocol Π, as the expected payoff of the "best" simulator that successfully simulates $\mathcal{A}$ in its ($\mathcal{A}$'s) favorite environment. This best simulator is the one that translates the adversary's breaches

[12] Following standard UC convention, the protocol description includes its hybrids.

against Π into breaches against the relaxed functionality $\langle\mathcal{F}\rangle$ in a faithful manner, i.e., so that the ideal breaches occur only if the adversary really makes them necessary for the simulator in order to simulate. As argued in [10], this corresponds to the simulator that minimizes the attacker's utility. Formally, for a functionality $\langle\mathcal{F}\rangle$ and a protocol Π, denote by $\mathcal{C}_{\mathcal{A}}$ the class of simulators that are "good" for $\mathcal{A}$, i.e, $\mathcal{C}_{\mathcal{A}} = \{\mathcal{S} \in \texttt{ITM} \mid \forall \mathcal{Z} : \text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}\}$.[13] Then the attacker's (expected) utility is defined as:

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z}\in\texttt{ITM}} \left\{ \inf_{\mathcal{S}\in\mathcal{C}_{\mathcal{A}}} \left\{ U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S}, \mathcal{Z}) \right\} \right\}.$$

For $\mathcal{A}$ and Π with $\mathcal{C}_{\mathcal{A}} = \emptyset$, the utility is ∞ by definition, capturing the fact that we only want to consider protocols which at the very least implement the relaxed (i.e., explicitly breachable) functionality $\langle\mathcal{F}\rangle$. Note that as the views in the above experiments are in fact random variable ensembles indexed by the security parameter κ, the probabilities of all the relative events are in fact functions of κ, hence the utility is also a function of κ. Note also that as long as $\mathcal{C}_{\mathcal{A}} = \emptyset$ is non-empty, for each value of κ, both the supremum and the infimum above exist and are finite and reachable by at least one pair $(\mathcal{S}, \mathcal{Z})$, provided the scoring function assigns finite payoffs to all possible transcripts (for $\mathcal{S} \in \mathcal{C}_{\mathcal{A}}$) (cf. [10]).

Remark 1 (Event-based utility [10]). In many applications, including those in our work, meaningful payoff functions have the following, simple representation: Let $(E_1, \ldots, E_\ell)$ denote a vector of (typically disjoint) events defined on the views (of $\mathcal{S}$ and $\mathcal{Z}$) in the ideal experiment corresponding to the security breaches that contribute to the attacker's utility. Each event E_i is assigned a real number γ_i, and the payoff function $v_{\mathtt{A}}^{\vec{\gamma}}$ assigns, to each ideal execution, the sum of γ_i's for which E_i occurred. The ideal expected payoff of a simulator is computed according to our definition as

$$U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S}, \mathcal{Z}) = \sum_{E_i\in\vec{E},\gamma_i\in\vec{\gamma}} \gamma_i \Pr[E_i],$$

where the probabilities are taken over the random coins of $\mathcal{S}$, $\mathcal{Z}$, and $\langle\mathcal{F}\rangle$.

Building on the above definition of utility, [10] introduces a natural notion of security against incentive-driven attackers. Intuitively, a protocol Π is *attack-payoff secure* in a given attack model $\mathcal{M} = (\mathcal{F}, \cdot, v_{\mathtt{A}})$, if the utility of the best adversary against this protocol is the same as the utility of the best adversary in attacking the $\mathcal{F}$-hybrid "dummy" protocol, which only relays messages between $\mathcal{F}$ and the environment.

Definition 1 (Attack-payoff security [10]). *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$ be an attack model inducing utilities $u_{\mathtt{A}}$ and $u_{\mathtt{D}}$ on the attacker and the designer, respectively,*[14] *and let $\phi^{\mathcal{F}}$ be the dummy $\mathcal{F}$-hybrid protocol. A protocol Π is* attack-payoff secure for $\mathcal{M}$ *if for all adversaries $\mathcal{A} \in \texttt{ITM}$,*

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u_{\mathtt{A}}(\phi^{\mathcal{F}}, \mathcal{A}) + \text{negl}(\kappa).$$

Intuitively, this security definition accurately captures security against an incentive-driven attacker, as in simulating an attack against the dummy $\mathcal{F}$-hybrid protocol, the simulator never needs to provoke any of the "breaching" events. Hence, the utility of the best adversary against Π equals the utility of an adversary that does not provoke any "bad event."

2.2 A Composable Model for Blockchain Protocols

In [1], Badertscher *et al.* present a universally composable treatment of the Bitcoin protocol, Π^{B}, in the UC framework. Here we recall the basic notions, the notation, and some results.

[13] This class is finite for every given value of the security parameter, Π, and $\mathcal{A}$.
[14] In [10], by default $u_{\mathtt{D}} = -u_{\mathtt{A}}$ as the game is zero-sum.

The Bitcoin ledger. The ledger functionality $\mathcal{G}_{\text{LEDGER}}^{\text{B}}$ maintains a ledger state $\texttt{state}$, which is a sequence of state blocks. A state block contains (application-specific) content values—the "transactions." For each honest party p_i, the ledger stores a pointer to a state block—the head of the state from p_i's point of view—and ensures that pointers increase monotonically and are not too far away from the head of the state (and that it only moves forward). Parties or the adversary might submit transactions, which are first validated by means of a predicate $\mathsf{ValidTx}_{\text{B}}$, and, if considered valid, are added to the functionality's buffer. At any time, the $\mathcal{G}_{\text{LEDGER}}^{\text{B}}$ allows the adversary to propose a candidate next-block for the state. However, the ledger enforces a specific *extend policy* specified by an algorithm $\mathsf{ExtendPolicy}$ that checks whether the proposal is compliant with the policy. If the adversary's proposal does not comply with the ledger policy, $\mathsf{ExtendPolicy}$ rejects the proposal. The policy enforced by the Bitcoin ledger can be succinctly summarized as follows:

- *Ledger's growth.* Within a certain number of rounds the number of added blocks must not be too small or too large.
- *Chain quality.* A certain fraction of the proposed blocks must be mined honestly and those blocks satisfy special properties (such as including all recent transactions)..
- *Transaction liveness.* Old enough (and valid) transactions are included in the next block added to the ledger state.

We refer to Appendix A.2 and Appendix A.3 for more details on this functionality.

The Bitcoin blockchain. In [1] it was proved that (a [12]-inspired abstraction of) Bitcoin as a synchronous-UC protocol [19], called the *ledger protocol* and denoted by Π^{B}, realizes the above ledger. Π^{B} uses blockchains to store a sequence of transactions. A *blockchain* $\mathcal{C}$ is a (finite) sequence of blocks $\mathbf{B}_1, \ldots, \mathbf{B}_\ell$. Each *block* $\mathbf{B}_i$ consist of a *pointer* $\texttt{s}_i$, a *state block* $\texttt{st}_i$, and a *nonce* $\texttt{n}_i$. string. The chain $\mathcal{C}^{\lceil k}$ is $\mathcal{C}$ with the last k blocks removed. The *state* $\vec{\texttt{st}}$ of the blockchain $\mathcal{C} = \mathbf{B}_1, \ldots, \mathbf{B}_\ell$ is defined as a sequence of its state blocks, i.e., $\vec{\texttt{st}} := \texttt{st}_1 || \ldots || \texttt{st}_\ell$.

The validity of a blockchain $\mathcal{C} = \mathbf{B}_1, \ldots, \mathbf{B}_\ell$ where $\mathbf{B}_i = \langle \texttt{s}_i, \texttt{st}_i, \texttt{n}_i \rangle$ depends on two aspects: *chain-level* validity, also referred to as syntactic validity, and a *state-level* validity also referred to as semantic validity. Syntactic validity is defined with respect to a difficulty parameter $\texttt{d} \in [2^\kappa]$, where κ is the security parameter, and a given hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$; it requires that, for each $i > 1$, the pointer $\texttt{s}_i$ is the hash of the previous block, i.e., $\texttt{s}_i = \mathsf{H}[\mathbf{B}_{i-1}]$ and that additionally $\mathsf{H}[\mathbf{B}_i] < \texttt{d}$ holds. Finding a nonce such that this inequality holds is typically called a "proof of work."

The semantic validity is defined on the state $\vec{\texttt{st}}$ encoded in the blockchain $\mathcal{C}$ and specifies whether this content is valid with respect to $\mathsf{ValidTx}_{\text{B}}$. This is implemented as the algorithm $\mathsf{isvalidstate}$ which depends on the predicate $\mathsf{ValidTx}_{\text{B}}$. $\mathsf{isvalidstate}$ checks that the blockchain state can be built in an iterative manner, such that each contained transaction is considered valid according to $\mathsf{ValidTx}_{\text{B}}$ upon insertion. It further ensures that the state starts with the genesis state and that state blocks contain a special *coin-base* transaction $\texttt{tx}_{\text{minerID}}^{\text{coin-base}}$ which assigns them to a miner. For more details refer to [1].

We denote by $\mathsf{isvalidchain}_D(\mathcal{C})$ the predicate that returns 1 iff chain $\mathcal{C}$ satisfies syntactic and semantic validity as defined above.

The Bitcoin protocol in UC. The Bitcoin protocol Π^{B} is executed in a hybrid world where parties have access to a random oracle functionality $\mathcal{F}_{\text{RO}}$ (that models the hash function H), a network $\mathcal{F}_{\text{N-MC}}$ and clock $\mathcal{G}_{\text{CLOCK}}$. Each party holds a local blockchain which initially contains just the genesis block. During the execution of the protocol the chains of honest parties might differ. However, it is ensured that they have "common prefix" [12] defining the ledger state as long as the majority (of the computing power) is honest. New transactions are added through a "mining process." First, each party collects valid transactions (with respect to $\mathsf{ValidTx}_{\text{B}}$) and creates a new state block. Next, the party makes a certain number of attempts to mine a new block which can be validly added to their local blockchain. This mining is done using the algorithm $\mathsf{extendchain}_D$ in [1]). The main idea of the algorithm is to find a proof of work by querying the random oracle $\mathcal{F}_{\text{RO}}$ which allows to extend the local chain by a syntactically and semantically correct block. After each mining attempt the party uses the network to multicast their current blockchain. If a party receives a longer chain, it uses it to replace its local chain. The protocol defines the ledger state to be a certain prefix of the contents of the longest chain held by each party. More specifically, if a party holds a valid chain $\mathcal{C}$ that encodes the sequence of state blocks $\vec{\mathtt{st}}$, then the ledger state is defined to be $\vec{\mathtt{st}}^{\lceil T}$, i.e., the party outputs a prefix of the encoded state blocks of its local longest chain. T is chosen such that honest parties output a consistent ledger state. The overall Bitcoin protocol is denoted by Π^{B}.

The flat model of computation. In this paper, we state the results in the synchronous flat model (with fixed difficulty) by Garay *et al.* [12]. This means we assume a number of parties, denoted by n, that execute the Bitcoin protocol Π^{B}, out of which t parties can get corrupted. For simplicity, the network $\mathcal{F}_{\text{N-MC}}$ guarantees delivery of messages sent by honest parties in round r to be available to any other party at the onset of round $r+1$. Moreover, every party will be invoked in every round and can make at most one "calculation" query to the random oracle $\mathcal{F}_{\text{RO}}$ in every round (and an unrestricted number of "verification" queries to check the validity of received chains)[15], and use the above diffusion network $\mathcal{F}_{\text{N-MC}}$ once in a round to send and receive messages. To capture these restrictions in a composable treatment, the real-world assumptions are enforced by means of a "wrapper" functionality, $\mathcal{W}_{\text{flat}}$, which adequately restricts access to $\mathcal{G}_{\text{CLOCK}}, \mathcal{F}_{\text{RO}}$ and $\mathcal{F}_{\text{N-MC}}$ as explained in [1].

Denote by ρ the fraction of dishonest parties (i.e., $t = \rho \cdot n$) and define $p := \frac{\mathtt{d}}{2^{\kappa}}$ which is the probability of finding a valid proof of work via a fresh query to $\mathcal{F}_{\text{RO}}$ (where $\mathtt{d}$ is fixed but sufficiently small, depending on n). Let $\alpha^{\text{flat}} = 1-(1-p)^{(1-\rho)\cdot n}$ be the mining power of the honest parties, and $\beta^{\text{flat}} = p \cdot (\rho \cdot n)$ be the mining power of the adversary.

Theorem 1. *Consider Π^{B} in the $\mathcal{W}_{\text{flat}}(\mathcal{G}_{\text{CLOCK}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{N-MC}})$-hybrid world. If, for some $\lambda > 1$, the honest-majority assumption*

$$\alpha^{\text{flat}} \cdot (1 - 4\alpha^{\text{flat}}) \geq \lambda \cdot \beta^{\text{flat}}$$

holds in any real-world execution, then protocol Π^{B} UC-realizes $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$ for some specific range of parameters (given in [1]*).*

3 Rational Protocol Design of Ledgers

In this section we present our framework for rational analysis of the Bitcoin protocol. It uses as basis the framework for *rational protocol design* (and analysis—RPD framework for short) by Garay *et al.* [10], extending it in various ways to better capture Bitcoin's features. (We refer to Section 2 and to the Appendix for RPD's main components and security definitions.) We note that although our analysis mainly focuses on Bitcoin, several of the extensions have broader applicability, and can be used for the rational analysis of other cryptocurrencies as well.

RPD's machinery offers the foundations for capturing incentive-driven attacks against multi-party protocols for a given specification. In this section we show how to tailor this methodology to the specific task of protocols aimed to securely implement a public ledger. The extensions and generalizations of the original RPD framework we provide add generic features to the RPD framework, including the ability to capture non-zero-sum attack games—which, as we argue, are more suitable for the implementation of a cryptocurrency ledger—and the extension of the class of events which yield payoff to the attacker and the designer.

The core hypothesis of our rational analysis is that the incentives of an attacker against Bitcoin—which affect his actions and attacks—depend only on the possible earnings or losses of the parties that launch the attack. We do not consider, for example, attackers that might create forks just for the fun of it. An attacker might create a "fork" in the blockchain if he expects to gain something by doing so. In more detail, we consider the following events that yield payoff (or inflict a cost) for running the Bitcoin protocol:

- *Inserting a block into the blockchain.* It is typical of cryptocurrencies that when a party manages to insert a block into the ledger's state, then it is rewarded for the effort it invested in doing so. In addition, it is typical in such protocols that the contents of the blocks (usually transactions) have some *transaction*

[15] This fine-grained round model with one hash query was already used by Pass et al. [26]. The extension to a larger, constant upper bound of calculation queries per round as in [12] is straightforward for the results in this work.

fee associated with them. (For simplicity, in our initial formalization [Sections 3 and 4] we will ignore transaction fees in our formal statements, describing how they are extended to also incorporate also such fees in Section 5.)

- *Spending resources to mine a block.* These resources might be the electricity consumed for performing the ining, the investment on mining hardware and its deterioration with time, etc.

Remark 2 (The miners/users separation principle). We remark that the scope of our work is to analyze the security of cryptocurrencies against incentive-driven attacks by the miners, i.e., the parties that are responsible for maintaining the blockchain. In particular, consistently with [12, 26, 1] we shall consider the inputs to the protocol as provided by a (not-necessarily rational) environment, which in particular captures the users of the system. As a result, other than the transaction fees, we will assume that the contents of the ledger do not affect the miners' strategies, which we will refer to as the *miners/users separation principle.* This principle captures the case where the users do not collude with the miners—an assumption implicit in the above works. We leave the full rational analysis of the protocol, including application layers for future research.

There are several challenges that one needs to overcome in deriving a formal treatment of incentive-driven attacks against Bitcoin. First, the above reward and cost mechanisms are measured in different "units." Specifically, the block reward is a cryptocurrency convention and would therefore be measured in the specific cryptocurrency's units, e.g., BTCs in the the case of the Bitcoin network. On the other hand, the cost for mining (e.g., the cost of electricity, equipment usage, etc.) would be typically measured in an actual currency. To resolve this mismatch—and refrain from adopting a specific currency—we introduce a variable $\mathtt{CR}$ which corresponds to the *conversion rate* of the specific cryptocurrency unit (e.g., BTCs) to the cost unit (e.g., euros or US dollars). As we shall see in the next section, using such an explicit exchange rate allows us to make statements about the quality of the Bitcoin network that depend on its price—as they intuitively should. For example, we can formally confirm high-level statements of the type: "Bitcoin is stable—i.e., miners have incentive to keep mining honestly—as long as its price is high enough" (cf. Section 4).

Furthermore, this way we can express all payoffs in terms of cost units: Assume that it takes r rounds for a miner (or a collection of miners) to insert a block into the state. Denote by $\mathtt{mcost}$ the cost for a single mining attempt (in our case a single RO query), and by $\mathtt{breward}$ the fraction of cryptocurrecy units (e.g., BTCs) that is given as a reward for each mined block.[16] Then, the payoff for the insertion of a single block is $\mathtt{breward} \cdot \mathtt{CR} - q_r \cdot \mathtt{mcost}$, where q_r is the number of queries to the RO that were required to mine this block during r rounds.

The second challenge is with respect to *when* should a miner receive the reward for mining. There are several reasons why solving a mining puzzle—thereby creating a new block—does not necessary guarantee a miner that he will manage to insert this block into the blockchain, and therefore be rewarded for it, including the possibility of collisions—more than one miner solving the puzzle—or, even worse, adversarial interference—e.g., network delays or "selfish mining." And even if the miner is the only one to solve the puzzle in a given round, he should only be rewarded for it if his block becomes part of the (permanent) state of the blockchain—the so-called blockchain's "common prefix."

To overcome this second challenge we rely on the RPD methodology. In particular, we will use the ideal experiment where parties have access to the global ledger functionality, where we can clearly identify the event of inserting a block into the state, and decide, by looking into the state, which miner added which block.[17]

In order to formalize the above intuitions and apply the RPD methodology to define the utilities in the attack game corresponding to implementing a ledger against an incentive-driven adversary, we need to make some significant adaptations and extensions to the original framework, which is what we do next. We then (Section 3.2) use the extended framework to define the attack-model for the Bitcoin protocol, and conclude the section by giving appropriate definitions of security and stability in this model.

[16] Currently, for the Bitcoin network, this is 1/4 of the original reward (12.5 BTCs).

[17] In [1], each block of the state includes the identifier of the miner who this block is attributed to.

3.1 Extending the RPD Framework

We describe how to extend the model from [10] to be able to use it in our context.

Black-box simulators. The first modification is adding more flexibility to how utilities are defined. The original definition of ideal payoff $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})$ computes the payoff of the simulator using the joint view of the environment and the functionality. This might become problematic when attempting to assign cost to resources used by the adversary—the RO queries in our scenario, for example. Indeed, these queries are not necessarily in this joint view, as depending on the simulator, one might not be able to extract them.[18] To resolve this we modify the definition to restrict it to black-box simulators, resulting in $\mathcal{C}_{\mathcal{A}}$ being the class of simulators that use the adversary as a black box. This will ensure that the queries to the RO are part of the interaction of the simulator with its adversary, and therefore present in the view of the simulator. Further, we include this part of the simulator's view in the definition of the scoring function $v_{\mathtt{A}}$, which is defined now as a mapping from the joint view of the relaxed functionality $\langle\mathcal{F}\rangle$, the environment $\mathcal{Z}$, and the simulator $\mathcal{S}$ to a real-valued *payoff*.

Non-zero-sum attack games. The second modification is removing the assumption that the attack game is zero-sum. Indeed, the natural incentive of the protocol designer in designing a Ledger protocol is not to optimally "tame" its attacker—as in [10]—but rather to maximize the revenue of the non-adversarially controlled parties while keeping the blockchain healthy, i.e., free of forks. This is an important modification as it captures attacks in which the adversary preserves his rate of blocks inserted into the state, but slows down the growth of the state to make sure that honest miners accrue less revenue in any time interval. For example, the so called "selfish-mining" strategy [8] provokes a slowdown since honest mining power is invested into mining on a chain which is not the longest one (as the longest chain is kept private as long as possible by the party that does the selfish-mining).

To formally specify the utility of the designer in such a non-zero-sum attack game, we employ a similar reasoning as used in the original RPD framework for defining the attacker's utility. The first step, relaxing the functionality, can be omitted provided that we relaxed it sufficiently in the definition of the attacker's utility. In the second step, we define the scoring mechanism for the incentives of the designer as a function $v_{\mathtt{D}}$ mapping the joint view of the relaxed functionality $\langle\mathcal{F}\rangle$, the environment $\mathcal{Z}$, and the simulator $\mathcal{S}$ to a real-valued *payoff*, and define the designer's *(ideal) expected payoff* for simulator $\mathcal{S}$ with respect to the environment $\mathcal{Z}$ as

$$U_{I^{\mathtt{D}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z}) = E(v_{\mathtt{D}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}),$$

where $v_{\mathtt{D}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ describes (as a random variable) the payoff of $\mathtt{D}$ allocated by $\mathcal{S}$ in an execution using directly the functionality $\langle\mathcal{F}\rangle$.

The third and final step is the trickiest. Here we want to use the above ideal expected payoff to define the expected payoff of a designer using protocol Π when the attacker is playing adversary $\mathcal{A}$. In order to ensure that our definition is consistent with the original definition in [10]—which applied to (only) zero-sum games—we need to make sure that the utility of the designer increases as the utility of the attacker decreases and vice versa. Thus, to assign utility for the designer to a strategy profile $(\Pi,\mathcal{A})$, we will use the same simulators and environments that were used to assign the utility for the attacker. Specifically, let $\mathbb{S}_{\mathtt{A}}$ denote the class of simulators that are used to formulate the utility of the adversary, and let $\mathbb{Z}_{\mathtt{A}}$ denote the class of environments that maximize this utility for simulators in $\mathbb{S}_{\mathtt{A}}$[19], then

$$\mathbb{S}_{\mathtt{A}} = \left\{\mathcal{S}\in\mathcal{C}_{\mathcal{A}} \text{ s.t. } \sup_{\mathcal{Z}\in\mathtt{ITM}}\{U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})\} = u_{\mathtt{A}}(\Pi,\mathcal{A})\right\} \tag{1}$$

and

$$\mathbb{Z}_{\mathtt{A}} = \left\{\mathcal{Z}\in\mathtt{ITM} \text{ s.t. for some } \mathcal{S}\in\mathbb{S}_{\mathtt{A}}: U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})\} = u_{\mathtt{A}}(\Pi,\mathcal{A})\right\}. \tag{2}$$

It is easy to verify that this choice of simulator respects the utilities being opposite in a zero-sum game as defined in [10], thereby preserving the results following the original RPD paradigm.

[18] Indeed, in the ideal simulation of the Bitcoin protocol presented in [1], there is no RO in the ideal world.

[19] Recall that as argued in Section 2.1, these sets are non-empty provided $\mathcal{C}_{\mathcal{A}} \neq \emptyset$.

Lemma 1. *Let* $v_{\mathtt{D}} = -v_{\mathtt{A}}$ *and let* $U_{I^{\mathtt{D}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z})$ *defined as above. For some* $\mathcal{S} \in \mathbb{S}_{\mathtt{A}}$ *and some* $\mathcal{Z} \in \mathbb{Z}_{\mathtt{A}}$*, define* $u_{\mathtt{D}}(\Pi, \mathcal{A}) := U_{I^{\mathtt{D}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z})$*. Then* $u_{\mathtt{D}}(\Pi, \mathcal{A}) = -u_{\mathtt{A}}(\Pi, \mathcal{A})$*.*

Proof. Since $v_{\mathtt{D}} = -v_{\mathtt{A}}$, we have that for all $\mathcal{Z}, \mathcal{S} \in \mathtt{ITM}$,

$$U_{I^{\mathtt{D}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z}) = -U_{I^{\mathtt{A}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z}). \tag{3}$$

However, by definition, since $\mathcal{S} \in \mathbb{S}_{\mathtt{A}}$, we have

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) = U_{I^{\mathtt{A}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z}) \overset{3}{=} -U_{I^{\mathtt{D}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z}) = -u_{\mathtt{D}}(\Pi, \mathcal{A}).$$

□

The above lemma confirms that for a zero-sum attack game we can take any pair $(\mathcal{S}, \mathcal{Z}) \in \mathbb{S}_{\mathtt{A}} \times \mathbb{Z}_{\mathtt{D}}$ in the definition of $u_{\mathtt{D}}(\Pi, \mathcal{A})$ and it will preserve the zero-sum property (and hence all the original RPD results). This is so because all these simulators induce the same utility $-u_{\mathtt{A}}(\Pi, \mathcal{A})$ for the designer. However, for our case of non-zero-sum games, each of those simulator/environment combinations might induce a different utility for the designer. To choose the one which most faithfully translates the designer's utility from the real to the ideal world we use the same line of argument as used in RPD for defining the attacker's utility: The best (i.e., the most faithful) simulator is the one which always rewards the designer whenever his protocol provokes some profitable event; in other words, the one that maximizes the designer's expected utility. Similarly, the natural environment is the one that puts the protocol in its worst possible situation, i.e., the one that minimizes its expected gain; indeed, such an environment will ensure that the designer is guaranteed to get his allocated utility. The above leads to the following definition for the designer's utility in non-zero-sum games:

$$u_{\mathtt{D}}(\Pi, \mathcal{A}) := \inf_{\mathcal{Z} \in \mathbb{Z}_{\mathtt{A}}} \left\{ \sup_{\mathcal{S} \in \mathbb{S}_{\mathtt{A}}} \left\{ U_{I^{\mathtt{D}}}^{\langle \mathcal{F} \rangle}(\mathcal{S}, \mathcal{Z}) \right\} \right\}.$$

For completeness, we set $u_{\mathtt{D}}(\Pi, \mathcal{A}) = -\infty$ if $\mathcal{C}_{\mathcal{A}} = \emptyset$, i.e., if the protocol does not even achieve the relaxed functionality. This is not only intutive—as $\mathcal{C}_{\mathcal{A}} = \emptyset$ means that the designer chose a protocol which does not even reach the relaxed goal—but also analogous to how RPD defines the attacker's utility for protocols that do not achieve their relaxed specification.[20]

Finally, the attack model for non-zero-sum games is defined as the quadruple $\mathcal{M} = (\mathcal{F}, \langle \mathcal{F} \rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$.

3.2 Bitcoin in the RPD Framework

Having formulated the above extensions to the RPD framework, we are ready to apply the methodology to analyze Bitcoin.

Basic foundations. We explain in more depth on how to implement the core steps of RPD. First, we define the Ledger functionality from [1] as Bitcoin's ideal goal (see Section 2.2). Following the three steps of the methodology, we start by defining the relaxed version of the Ledger, denoted as $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$. Informally, the relaxed Ledger functionality operates as the original ledger with the following modifications:

The state is a tree: Instead of storing a single ledger state `state` as a straight-line blockchain-like structure, $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$ stores a tree `state-tree` of state blocks where for each node the direct path from the root defines a possible ledger state that might be presented to any of the honest miners. The functionality maintains for each registered party $p_i \in \mathcal{P}$ a pointer $\mathtt{pt}_i$ to a node in the tree which defines p_i's current-state view. Furthermore, instead of restricting the adversary to only be able to set the state "slackness" to be not larger than a specific parameter, $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$ offers the command SET-POINTER which allows the adversary to set the pointers of honest parties within `state-tree` with the following restriction: The pointer of an honest party can only be set to a node whose distance to the root is at least the current-pointer node's.

[20] Recall that RPD sets $u_{\mathtt{A}}(\Pi, \mathcal{A}) = \infty$ if $\mathcal{A}$ cannot be simulated, i.e., if $\mathcal{C}_{\mathcal{A}} = \emptyset$.

Relaxed validity check of transactions: All submitted transactions are accepted into the buffer `buffer` without validating against `state-tree`. Moreover, transactions in `buffer` which are added to `state-tree` are not removed as they could be reused at another branch of `state-tree`.

Ability to create forks: This relaxation gives the simulator the explicit power to create a fork on the ledger's state. This is done as follows: The command NEXT-BLOCK—which, recall, allows the simulator to propose the next block—is modified to allow the simulator to extend an arbitrary leaf of a sufficiently long rooted path of `state-tree`. Thus, when `state-tree` is just a single path, this command operates as in the original ledger from [1]. Additionally, in the relaxed ledger, the simulator is also allowed to add the next block to an intermediate, i.e., non-leaf node of `state-tree`. This is done by using an extra command FORK which, other than extending the chain from the indicated block provides the same functionality as NEXT-BLOCK.

Relaxed state-extension policy: As explained in Section 2.2, the extend policy is a compliance check that the ledger functionality performs on blocks that the simulator proposes to be added to the ledger's state. This is to ensure that they satisfy certain conditions. This is the mechanism which the ledger functionality uses to enforce, among others, common generic-ledger properties from the literature, such as the chain quality or the chain growth properties, and for Bitcoin ledgers the transaction-persistence/stability properties [12, 26]. of the ledger state, or on transaction persistence/stability [12].

The relaxed ledger uses a much more permissive extend policy, denoted as weakExtendPolicy, derived from ExtendPolicy with the following modifications: Intuitively, in contrast to ExtendPolicy, the weaker version does not check if the adversary inserts too many or too few blocks, and it does not check if all old-enough transactions have been included. There is also no check of whether enough blocks are mined by honest parties, i.e., that there are enough blocks with coin-base transactions from honest parties. In other words, weakExtendPolicy does not enforce any concrete bounds on the chain quality or the chain growth properties of the ledger state, or on transaction persistence/stability. It rather ensures basic validity criteria of the resulting ledger state.

More formally, instead of `state`, it takes `state-tree` and a pointer `pt` as input. It first computes a valid default block $\vec{N}_{\texttt{df}}$ which can be appended at the longest branch of `state-tree`. It then checks if the proposed blocks $\vec{N}$ can be safely appended to the node `pt` (to yield a valid state). If this is the case it returns $(\vec{N}, \texttt{pt})$; otherwise it returns $\vec{N}_{\texttt{df}}$ and a pointer to the leaf of the longest branch in `state-tree`.

For completeness we have included the formal description of the relaxed ledger functionality as Supplement B. This completes the first step of the RPD methodology.

The second step is defining the scoring function. This is where our application of RPD considerably deviates from past works [10, 11]. In particular, those works consider attacks against generic secure multi-party computation protocols, where the ideal goal is the standard secure function evaluation (SFE) functionality (cf. [5]). The security breaches are breaking correctness and privacy [10] or breaking fairness [11]. These can be captured by relaxing the SFE functionality to allow the simulator to request extra information (breaking privacy), reset the outputs of honest parties to a wrong value (breaking correctness), or cause an abort (breaking fairness.) The payoff function is then defined by looking at events corresponding to whether or not the simulator provokes these events, and the adversary is given payoff whenever the best simulator is forced to provoke them in order to simulate the attack.

However, attacks against the ledger that have as an incentive increasing the revenue of a coalition are not necessarily specific events corresponding to the simulator sending special "break" commands. Rather, they are events that are extracted from the joint views (e.g., which blocks make it to the state and when). Hence, attacks to the ledger correspond to the simulator implicitly "tweeking" its parameters. Therefore, in this work we take the following approach to define the payoffs of the attacker and designer. In contrast to the RPD examples in [10, 11], which use explicit events that "downgrade" the ideal functionality for defining utility, we directly use more intuitive events defined on the joint view of the environment, the functionality, and the simulator. The reason is that as we have assumed that the only rationale is to increase one's profit, the incentives in case of cryptocurrencies are as follows: whenever a block is mined, the adversary gets rewarded. A "security breach" is relevant if (and only if) the adversary can get a better reward by doing so.

Defining concrete utility functions. Defining the utility functions lies at the core of a rational analysis of a blockchain protocol like Bitcoin. The number of aspects that one would like to consider steers the complexity of a concrete analysis, the ultimate goal being to reflect exactly the incentive structure of the actual blockchain ecosystem. Our extended RPD framework for blockchain protocols provides a guideline to defining utility functions of various complexity and to conduct the associated formal analysis. Recall that the utility functions are the means to formalize the assumed underlying incentive structure. As such, our approach is extensible: if certain relevant properties or dynamics are identified or believed (such as reflecting a doomsday risk of an attacker or a altruistic motivation of honest miners), one can enrich the incentive structure by reflecting the associated events and rewards in the utility definition, or by making the costs and rewards time-dependent variables. The general goal of this line of research on rational aspects of cryptocurrencies is to eventually arrive at a more detailed model and, if the assumptions are reasonable, to have more predictive models for reality.

Below we define a first, relatively simple incentive model to concretely showcase our methodology. We conduct the associated rational analysis in the next section and observe that, although being a simplified model, we can already draw interesting conclusions from such a treatment.

Utility of the attacker. Informally, this particular utility is meant to capture the average revenue of the attacker. Consider the following sequence of events defined on the views of the environment, the relaxed ledger functionality, and the black-box simulator of the entire experiment (i.e., until the environment halts) for a given adversary $\mathcal{A}$:

1. For each pair $(q, r)_{\in} \mathbb{N}^2$ define event $W^{\mathtt{A}}_{q,r}$ as follows: The simulator makes q mining queries in round r, i.e., it receives q responses on different messages to the RO in round r.[21]

2. For each pair $(b, r) \in \mathbb{N}^2$ define event $I^{\mathtt{A}}_{b,r}$ as follows: The simulator inserts b blocks into the state of the ledger in round r, such that all these blocks were previously queries to the (simulated) random oracle by the adversary. More formally, $I^{\mathtt{A}}_{b,r}$ occurs if the function extend policy (of the weak ledger) is successfully invoked and outputs a sequence of b non-empty blocks (to be added to the state), where for each of these blocks the following properties hold: (1) The block has appeared in the past in the transcript between the adversary and the simulator, and (2) the contents of the block have appeared on this transcript prior to the block's first appearance, as a query from the the adversary to its (simulated) RO. We note in passing that this event definition ensures that the simulator (and therefore also the adversary) does not earn reward by adaptively corrupting parties after they have done the work/query to mine a block but before their block is added into the state. In other words, the adversary only gets rewarded for state blocks which corrupted parties mined while they where already under the adversary's control.

Now, using the simplified event-based utility definition (Remark 1) we define the attacker's utility for a strategy profile $(\mathit{\Pi}, \mathcal{A})$ in the attack game as:[22]

$$u^{\text{₿}}_{\mathtt{A}}(\mathit{\Pi}, \mathcal{A}) = \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \mathtt{breward} \cdot \mathtt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}] \right.\right.$$
$$\left.\left. - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}] \right\} \right\}.$$

We remark that although the above sums are in principle infinite, in any specific execution these sums will have only as many (non-zero) terms as the number of rounds in the protocol. Indeed, if the experiment

[21] Observe that since our ideal world is the $\mathcal{G}_{\text{CLOCK}}$-hybrid synchronous world, the round structure is trivially extracted from the simulated ideal experiment by the protocol definition and the clock value. Furthermore, the adversary's mining queries can be trivially extracted by its interaction with the black-box simulator.

[22] Recall that we assume synchronous execution as in [1] where the environment gets to decide how many rounds it wishes to witness.

finishes in r' rounds then for any $r > r'$, $\Pr[I^{\mathcal{A}}_{b,r}] = \Pr[W^{\mathcal{A}}_{q,r}] = 0$ for all $b \in \mathbb{N}$. Furthermore, we assume that $\texttt{breward}, \texttt{CR}$ and $\texttt{mcost}$ are $O(1)$, i.e., independent of the security parameter.

The above expression can be simplified to the following more useful expression. Let $B^{\mathtt{A}}$ denote the random variables corresponding to the number of blocks contributed to the ledger's state by adversarial miners and $Q^{\mathtt{A}}$ denote the number of queries to the RO performed by adversarial miners (throughout the execution of the random experiment). Then the adversary's utility can be described using the expectations of these random variables as follows:

$$u^{\text{B}}_{\mathtt{A}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z}\in\mathtt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}}\in\mathcal{C}_{\mathcal{A}}} \left\{ \texttt{breward} \cdot \texttt{CR} \cdot E(B^{\mathtt{A}}) - \texttt{mcost} \cdot E(Q^{\mathtt{A}}) \right\} \right\}.$$

Utility of the designer. Since the game is not zero-sum we also need to formally specify the utility of the protocol designer. Recall that we have assumed that, analogously to the attacker, the designer accrues utility when honest miners insert a block into the state, and spends utility when mining—i.e., querying the RO. In addition, what differentiates the incentives of the designer from that of an attacker is that his most important goal is to ensure the "health" of the blockchain, i.e., to avoid forks. To capture this, we will assign a cost for the designer to the event the simulator is forced to request the relaxed ledger functionality to fork, which is larger than his largest possible gain. This yields the following events that are relevant for the designer's utility.

1. For each pair $(q, r) \in \mathbb{N}^2$ define $W^{\Pi}_{q,r}$ as follows: The honest parties, as a set, make q mining queries in round r. [23]
2. For each pair $(b, r) \in \mathbb{N}^2$ define $I^{\Pi}_{b,r}$ as follows: The honest parties jointly insert b blocks into the state of the ledger in round r; that is, the simulator inserts b blocks into the state of the ledger in round r, such that for each of these blocks, at least one of the two properties specified in the the above definition of $I^{\mathtt{A}}_{b,r}$ does not hold.[24]
3. For each $r \in \mathbb{N}$ define K_r as follows: The simulator uses the FORK command in round r.

The utility of the designer is then defined similarly to the attacker's, where we denote by $\mathbb{S}_{\mathtt{A}}$ the class of simulators that assign to the adversary his actual utility (cf. Equation 1):

$$u^{\text{B}}_{\mathtt{D}}(\Pi, \mathcal{A}) = \inf_{\mathcal{Z}\in\mathbb{Z}} \left\{ \sup_{\mathcal{S}^{\mathcal{A}}\in\mathbb{S}_{\mathcal{A}}} \left\{ \sum_{(b,r)\in\mathbb{N}^2} b \cdot \texttt{CR} \cdot (\texttt{breward} \cdot \Pr[I^{\Pi}_{b,r}] - 2^{\mathsf{polylog}(\kappa)} \cdot \Pr[K_r]) \right.\right.$$
$$\left.\left. - \sum_{(q,r)\in\mathbb{N}^2} q \cdot \texttt{mcost} \cdot \Pr[W^{\Pi}_{q,r}] \right\} \right\}.$$

At first glance, the choice of $2^{\mathsf{polylog}(\kappa)}$ might seem somewhat arbitrary. However, it is there to guarantee that if the ledger state forks (recall that this reflects a violation of the common-prefix property) with noticeable probability, then the designer is punished with this super-polynomially high penalty to make his expected payoff negative as κ grows. On the other hand, if the probability of such a fork is sufficiently small (e.g. in the order of $2^{-\Omega(\kappa)}$), then the loss in utility is made negligible. This, combined with the fact that our stability notions will render negligible losses in the utility irrelevant, will allow the designer the freedom to provide slightly imperfect protocols, i.e., protocols where violations of the common-prefix property occur with sufficiently small probability.

We will denote by $\mathcal{M}^{\text{B}}$ the Bitcoin attack model which has $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$ as the goal, $\langle \mathcal{G}^{\text{B}}_{\text{LEDGER}} \rangle$ as the relaxed functionality, and scoring functions for the attacker and designer inducing utilities $u^{\text{B}}_{\mathtt{A}}$ and $u^{\text{B}}_{\mathtt{D}}$, respectively.

[23] Note that although there is no RO in the ideal model of [1], whenever a miner would make such a query in the Bitcoin protocol, the corresponding dummy party sends a special MAINTAIN-LEDGER command to the Ledger functionality, making it possible for us to count the mining queries also in the ideal world.

[24] By definition, these two properties combined specify when the adversary should be considered the recipient of the reward.

3.3 Attack-Payoff Security and Incentive Compatibility

The definition of the respective utilities for designer and attacker completes the specification of an attack game. Next, we define the appropriate notions of security and stability as they relate to Bitcoin and discuss their meaning.

We start with *attack-payoff security* [10], which, as already mentioned, captures that the adversary would have no incentive to make the protocol deviate from a protocol that implements the ideal specification (i.e., from a protocol that implements the ideal [non-relaxed] ledger functionality), and which is useful in arguing about the resistance of the protocol against incentive-driven attacks. However, in the context of Bitcoin analysis, one might be interested in achieving an even stronger notion of incentive-driven security, which instead of restricting the adversary to strategies that yield payoff as much as the ideal ledger $\mathcal{G}_{\text{LEDGER}}^{\text{B}}$ from [1] would, restricts him to play in a coordinated fashion but *passively*, i.e., follow the mining procedure mandated by the Bitcoin protocol, including announcing each block as soon as it is found, but ensure that no two corrupt parties try to solve the same puzzle (i.e., use the same nonce).

One can think of the above strategy as corresponding to cooperating mining-pools which run the standard Bitcoin protocol. Nonetheless, as the adversary has control over message delays, he is able to make sure that whenever he finds a new block in the same round as some other party, his own block will be the one propagated first[25], and therefore the one that will be added to the blockchain. Note that a similar guarantee is not there for honest miners as in the event of collisions—two miners solve a puzzle in the same round—the colliding miners have no guarantee about whose block will make it. We will refer to such an adversary that sticks to the Bitcoin mining procedure but makes sure his blocks are propagated first as *front running*.

Definition 2 (Front-running, passive mining adversary). *The front-running adversarial strategy $\mathcal{A}_{\texttt{fr}}$ is specified as follows: Upon activation in round $r > 0$, $\mathcal{A}_{\texttt{fr}}$ activates in a round-robin fashion all its (passively) corrupted parties, say $p_1, \ldots, p_t$. When party p_i generated some new message to be sent through the network, $\mathcal{A}_{\texttt{fr}}$ immediately delivers m to all its recepients.*[26] *In addition, upon any activation, any message submitted to the network $\mathcal{F}_{\text{N-MC}}$ by an honest party is maximally delayed.*

Note that there might be several front-running, passive mining strategies, depending on which parties are corrupted and (in case of adaptive adversaries) when. We shall denote the class of all such adversary strategies by $\mathbb{A}_{\texttt{fr}}$. We are now ready to provide the definition of (strong) attack-payoff security for Bitcoin. The definition uses the standard notion of *negl-best-response* strategy from game theory: Consider a two-player game with utilities u_1 and u_2, respectively. A strategy for m_1 of p_1 is *best response* to a strategy m_2 of p_2 if for all possible strategies m_1', $u_1(m_1', m_2) \leq u_1(m_1, m_2) + \text{negl}(\kappa)$. For conciseness, in the sequel we will refer to negl-best-response simply as best-response strategies.

Definition 3. *A protocol Π is* strongly attack-payoff secure *for attack model $\mathcal{M}^{\text{B}}$ if for some $\mathcal{A} \in \mathbb{A}_{\texttt{fr}}$ the attacker playing $\mathcal{A}$ is a (negl-)best-response to the designer playing Π.*

Remark 3. It is instructive to see that for such a weak class of adversaries the usual blockchain properties hold with very nice parameters [27]: first, the common-prefix property is satisfied except with negligible probability (as no intentional forks are provoked by anyone). Second, the fraction of honest blocks (in an interval of say k blocks) is roughly $\frac{\alpha}{\alpha+\beta} \overset{p \lll 1}{\approx} \frac{(1-\rho)np}{(1-\rho)np+\rho np} = (1-\rho)$ and thus, in expectation, the chain quality corresponds to the relative mining power of honest parties. Finally, since the adversary does contribute his mining power to the main chain, the number of rounds it takes for the chain to grow by k blocks is in expectation $\frac{k}{\alpha+\beta} \overset{p \lll 1}{\approx} \frac{k}{np}$.

Security thus means that if the honest parties stick to their protocol then the adversary has no incentive to deviate. However, unlike in [10], where the game is zero-sum, in a non-zero-sum setting it does not imply

[25] This can be thought of as a "rushing" strategy with respect to network delays.

[26] I.e., $\mathcal{A}_{\texttt{fr}}$ sets the delay of the corresponding transmissions to 0.

[27] Recall the notation introduced in Section 2.2: n denotes the number of parties, ρ the fraction of corrupted parties, α and β denote honest and dishonest mining power, respectively, and p is the probability of a fresh RO-query to return a correct PoW solution.

that the *designer* has an incentive to stick to the protocol. This means that the definition is useful to answer the question whether, assuming the network keeps mining, some of the miners have an incentive to deviate from the protocol, but it does not address the question of why the *honest miners* would keep mining. To address this question, we adopt the notion of *incentive compatibility* (IC).

Informally, a protocol being incentive-compatible means that both the attacker and the designer are willing to stick to it. In other words, it is strongly attack-payoff secure—i.e., the adversary will run it if the honest parties do—*and* if the adversary plays it passively (and front-running), then the honest miners will have an incentive to follow the protocol—i.e., the protocol is the designer's best response to a passive front-running adversary. We note that requiring IC for Bitcoin for the class of all possible protocols would imply a proof that Bitcoin is not only a protocol that the miners wish to follow, but also that there is no other protocol that they would rather participate in instead. This is clearly too strong a requirement, even more so in the presence of results [12, 27] that argue that there are alternative "fairer" blockchain protocols which improve on the miners' expected revenue. Thus, we can only hope to make such statements for a subclass of possible protocols, and therefore devise a version of IC which is parameterized by the set of all acceptable deviations (i.e., alternative protocols) $\mathbb{\Pi}$. For full generality, we also parameterize it with respect to the class of acceptable adversaries $\mathbb{A}$, but stress that all statements in this work are for the class of all (PPT) adversaries.

Towards providing the formal definition of IC, we first give the straightforward restriction of equilibrium (in our case, subgame-perfect equilibrium) to a subset of strategies.

Definition 4. *Let $\mathbb{\Pi}$ and $\mathbb{A}$ be sets of possible strategies for the designer and the attacker, respectively. We say that a pair $(\Pi, \mathcal{A}) \in (\mathbb{\Pi}, \mathbb{A})$ is* a $(\mathbb{\Pi}, \mathbb{A})$-subgame perfect equilibrium *in the attack game defined by model $\mathcal{M}$, if it is a (negl(κ)-)subgame-perfect equilibrium on the restricted attack game where the set of all possible deviations of the designer (resp., the attacker) is $\mathbb{\Pi}$ (resp., $\mathbb{A}$).*

The formal definition of (parameterized) IC is then as follows:

Definition 5. *Let Π be a protocol and $\mathbb{\Pi}$ be a set of polynomial-time protocols that have access to the same hybrids as Π. We say that Π is* $\mathbb{\Pi}$-incentive compatible *($\mathbb{\Pi}$-IC for short) in the attack model $\mathcal{M}$ iff for some $\mathcal{A} \in \mathbb{A}_{fr}$, $(\Pi, \mathcal{A})$ is a $(\mathbb{\Pi}, \mathtt{ITM})$-subgame-perfect equilibrium in the attack game defined by $\mathcal{M}$.*

4 Analysis of Bitcoin without Transaction Fees

In this section, we present our RPD analysis of Bitcoin for the concrete incentive structure $\mathcal{M}^{\text{B}}$ defined in the previous section. We note that this incentive structure does not, in particular, reflect rewards that stem from transaction fees and hence the reward per block is constant. First, in Section 4.1, we prove that Bitcoin is strongly attack-payoff secure—i.e., if the designer plays it, the attacker is better off sticking to it as well (but in a front-running fashion). The result is independent of the distribution of computing power to honest vs adversarial miners and independent of the conversion rate or the values of `breward` and `mcost`.

Subsequently, in Section 4.2, we investigate the role of mining costs vs conversion rate vs block rewards for the stability (i.e., IC) of Bitcoin in the presence of such incentive-driven coordinated coalitions (e.g., utility-maximizing mining pools.) We devise conditions on these values that either make the utility of honest parties negative—hence make playing the Bitcoin protocol a sub-optimal choice of the protocol designer, or yield high enough utility for mining that makes Bitcoin *optimal* among all possible deviations from the standard protocol that are still compatible with the Bitcoin network (i.e., produce valid blockchains); combining this with the results from Section 4.1, we deduce that for this latter range of parameters Bitcoin is incentive-compatible.

4.1 Attack-Payoff Security of Bitcoin (without Fees)

The attack-payoff security of Bitcoin without fees is stated in the following theorem.

Theorem 2. *The Bitcoin protocol is strongly attack-payoff secure in the attack model $\mathcal{M}^{\text{₿}}$.*

Proof. The theorem follows as a direct corollary of the following general lemma.

Lemma 2. *Given any adversarial strategy, there is a front-running, semi-honest mining adversary $\mathcal{A}$ that achieves better utility. In particular, the adversarial strategy $\mathcal{A}$ makes as many RO-queries per round as allowed by the real-world restrictions, and one environment that maximizes its utility is the environment $\mathcal{Z}$ that activates $\mathcal{A}$ as the first ITM in every round until $\mathcal{A}$ halts.*

Proof intuition. The proof of the lemma consists of three steps. First, we analyze Bitcoin in the real world. By invoking the subroutine-replacement theorem from [10, Theorem 6], we are able to work in a hybrid world where we can easily compute the relevant values, such as the number of blocks an adversary can mine in a given interval of rounds (the hybrid world is the so-called state-exchange hybrid world of [1] which we quickly recall in Appendix C. Second, we show by a generic argument that this real-world analysis is sufficient to compute the payoffs for the attacker (which is defined on the transcript in the ideal world). Last but not least, we make a case distinction whether the adversary has expected utility smaller than zero (in which he does not corrupt any party and does not participate in the network), or whether mining Bitcoin is profitable for the attacker. In both cases, we prove that for any attacker $\mathcal{A}$, we can devise a front-running and semi-honest mining adversary which gets higher utility. The formal proof of the lemma is found in Appendix D. $\square$

4.2 Incentive Compatibility of Bitcoin (Without Fees)

We proceed by investigating how the IC of Bitcoin depends on the relation between rewards and the conversion rate. Concretely, we describe a sufficient condition for IC (Theorem 4) and a condition that makes it non-IC (Theorem 3). We start with the negative result, which, informally, says that if the expected costs are too high with respect to the expected rewards, then Bitcoin is not IC (although it is strongly attack-payoff secure as proved above). As above, we denote by p the probability of solving a proof of work (and hence being a candidate to extend the ledger state) using one query to the random oracle (or equivalently, that a query to the state-exchange functionality successfully extends a state).

Theorem 3. *For $n > 0$ and $\texttt{breward} \cdot \texttt{CR} < \frac{\texttt{mcost}}{p}$ the Bitcoin protocol is not incentive compatible.*

The proof is a straightforward calculation of the utility for the designer per round. Under the above condition, this expectation is less than 0, since they spend (on average) more on queries than what the reward compensates. Hence, the best response would be a protocol that does nothing.

While the above condition implies that the Bitcoin protocol is not a stable solution for all choices of the rewards, costs, and CR, we next provide conditions under which the standard Bitcoin protocol is in fact a stable solution in the attack game. For this, we need to compare it to arbitrary alternative strategies that produce valid blocks for the Bitcoin network. Informally, our condition for IC requires that CR and breward are sufficiently higher than the costs.

Theorem 4. *Consider the real world consisting of the random oracle functionality $\mathcal{F}_{\mathrm{RO}}$, the diffusion network $\mathcal{F}_{\mathrm{N\text{-}MC}}$, and the clock $\mathcal{G}_{\mathrm{CLOCK}}$, and let $\mathcal{W}_{\mathrm{flat}}(\cdot)$ be the wrapper that formalizes the restrictions of the flat model.*[28] *Consider the class $\Pi_{\mathsf{isvalidchain}_{H,d}(\cdot)}$ of protocols Π that are defined for the $\mathcal{W}_{\mathrm{flat}}(\mathcal{G}_{\mathrm{CLOCK}}, \mathcal{F}_{\mathrm{RO}}, \mathcal{F}_{\mathrm{N\text{-}MC}})$-hybrid world and which are compatible with the Bitcoin network, i.e., which obey the following two restrictions:*

1. *With probability 1, the real-world transcript (i.e., the real-world UC-execution of Π, any environment and adversary) does not contain a chain $\mathcal{C}$ with $\mathsf{isvalidchain}_{H,d}(\mathcal{C}) = 0$ and this chain was an output to the network from an uncorrupted protocol instance running Π.*

[28] Recall from [1] that we model restrictions by using functionality wrappers. The above implemented restrictions correspond to the so-called flat model of Bitcoin, where each party gets one query to the random oracle per round and can send and receive one vector of messages in each round.

2. *Upon input* (READ, *sid*) *to a protocol instance, the return value is* (READ, *sid*, $\vec{\mathtt{st}}^{\lceil T}$) *(for some integer* T*), where* $\vec{\mathtt{st}}^{\lceil T}$ *denotes the prefix of the state* $\vec{\mathtt{st}}$ *encoded in the longest valid chain* $\mathcal{C}$ *received by this protocol instance.*

With respect to the class $\Pi_{\mathit{isvalidchain}_{H,d}(\cdot)}$*, the Bitcoin protocol is an incentive-compatible choice for the protocol designer if Bitcoin is profitable as in Lemma 3, i.e., if we are in the region* $\mathtt{breward} \cdot \mathtt{CR} > \frac{n \cdot \mathtt{mcost}}{p}$*, and if*

$$\mathtt{breward} \cdot \mathtt{CR} > \frac{\mathtt{mcost}}{p \cdot (1-p)^{n-1}}. \tag{4}$$

Remark 4. Formula 4 constitutes a stronger requirement than the mere condition that mining should be profitable (which we treat separately in Lemma 3 for completeness). The theorem says that the probability that a fixed miner is uniquely successful stands in a reasonable relation to the mining cost and block rewards to achieve a stable solution. While Bitcoin would already yield positive utility to the protocol designer in the case of $\mathtt{breward} \cdot \mathtt{CR} > \frac{n \cdot \mathtt{mcost}}{p}$, we have for large n, $\frac{\mathtt{mcost}}{p} \cdot n \leq \frac{\mathtt{mcost}}{p} \cdot (\frac{1}{1-p})^{n-1}$ (for $p \in (0,1)$).

Proof intuition. The proof follows by demonstrating, in a sequence of claims, that the actual choices of the Bitcoin protocol (i.e., our abstraction of it) are optimal under the conditions of the theorem. This includes proving that the assumed resources cannot be employed in a way that would yield better payoff to the protocol designer. Intuitively, if the protocol has to be compatible with the Bitcoin network (i.e., it has to produce valid chains with probability 1), and invest its resources to achieve the optimum reward vs. query ratio in a setting where it knows it is running against front-running adversary running Bitcoin (such as mining pools). Optimality under the theorem's condition follows by deducing a couple of useful properties from the fact that the protocol has to work potentially independently (per round) and by computing (and maximizing) the distribution of the possible query-vs.-reward ratios. The formal proof is found in Appendix D. □

We note that the above conditions are not necessarily tight. Thus one might wonder whether we can prove or disprove their tightness, and in the latter case investigate tight conditions for the statements to hold. We conclude this section with the following lemma which serves as first partial attempt to investigate this gap. The lemma implies that there might be potential to prove (partial) IC even for values of the parameters that fall in the gap between the above theorems. We leave the thorough investigation of this gap in terms of stability as a future research direction.

Lemma 3. *If* $\mathtt{breward} \cdot \mathtt{CR} > \frac{n \cdot \mathtt{mcost}}{p}$ *then the Bitcoin protocol yields, with overwhelming probability, a positive utility for the protocol designer in the presence of front-running adversaries, i.e., the Bitcoin protocol is profitable in such a setting.*

5 Analysis of Bitcoin with Transaction Fees

Recall that in our formal treatment a chain $\mathcal{C}$ encodes a ledger state $\vec{\mathtt{st}}$. A ledger state is a sequence of individual state-blocks, i.e., $\vec{\mathtt{st}} = \mathtt{st}_1 || \ldots || \mathtt{st}_\ell$. In addition, each state-block $\mathtt{st} \in \vec{\mathtt{st}}$ (except the genesis state) of the state encoded in the blockchain has the form $\mathtt{st} = \mathsf{Blockify}(\vec{N})$ where $\vec{N}$ is a vector of transactions, i.e., $\vec{N} = \mathtt{tx}_1, \ldots, \mathtt{tx}_k$. A transaction $\mathtt{tx}_i$ can be seen as the abstract content of a block. Our above analysis assumes that the contents of the blocks do not affect the incentives of the attacker and the designer. In the real-world execution of Bitcoin, however, this is not the case as the contents of the blocks are money-like transactions and have transaction fees associated with them. We model these using positive-valued function $\mathtt{tx} \mapsto f(\mathtt{tx})$ mapping individual transactions to a positive real value that are integer multiples of 1 *Satoshi* (equals 10^{-8} Bitcoin).[29] For sake of brevity, we will also denote by $\hat{f}(\mathtt{st}) := \sum_{\mathtt{tx} \in \mathtt{st}} f(\mathtt{tx})$ the sum of all fees contained in the state block $\mathtt{st}$. The fees have to be considered when defining the utilities in a rational analysis since they are added to the (flat) block reward and the total sum is given as a reward to the miner

[29] Note that this modeling aspect is not sensitive to the basic unit of measurement.

who inserts the block into the ledger state. Hence, this section treats the case where overall block rewards can be a dynamic quantity. In fact, the plan for Bitcoin is to eventually drop the block rewards at which point mining will be incentivized exclusively by the associated transaction fees. In this section we study the security and stability of the Bitcoin network incorporating also such fees.

5.1 Utility Functions with Fees

We first have to change the definition of the utility functions to incorporate that the attacker and the designer receive a different reward when inserting a block into the ledger state. The difference are the transactions fees. To this end, we first introduce a set $\mathcal{T}_{\mathcal{Z}}$ which contains all transactions that are submitted by the environment (and in particular not by the adversary), and then define the relevant events to capture fees in our model.[30]

– In an execution, let $\mathcal{T}_{\mathcal{Z}}$ be the set of transactions such that $\texttt{tx} \in \mathcal{T}_{\mathcal{Z}}$ if and only if $\texttt{tx}$ first appeared as an input from the environment (i.e., the first occurrence of $\texttt{tx}$ is in a command $(\textsc{submit}, \texttt{tx})$ in this execution).
– For each $(\mu, r) \in \mathbb{N}^2$ the event $F^{\mathtt{A}}_{r,\mu}$ is defined as follows: $F^{\mathtt{A}}_{r,\mu}$ denotes the event that the total sum of the transaction fees $f(\texttt{tx})$ of all $\texttt{tx} \in \mathcal{T}_{\mathcal{Z}}$ contained in the blocks that the adversary adds to the state in round r is equal to $\mu \cdot 10^{-8} \cdot \texttt{CR}$ cost units.[31]
– For each $(\mu, r) \in \mathbb{N}^2$ let the event $F^{\mathtt{D}}_{r,\mu}$ be defined as follows: $F^{\mathtt{D}}_{r,\mu}$ is the event that the total sum of the transaction fees $f(\texttt{tx})$ of all $\texttt{tx} \in \mathcal{T}_{\mathcal{Z}}$ contained in the blocks that the honest miners (jointly) add to the state in round r is equivalent to $\mu \cdot 10^{-8} \cdot \texttt{CR}$ cost units.

Since it is the environment that decides on the block-content, the sum of the fees in each block is effectively a random variable whose distribution is induced by the environment. The utilities of the attacker and designer that incorporate fees are defined as follows (we use $\hat{u}^{\text{₿}}_{\mathtt{A}}$ and $\hat{u}^{\text{₿}}_{\mathtt{D}}$ to denote the utilities when fees are added to the incentives):

$$\hat{u}^{\text{₿}}_{\mathtt{A}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z} \in \texttt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \texttt{breward} \cdot \texttt{CR} \cdot E(B^{\mathtt{A}}) - q \cdot \texttt{mcost} \cdot E(Q^{\mathtt{A}}) \right.\right.$$
$$\left.\left. + \sum_{(\mu,r) \in \mathbb{N}^2} \mu \cdot 10^{-8} \cdot \texttt{CR} \cdot \Pr[F^{\mathtt{A}}_{r,\mu}] \right\}\right\}$$

and

$$\hat{u}^{\text{₿}}_{\mathtt{D}}(\Pi, \mathcal{A}) = \inf_{\mathcal{Z} \in \mathbb{Z}} \left\{ \sup_{\mathcal{S}^{\mathcal{A}} \in \mathbb{S}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \texttt{CR} \cdot (\texttt{breward} \cdot \Pr[I^{\mathtt{D}}_{b,r}] - 2^{\mathsf{polylog}(\kappa)} \cdot \Pr[K_r]) \right.\right.$$
$$\left.\left. - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \texttt{mcost} \cdot \Pr[W^{\mathtt{D}}_{q,r}] + \sum_{(\mu,r) \in \mathbb{N}^2} \mu \cdot 10^{-8} \cdot \texttt{CR} \cdot \Pr[F^{\mathtt{D}}_{r,\mu}] \right\}\right\}.$$

Note that the multiplicative factor 10^{-8} is there to allow us to set μ to the integer multiple of one Satoshi that the fee yields. We will denote by $\hat{\mathcal{M}}^{\text{₿}}$ the Bitcoin attack model which has $\mathcal{G}^{\text{₿}}_{\text{LEDGER}}$ as the goal, $\langle \mathcal{G}^{\text{₿}}_{\text{LEDGER}} \rangle$ as the relaxed functionality, and scoring functions for the attacker and designer inducing utilities $\hat{u}^{\text{₿}}_{\mathtt{A}}$ and $\hat{u}^{\text{₿}}_{\mathtt{D}}$.

Upper bounds on fees and total reward for blocks. In reality, transaction fees and the overall reward of a block are naturally bounded (either by size limits or by restricting the total value of the system).[32] In the following, we assume that for all $\texttt{tx}$, $f(\texttt{tx}) \leq \mathsf{max}_{\text{fee}}$, and that the sum of fees per block is bounded, yielding an upper bound on the total profit per block: For all state blocks $\texttt{st}$ we require that $\texttt{breward} + \hat{f}(\texttt{st}) \leq \mathsf{max}_{\text{block}}$, where $\mathsf{max}_{\text{fee}}$ and $\mathsf{max}_{\text{block}}$ are (strictly) positive multiples of one Satoshi.

[30] Note that we assume that only transactions submitted by the environment can yield fees, since the environment models "the application layer". In particular, if the adversary creates a transaction on his own and includes it in his next mined block, then this should not assign him any payoff.
[31] Recall that $\texttt{CR}$ is the conversion of one cryptocurrency unit (e.g., one bitcoin) to one cost unit (e.g., one US dollar).
[32] For example, the number of total Bitcoins is limited and the block-size is bounded.

Restrictions on the availability of transactions. So far in our treatment, the environment induces a distribution on the available transactions and is in principle unrestricted in doing so. For example, the set $\mathcal{T}_{\mathcal{Z}}$ is not bounded in size except by the running time of $\mathcal{Z}$. As will become apparent below in Theorem 5, putting no restrictions on the set $\mathcal{T}_{\mathcal{Z}}$ can still lead to meaningful statements that apply, for example, to applications that are believed to generate an (a priori) unbounded number of transactions. However, to model different kinds of scenarios that appear in the real world, we have to develop a language that allows us to speak about limited availability of transactions. To this end, we introduce parameterized environments $\mathcal{Z}^{\mathcal{D}}$. More precisely, let $\mathcal{D}$ be an oracle which takes inputs $(\textsc{NextTxs}, r)$ and returns a vector $\vec{T}_r = (\mathtt{tx}_1, p_{i_1}), \ldots, (\mathtt{tx}_k, p_{i_k})$. We say that an environment is $\mathcal{D}$-respecting, if, in every round r, the environment queries the oracle $\mathcal{D}$ and only transactions $\mathtt{tx} \in \vec{T}_r$ are added to $\mathcal{T}_{\mathcal{Z}}$. We further require that $\mathcal{Z}$ submits $(\textsc{submit}, \mathtt{tx}_i)$ to party p_k in round r if and only if $(\mathtt{tx}_i, p_k) \in \vec{T}_r$. For simplicity, we call $\mathcal{D}$ simply a distribution. The utility for the attacker in such environments is taken to be the supremum as above, but only over all $\mathcal{D}$-respecting environments.

5.2 Analysis of Bitcoin (with Fees)

The following theorem says that if we look at unrestricted environments, then Bitcoin is still incentive compatible. This is a consequence of Theorem 2 and Theorem 4 and proven formally in Appendix D.

Theorem 5. *Consider arbitrary environments and let the sum of the transaction fees per block be bounded by* $\mathsf{max}_{\text{block}} > 0$. *Then the Bitcoin protocol is strongly attack-payoff secure in the attack model* $\hat{\mathcal{M}}^{\text{B}}$. *It is further incentive-compatible with respect to the class of protocols that are compatible with the Bitcoin network under the same conditions as in Theorem 4), i.e., if*

$$\mathtt{breward} \cdot \mathtt{CR} > \frac{\mathtt{mcost}}{p \cdot (1-p)^{n-1}}.$$

The previous statement is void in case the flat block reward is 0. However, for certain types of distributions $\mathcal{D}$, namely, the ones that provide sufficient high-fee transactions to the participants, it will remain in an equilibrium state. The statement is proven in Section D.

Theorem 6. *Consider distributions $\mathcal{D}$ with the following property: In every round, $\mathcal{D}$ outputs a vector of transactions such that any party gets as input a list of transactions to build a valid next state block* $\mathtt{st}$ *to extend the longest chain and such that* $\hat{f}(\mathtt{st}) = \mathsf{max}_{\text{block}}$ *holds (where* $\mathsf{max}_{\text{block}} > 0$*). Then, with respect to $\mathcal{D}$-respecting environments, the Bitcoin protocol is strongly attack-payoff secure in the attack model* $\hat{\mathcal{M}}^{\text{B}}$. *It is further incentive compatible with respect to the class of protocols that are compatible with the Bitcoin network (as defined in Theorem 4) if* $\mathsf{max}_{\text{block}} \cdot \mathtt{CR} > \frac{\mathtt{mcost}}{p \cdot (1-p)^{n-1}}$.

However, if an application cannot provide enough transactions, it becomes problematic, as the following counterexample shows.

Theorem 7. *There exist distributions $\mathcal{D}$ such that the Bitcoin protocol is neither attack-payoff secure nor strongly attack-payoff secure with respect to $\mathcal{D}$-respecting environments.*

Proof. The proof is straightforward and follows from a general observation: assume there is just a single transaction in the network which has been received only by a corrupted party p_i. Then, the adversary does not publish this transaction to the network. If he does not, then he will be the one claiming the reward with probability one, which is his best choice. Hence, he does not follow the protocol (as the semi-honest front-running adversary would do) and hence it cannot be strongly attack-payoff secure.

Furthermore, the protocol is also not attack-payoff secure. If the honest-majority assumption does not hold, and thus an adversary can fork the ledger state, he would exercise his power to create a ledger state where it is a corrupted party who mines the block containing the only transaction in the system as this will yield better reward than simply mining on empty blocks. □

Fallback security. Note that because cryptographic security trivially implies attack-payoff security for all possible environments and utilies, we can easily derive a fallback security notion: If the majority of miners mines honestly, then we get attack-payoff security; and even if this fails, we still get attack-payoff security under the assumption that the distribution of the fees and the relation between rewards vs costs vs conversion rate are as in Theorem 5 or 6.

References

[1] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, August 2017.

[2] Github: Bitcoin Core version 0.12.0. Wallet: Transaction fees. `https://github.com/bitcoin/bitcoin/blob/v0.12.0/doc/release-notes.md#wallet-transaction-fees`.

[3] Joseph Bonneau. Why buy when you can rent? - Bribery attacks on bitcoin-style consensus. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 19–26. Springer, Heidelberg, February 2016.

[4] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[5] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[6] Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 154–167. ACM Press, October 2016.

[7] Ittay Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy*, pages 89–103. IEEE Computer Society Press, May 2015.

[8] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, March 2014.

[9] Georg Fuchsbauer, Jonathan Katz, and David Naccache. Efficient rational secret sharing in standard communication networks. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 419–436. Springer, Heidelberg, February 2010.

[10] Juan A. Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th FOCS*, pages 648–657. IEEE Computer Society Press, October 2013.

[11] Juan A. Garay, Jonathan Katz, Björn Tackmann, and Vassilis Zikas. How fair is your protocol? A utility-based approach to protocol optimality. In Chryssis Georgiou and Paul G. Spirakis, editors, *34th ACM PODC*, pages 281–290. ACM, July 2015.

[12] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

[13] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323. Springer, Heidelberg, August 2017.

[14] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 3–16. ACM Press, October 2016.

[15] Oded Goldreich. *Foundations of Cryptography: Volume1, Basic Tools.* 2003.

[16] Ronen Gradwohl, Noam Livne, and Alon Rosen. Sequential rationality in cryptographic protocols. In *51st FOCS*, pages 623–632. IEEE Computer Society Press, October 2010.

[17] Joseph Y. Halpern, Rafael Pass, and Lior Seeman. Computational extensive-form games. In *EC*, 2016.

[18] Jonathan Katz. Bridging game theory and cryptography: Recent results and future directions (invited talk). In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 251–272. Springer, Heidelberg, March 2008.

[19] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

[20] Gillat Kol and Moni Naor. Games for exchanging information. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 423–432. ACM Press, May 2008.

[21] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 706–719. ACM Press, October 2015.

[22] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `http://bitcoin.org/bitcoin.pdf`.

[23] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *S&P*, 2016.

[24] Shien Jin Ong, David C. Parkes, Alon Rosen, and Salil P. Vadhan. Fairness with an honest minority and a rational majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 36–53. Springer, Heidelberg, March 2009.

[25] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* MIT Press.

[26] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, May 2017.

[27] Rafael Pass and Elaine Shi. FruitChains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *36th ACM PODC*, pages 315–324. ACM, July 2017.

[28] Meni Rosenfeld. Analysis of bitcoin pooled mining reward systems. *CoRR*, 2011.

[29] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In Jens Grossklags and Bart Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 515–532. Springer, Heidelberg, February 2016.

[30] Okke Schrijvers, Joseph Bonneau, Dan Boneh, and Tim Roughgarden. Incentive compatibility of bitcoin mining pool reward functions. In Jens Grossklags and Bart Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 477–498. Springer, Heidelberg, February 2016.

[31] Jason Teutsch, Sanjay Jain, and Prateek Saxena. When cryptocurrencies mine their own business. In Jens Grossklags and Bart Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 499–514. Springer, Heidelberg, February 2016.

Appendix

A More Background on the RPD Framework and the Bitcoin Ledger

In this section we introduce some notation and review the basic concepts and definitions from the literature, in particular from [10] and [1] that that are needed for evaluating our results.

We start by reviewing giving mode details on the concepts and definitions of the RPD framework [10], which allows to capture the design of optimally secure protocols against incentive-driven (aka "rational") attackers. In more detail, the RPD framework builds upon simulation-based security in the sense of the UC framework [5]. Recall that in this framework, one typically proves that a given multi-party protocol (such as Bitcoin) *securely realizes* a certain ideal functionality (e.g., the "ledger" functionality). Next, we briefly recap the recent result in [1], where a universally composable treatment of the Bitcoin protocol is presented, and which will be the basis for our RPD treatment.

A.1 More Details on the RPD Framework

In [10], Garay *et al.* capture incentive-driven adversaries by casting attacks as a *meta-game* between two rational players, the protocol designer D and the attacker A, which we now describe.

The attack game $\mathcal{G}_{\mathcal{M}}$. The game is parameterized by a (multi-party) functionality $\mathcal{F}$ known to both agents D and A which corresponds to the ideal goal the designer is trying to achieve (and the attacker to break.) Looking ahead, when we analyze Bitcoin, $\mathcal{F}$ will be a ledger functionality (cf. [1]). The designer D chooses a protocol for realizing the functionality $\mathcal{F}$ from the set of all probabilistic and polynomial-time (PPT) computable protocols, where a protocol consists of the code that the (honest) parties are supposed to execute. D sends to A the description $\Pi \subseteq \{0,1\}^*$ of this protocol.[33] Upon receiving Π, A chooses a PPT interactive Turing machine (ITM) $\mathcal{A}$ to attack protocol Π. The set Z of possible terminal histories is then the set of sequences of pairs $(\Pi, \mathcal{A})$, where Π is an n-party protocol, and $\mathcal{A}$ is an adversarial strategy for attacking Π in a traditional cryptographic definition.

Consistently with [10], we denote the corresponding attack game by $\mathcal{G}_{\mathcal{M}}$, where $\mathcal{M}$ is referred to as the *attack model*, which specifies all the public parameters of the game, namely: (1) the functionality, (2) the description of the relevant action sets, and (3) the utilities assigned to certain actions (see below.)

For defining stability, RPD uses the standard solution concept for extensive games with perfect information, namely, *subgame-perfect equilibrium*, where, informally, the actions of each party *at any point in the game* (i.e., after any history) form a best response to this history (cf. [25, Definition 97.2]: Nash-equilibrium of a Stackelberg game). However, as we are interested in cryptographic security definitions with negligible error terms, this notion is refined to *ϵ-subgame perfect equilibrium*, which considers as solutions all profiles in which the parties' utilities are ϵ-close to their best-response utilities (see [10] for a formal definition.) Throughout this paper we will only consider $\epsilon = \text{negl}(\kappa)$; in slight abuse of notation, we will refer to negl(κ)-*subgame perfect equilibrium* simply as subgame perfect.

The utilities. The core novelty of the RPD framework is in how utilities are defined. Since the underlying game is zero-sum, it suffices to define the attacker's utility. This utility depends on the goals of the attacker, more precisely, the security breaches which he succeeds to provoke. As a first attempt, one might try to identify such breaches in the protocol transcript and assign payoffs to them. However, as noted in [10], certain security breaches, such as learning noticeable information on the honest parties' inputs, cannot be decided by looking at the protocol's transcript alone. To solve this issue, [10] uses the simulation paradigm for assigning a utility to the attacker. Intuitively, the idea is to map attacks to the protocol onto attacks in the ideal world and reward the attacker when this mapping yields a payoff. In more detail, the attacker's utility is defined via the following three-step process:

[33] Note that this description includes also the protocol's hybrids.

1. The first step is to modify the ideal functionality $\mathcal{F}$ to obtain a (possibly weaker) ideal functionality $\langle\mathcal{F}\rangle$, which explicitly allows the attacks we wish to model. For example, $\langle\mathcal{F}\rangle$ could give its simulator access to the parties' inputs. (This allows to score attacks that aim at input-privacy breaches.)

2. The second step induces a scoring mechanism for the different breaches that are of interest to the adversary. Specifically, it defines a function $v_{\mathtt{A}}$ mapping the joint view of the relaxed functionality $\langle\mathcal{F}\rangle$ and the environment $\mathcal{Z}$ to a real-valued *payoff*. Given this mapping one can then define the random variable (ensemble) $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ as the result of applying $v_{\mathtt{A}}$ to the views of $\langle\mathcal{F}\rangle$ and $\mathcal{Z}$ in a random experiment describing an ideal evaluation with ideal-world adversary $\mathcal{S}$. In other words, $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ describes (as a random variable ensemble) the payoff of $\mathcal{S}$ in an execution using directly the functionality $\langle\mathcal{F}\rangle$.[34] The *attacker's (ideal) expected payoff* for simulator $\mathcal{S}$ and environment $\mathcal{Z}$ is defined to be the expected value of $v^{\mathtt{A}}_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$, i.e.,
$$U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z}) = E(v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}).$$
The triple $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v^{\mathtt{A}})$ constitutes the *attack model.*

3. In the third and final step we use the above (ideal) expected payoff to define the *real expected payoff* of the attacker—and via that his utility—for a given strategy profile $(\mathit{\Pi}, \mathcal{A})$: The *attacker's (real) expected payoff* for a strategy profile $(\mathit{\Pi}, \mathcal{A})$ with respect to environment $\mathcal{Z}$, where $\mathit{\Pi}$ realizes $\langle\mathcal{F}\rangle$, is taken to be the payoff of the "best" simulator for $\mathcal{A}$, that is, the simulator that successfully emulates $\mathcal{A}$ while achieving the *minimum* score.[35]

   Formally, for a functionality $\langle\mathcal{F}\rangle$ and a protocol $\mathit{\Pi}$, denote by $\mathcal{C}_{\mathcal{A}}$ the class of simulators that are "good" for $\mathcal{A}$, i.e, $\mathcal{C}_{\mathcal{A}} = \{\mathcal{S} \in \mathtt{ITM} \mid \forall\mathcal{Z} : \mathrm{EXEC}_{\mathit{\Pi},\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}\}$. The *attacker's real expected payoff* for strategy profile $(\mathcal{A}, \mathcal{Z})$ is then defined as
$$U_{R^{\mathtt{A}}}^{\mathit{\Pi},\langle\mathcal{F}\rangle}(\mathcal{A},\mathcal{Z}) = \inf_{\mathcal{S}\in\mathcal{C}_{\mathcal{A}}}\{U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})\}.$$
   In other words, $U_{R^{\mathtt{A}}}^{\mathit{\Pi},\langle\mathcal{F}\rangle}$ assigns to each pair $(\mathcal{A},\mathcal{Z}) \in \mathtt{ITM} \times \mathtt{ITM}$ (and each value of the security parameter κ) a real number corresponding to the expected payoff obtained by $\mathcal{A}$ in attacking $\mathit{\Pi}$ within environment $\mathcal{Z}$. [36]

   Having defined the real expected payoff, $U_{R^{\mathtt{A}}}^{\mathit{\Pi},\langle\mathcal{F}\rangle}(\mathcal{A},\mathcal{Z})$, the (expected) utility of the attacker for an adversary-strategy $\mathcal{A}$, is defined as the (maximal) payoff of $\mathcal{A}$, i.e., its expected payoff when executing the protocol with $\mathcal{A}$'s "preferred" environment, i.e.,
$$u_{\mathtt{A}}(\mathit{\Pi},\mathcal{A}) = \sup_{\mathcal{Z}\in\mathtt{ITM}}\{U_{R^{\mathtt{A}}}^{\mathit{\Pi},\langle\mathcal{F}\rangle}(\mathcal{A},\mathcal{Z})\}.$$

Note that as the views in the above experiments are in fact random variable ensembles indexed by the security parameter κ, the probabilities of all the relative events are in fact functions of κ, hence the utility is also a function of κ.

Remark 5 (Event-based utility [10].). In many applications, however, including those in our work, meaningful payoff functions have the following, simple representation: Let $(E_1, \ldots, E_\ell)$ denote a vector of (disjoint) events defined on the views (of $\mathcal{S}$ and $\mathcal{Z}$) in the ideal experiment corresponding to the security breaches that contribute to the attacker's utility. Each event E_i is assigned a real number γ_i, and the payoff function $v_{\mathtt{A}}^{\vec{\gamma}}$ assigns, to each ideal execution, the sum of γ_i's for which E_i occurred. The ideal expected payoff of a simulator is computed according to our definition as
$$U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z}) = \sum_{E_i\in\vec{E},\gamma_i\in\vec{\gamma}} \gamma_i \Pr[E_i],$$
where the probabilities are taken over the random coins of $\mathcal{S}$, $\mathcal{Z}$, and $\langle\mathcal{F}\rangle$.

[34] Recall that the views in the above experiment are random variable ensembles indexed by the security parameter.
[35] Refer to [10] for an explanation of why the minimizing simulator is the right choice.
[36] For adversaries $\mathcal{A}$ with $\mathcal{C}_{\mathcal{A}} = \emptyset$, the score is ∞ by definition.

RPD security. Building on the above definition of utility, [10] introduces a natural notion of security against incentive-driven attackers. Intuitively, a protocol Π is $u_{\mathtt{A}}$*-attack-payoff secure* in a given attack model $\mathcal{M} = (\mathcal{F}, \cdot, v_{\mathtt{A}})$, if the utility of the best adversary against this protocol is the same as the utility of the best adversary in attacking the $\mathcal{F}$-hybrid "dummy" protocol, which only relays messages between $\mathcal{F}$ and the environment.

Definition 6 (Attack-payoff security [10]). *Let $\mathcal{M} = (\mathcal{F}, \langle \mathcal{F} \rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$ be an attack model inducing utilities $u_{\mathtt{A}}$ and $u_{\mathtt{D}}$ on the attacker and the designer, respectively,*[37] *and let $\phi^{\mathcal{F}}$ be the dummy $\mathcal{F}$-hybrid protocol. A protocol Π is* attack-payoff secure for $\mathcal{M}$ *if for all adversaries $\mathcal{A} \in \mathtt{ITM}$,*

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u_{\mathtt{A}}(\phi^{\mathcal{F}}, \mathcal{A}) + \text{negl}(\kappa).$$

Intuitively, this security definition accurately captures security against an incentive-driven attacker, as in simulating an attack against the dummy $\mathcal{F}$-hybrid protocol, the simulator never needs to provoke any of the "breaching" events. Hence, the utility of the best adversary against Π equals the utility of an adversary that does not provoke any "bad event."

A.2 More Details on the Ledger Functionality

In [1], Badertscher *et al.* present a universally composable treatment of the Bitcoin protocol, Π^{B}, in the UC framework. Here we give more details on the ideal goal that Bitcoin aims to achieve, namely the ledger functionality.

The Bitcoin ledger. The ledger functionality $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$ maintains a ledger state `state`, which is a sequence of state blocks. A state block contains (application-specific) content values—the "transactions." For each honest party, the ledger stores a pointer to a state block which defines the local view of that party onto the ledger state. The functionality ensures that each pointer is not too far away from the head of the state (and that it only moves forward). The ledger allows any party (whether an honest miner or the adversary) to submit transactions which are first validated by means of a predicate $\mathsf{ValidTx}_{\text{B}}$, and, if considered valid, are added to the functionality's buffer (the adversary is informed upon any such action). At any time, the $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$ allows the adversary to propose a candidate state block (by specifying a vector of transactions) it desires to be appended to the ledger state. However, the adversary has to obey the ledger functionality's policy, which is specified by an algorithm $\mathsf{ExtendPolicy}$ that checks whether the proposal is compliant with the policy. If the adversary's proposal does not comply with the ledger policy, $\mathsf{ExtendPolicy}$ will reject the proposal. The policy enforced by the Bitcoin ledger can be succinctly summarized as follows:

- *Ledger's growth.* Within a certain number of rounds the number of added blocks must not be too small or too large. In particular, the adversary cannot wait too long before proposing a new block, otherwise, the ledger will extend the state on its own.
- *Chain quality.* Each proposed block is associated with a flag that, intuitively, indicates whether it was mined using the honest mining process. This policy ensures that a certain fraction of the proposed blocks must be mined this way.
- *Transaction liveness.* If a transaction is old enough and valid, it will be included in a block and added to the ledger state. In particular, the adversary cannot delay valid transaction for an arbitrary number of rounds.

We provide a more formal description of $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$ in Appendix A.3. Refer to [1] for additional details.

The Bitcoin ledger is securely realized by the Bitcoin protocol, or, more specifically, by an abstraction of it called the *ledger protocol*, which casts it as a synchronous UC protocol as explained in the preliminaries of this work.

[37] In [10], by default $u_{\mathtt{D}} = -u_{\mathtt{A}}$ as the game is zero-sum.

A.3 Formal Description of the Bitcoin Ledger Functionality

This section contains a more formal description of the Bitcoin ledger functionality referred to in the text. In particular, we also provide the Bitcoin ExtendPolicy. More details on the ledger functionality can be found in [1].

Functionality $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$

$\mathcal{G}_{\text{LEDGER}}$ is parametrized by four algorithms, $\mathsf{Validate}$, $\mathsf{ExtendPolicy}$, $\mathsf{Blockify}$, and $\mathsf{predict\text{-}time}$, along with two parameters: $\mathtt{windowSize}, \mathtt{Delay} \in \mathbb{N}$. The functionality manages variables $\mathtt{state}$, $\mathtt{NxtBC}$, $\mathtt{buffer}$, τ_L, and $\vec{\tau}_{\mathtt{state}}$, as described above. The variables are initialized as follows: $\mathtt{state} := \vec{\tau}_{\mathtt{state}} := \mathtt{NxtBC} := \varepsilon$, $\mathtt{buffer} := \emptyset$, $\tau_L = 0$.

The functionality maintains the set of registered parties $\mathcal{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and the (sub-set) of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$ (following the definition of de-synchronized of [1]). The sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered, if it is registered with the clock then it is added to the party sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded; if the current time is $\tau_L > 0$, it is also added to $\mathcal{P}_{DS}$. Similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ (and therefore also from $\mathcal{P}_{DS}$ or $\mathcal{H}$). The ledger maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$.
For each party $p_i \in \mathcal{P}$ the functionality maintains a pointer $\mathtt{pt}_i$ (initially set to 1) and a current-state view $\mathtt{state}_i := \varepsilon$ (initially set to empty). The functionality also keeps track of the timed honest-input sequence in a vector $\vec{\mathcal{I}}^T_H$ (initially $\vec{\mathcal{I}}^T_H := \varepsilon$).

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, \tau)$ set $\tau_L := \tau$ and do the following:

1. Let $\hat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered (continuously) since time $\tau' < \tau_L - \mathtt{Delay}$ (to both ledger and clock). Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \hat{\mathcal{P}}$.
2. If I was received from an honest party $p_i \in \mathcal{P}$:
   (a) Set $\vec{\mathcal{I}}^T_H := \vec{\mathcal{I}}^T_H || (I, p_i, \tau_L)$;
   (b) Compute $\vec{N} = (\vec{N}_1, \dots, \vec{N}_\ell) := \mathsf{ExtendPolicy}(\vec{\mathcal{I}}^T_H, \mathtt{state}, \mathtt{NxtBC}, \mathtt{buffer}, \vec{\tau}_{\mathtt{state}})$ and if $\vec{N} \neq \varepsilon$ set $\mathtt{state} := \mathtt{state} || \mathsf{Blockify}(\vec{N}_1) || \dots || \mathsf{Blockify}(\vec{N}_\ell)$ and $\vec{\tau}_{\mathtt{state}} := \vec{\tau}_{\mathtt{state}} || \tau^\ell_L$, where $\tau^\ell_L = \tau_L || \dots, || \tau_L$.
   (c) For each $\mathtt{BTX} \in \mathtt{buffer}$: if $\mathsf{Validate}(\mathtt{BTX}, \mathtt{state}, \mathtt{buffer}) = 0$ then delete $\mathtt{BTX}$ from $\mathtt{buffer}$. Also, reset $\mathtt{NxtBC} := \varepsilon$.
   (d) If there exists $p_j \in \mathcal{H} \setminus \mathcal{P}_{DS}$ such that $|\mathtt{state}| - \mathtt{pt}_j > \mathtt{windowSize}$ or $\mathtt{pt}_j < |\mathtt{state}_j|$, then set $\mathtt{pt}_k := |\mathtt{state}|$ for all $p_k \in \mathcal{H} \setminus \mathcal{P}_{DS}$.
3. Depending on the above input I and its sender's ID, $\mathcal{G}_{\text{LEDGER}}$ executes the corresponding code from the following list:
   - *Submiting a transaction:*
     If $I = (\text{SUBMIT}, \text{sid}, \mathtt{tx})$ and is received from a party $p_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of a corrupted party p_i) do the following
     (a) Choose a unique transaction ID txid and set $\mathtt{BTX} := (\mathtt{tx}, \text{txid}, \tau_L, p_i)$
     (b) If $\mathsf{Validate}(\mathtt{BTX}, \mathtt{state}, \mathtt{buffer}) = 1$, then $\mathtt{buffer} := \mathtt{buffer} \cup \{\mathtt{BTX}\}$.
     (c) Send $(\text{SUBMIT}, \mathtt{BTX})$ to $\mathcal{A}$.
   - *Reading the state:*
     If $I = (\text{READ}, \text{sid})$ is received from a party $p_i \in \mathcal{P}$ then set $\mathtt{state}_i := \mathtt{state}|_{\min\{\mathtt{pt}_i, |\mathtt{state}|\}}$ and return $(\text{READ}, \text{sid}, \mathtt{state}_i)$ to the requestor. If the requestor is $\mathcal{A}$ then send $(\mathtt{state}, \mathtt{buffer}, \vec{\mathcal{I}}^T_H)$ to $\mathcal{A}$.
   - *Maintaining the ledger state:*
     If $I = (\text{MAINTAIN-LEDGER}, \text{sid}, \text{minerID})$ is received by an honest party $p_i \in \mathcal{P}$ and (after updating $\vec{\mathcal{I}}^T_H$ as above) $\mathsf{predict\text{-}time}(\vec{\mathcal{I}}^T_H) = \hat{\tau} > \tau_L$ then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. Else send I to $\mathcal{A}$.

- *The adversary proposing the next block:*
  If $I = (\textsc{next-block}, \text{hFlag}, (\text{txid}_1, \ldots, \text{txid}_\ell))$ is sent from the adversary, update $\texttt{NxtBC}$ as follows:
  (a) Set $\text{listOfTxid} \leftarrow \epsilon$
  (b) For $i = 1, \ldots, \ell$ do: if there exists $\texttt{BTX} := (x, \text{txid}, \text{minerID}, \tau_L, p_i) \in \texttt{buffer}$ with ID $\text{txid} = \text{txid}_i$ then set $\text{listOfTxid} := \text{listOfTxid}||\text{txid}_i$.
  (c) Finally, set $\texttt{NxtBC} := \texttt{NxtBC}||(\text{hFlag}, \text{listOfTxid})$ and output $(\textsc{next-block}, ok)$ to $\mathcal{A}$.
- *The adversary setting state-slackness:*
  If $I = (\textsc{set-slack}, (p_{i_1}, \hat{\texttt{pt}}_{i_1}), \ldots, (p_{i_\ell}, \hat{\texttt{pt}}_{i_\ell}))$, with $\{p_{i_1}, \ldots, p_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$ do the following:
  (a) If for all $j \in [\ell] : |\texttt{state}| - \hat{\texttt{pt}}_{i_j} \leq \texttt{windowSize}$ and $\hat{\texttt{pt}}_{i_j} \geq |\texttt{state}_{i_j}|$, set $\texttt{pt}_{i_1} := \hat{\texttt{pt}}_{i_1}$ for every $j \in [\ell]$ and return $(\textsc{set-slack}, ok)$ to $\mathcal{A}$.
  (b) Otherwise set $\texttt{pt}_{i_j} := |\texttt{state}|$ for all $j \in [\ell]$.
- *The adversary setting the state for desychronized parties:*
  If $I = (\textsc{desync-state}, (p_{i_1}, \texttt{state}'_{i_1}), \ldots, (p_{i_\ell}, \texttt{state}'_{i_\ell}))$, with $\{p_{i_1}, \ldots, p_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$, set $\texttt{state}_{i_j} := \texttt{state}'_{i_j}$ for each $j \in [\ell]$ and return $(\textsc{desync-state}, ok)$ to $\mathcal{A}$.

The ExtendPolicy for Bitcoin is given below. The extend policy is parameterized by the ledger parameters that control the speed and the quality of the ledger.

Algorithm ExtendPolicy for $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$

```
function ExtendPolicy($\vec{\mathcal{I}}_H^T$, state, NxtBC, buffer, $\vec{\tau}_{\texttt{state}}$)
    Let $\tau_L$ be current ledger time (computed from $\vec{\mathcal{I}}_H^T$)
    // The function must not have side-effects: Only modify copies of relevant values.
    Create local copies of the values buffer, state, and $\vec{\tau}_{\texttt{state}}$.
    // First, create a default honest client block as alternative:
    Set $\vec{N}_{\texttt{df}} \leftarrow \texttt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
    Sort buffer according to time stamps.
    Let $\vec{\texttt{tx}} = (\texttt{tx}_1, \ldots, \texttt{tx}_\ell)$ be the transactions in buffer
    Set $\texttt{st} \leftarrow \mathsf{blockify}_{\text{B}}(\vec{N}_{\texttt{df}})$
    repeat
        Let $\vec{\texttt{tx}} = (\texttt{tx}_1, \ldots, \texttt{tx}_\ell)$ be the current list of (remaining) transactions
        for $i = 1$ to $\ell$ do
            if $\mathsf{ValidTx}_{\text{B}}(\texttt{tx}_i, \texttt{state}||\texttt{st}) = 1$ then
                $\vec{N}_{\texttt{df}} \leftarrow \vec{N}_{\texttt{df}}||\texttt{tx}_i$
                Remove $\texttt{tx}_i$ from $\vec{\texttt{tx}}$
                Set $\texttt{st} \leftarrow \mathsf{blockify}_{\text{B}}(\vec{N}_{\texttt{df}})$
            end if
        end for
    until $\vec{N}_{\texttt{df}}$ does not increase anymore
    // Possibly more than one block have to be added
    // Let $\tau_{\texttt{low}}$ be the time of the block which is windowSize $- 1$ blocks behind the head of the state.
    if $|\texttt{state}| + 1 \geq \texttt{windowSize}$ then
        Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + 2]$
    else
        Set $\tau_{\texttt{low}} \leftarrow 0$
    end if
    $c \leftarrow 1$
    while $\tau_L - \tau_{\texttt{low}} > \texttt{maxTime}_{\texttt{window}}$ do
        Set $\vec{N}_c \leftarrow \texttt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
        $\vec{N}_{\texttt{df}} \leftarrow \vec{N}_{\texttt{df}}||\vec{N}_c$
        $c \leftarrow c + 1$
        // Update $\tau_{\texttt{low}}$ to the time of the state block which is $\texttt{windowSize} - c$ blocks behind the head.
        if $|\texttt{state}| + c \geq \texttt{windowSize}$ then
            Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + c + 1]$
        else
            Set $\tau_{\texttt{low}} \leftarrow 0$
        end if
    end while
    // Now, parse the proposed block by the adversary
    // Possibly more than one block should be added
    Parse NxtBC as a vector $((\text{hFlag}_1, \texttt{NxtBC}_1), \cdots, (\text{hFlag}_n, \texttt{NxtBC}_n))$
    $\vec{N} \leftarrow \varepsilon$    // Initialize Result
    // Determine the time of the state block which is windowSize blocks behind the head of the state
    if $|\texttt{state}| \geq \texttt{windowSize}$ then
        Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + 1]$
    else
        Set $\tau_{\texttt{low}} \leftarrow 0$
    end if
    $\mathsf{oldValidTxMissing} \leftarrow \mathsf{false}$    // Flag to keep track whether old enough, valid transactions are inserted.
    for each list $\texttt{NxtBC}_i$ of transaction IDs do
        // Compute the next state block
        $\vec{N}_i \leftarrow \varepsilon$
        // Verify validity of $\texttt{NxtBC}_i$ and compute content
        Use the txid contained in $\texttt{NxtBC}_i$ to determine the list of transactions
        Let $\vec{\texttt{tx}} = (\texttt{tx}_1, \ldots, \texttt{tx}_{|\texttt{NxtBC}_i|})$ denote the transactions of $\texttt{NxtBC}_i$
        if $\texttt{tx}_1$ is not a coin-base transaction then
            return $\vec{N}_{\texttt{df}}$
        else
            $\vec{N}_i \leftarrow \texttt{tx}_1$
            for $j = 2$ to $|\texttt{NxtBC}_i|$ do
                Set $\texttt{st}_i \leftarrow \mathsf{blockify}_{\text{B}}(\vec{N}_i)$
                if $\mathsf{ValidTx}_{\text{B}}(\texttt{tx}_j, \texttt{state}||\texttt{st}_i) = 0$ then
                    return $\vec{N}_{\texttt{df}}$
                end if
                $\vec{N}_i \leftarrow \vec{N}_i||\texttt{tx}_j$
            end for
            Set $\texttt{st}_i \leftarrow \mathsf{blockify}_{\text{B}}(\vec{N}_i)$
        end if
        // Test that all old valid transaction are included
        if the proposal is declared to be an honest block, i.e., $\text{hFlag}_i = 1$ then
            for each $\texttt{BTX} = (\texttt{tx}, \text{txid}, \tau', p_i) \in \texttt{buffer}$ of an honest party $p_i$ with time $\tau' < \tau_{\texttt{low}} - \frac{\texttt{Delay}}{2}$ do
                if $\mathsf{ValidTx}_{\text{B}}(\texttt{tx}, \texttt{state}||\texttt{st}_i) = 1$ but $\texttt{tx} \notin \vec{N}_i$ then
                    $\mathsf{oldValidTxMissing} \leftarrow \mathsf{true}$
                end if
            end for
        end if
        $\vec{N} \leftarrow \vec{N}||\vec{N}_i$
        $\texttt{state} \leftarrow \texttt{state}||\texttt{st}_i$
        $\vec{\tau}_{\texttt{state}} \leftarrow \vec{\tau}_{\texttt{state}}||\tau_L$
        // Must not proceed with too many adversarial blocks
        Determine the most recent honest block $\texttt{st}_j$ in $\texttt{state}$ (last proposal added with $\text{hFlag} = 1$).
        if $|\texttt{state}| - j \geq \eta$ then
            return $\vec{N}_{\texttt{df}}$
        end if
```

```
        // Determine $\tau_{\texttt{low}}$: the time of the state block which is windowSize blocks behind the head of the
        // current, potentially already extended state
        if $|\texttt{state}| \geq \texttt{windowSize}$ then
            Set $\tau_{\texttt{low}} \leftarrow \vec{\tau}_{\texttt{state}}[|\texttt{state}| - \texttt{windowSize} + 1]$
        else
            Set $\tau_{\texttt{low}} \leftarrow 0$
        end if
    end for
    // Final checks (if policy is violated, it is enforced by the ledger):
    // Must not proceed too fast, too slow, or with missing transactions.
    if $\tau_L - \tau_{\texttt{low}} < \texttt{minTime}_{\texttt{window}}$ then
        return $\varepsilon$
    else if $\tau_{\texttt{low}} > 0$ and $\tau_L - \tau_{\texttt{low}} > \texttt{maxTime}_{\texttt{window}}$ then // A sequence of blocks cannot take too much time.
        return $\vec{N}_{\texttt{df}}$
    else if $\tau_{\texttt{low}} = 0$ and $\tau_L - \tau_{\texttt{low}} > 2 \cdot \texttt{maxTime}_{\texttt{window}}$ then // Bootstrapping cannot take too much time.
        return $\vec{N}_{\texttt{df}}$
    else if oldValidTxMissing then // If not all old enough, valid transactions have been included.
        return $\vec{N}_{\texttt{df}}$
    end if
    return $\vec{N}$
end function
```

B The Weak Bitcoin Ledger

In this section we formally describe the weakened Bitcoin ledger functionality $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$.

Functionality $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$

$\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$ is parametrized by four algorithms, Validate, weakExtendPolicy, Blockify, and predict-time, along with two parameters: $\texttt{windowSize}, \texttt{Delay} \in \mathbb{N}$. The functionality manages variables `state-tree`, `NxtBC`, `buffer`, and τ_L, where `state-tree` is a tree of state blocks. The variables are initialized as follows: $\texttt{state-tree} = \texttt{gen}$, $\texttt{NxtBC} := \varepsilon$, $\texttt{buffer} := \emptyset$, $\tau_L = 0$.

The functionality maintains the set of registered parties $\mathcal{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and the (sub-set) of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$ (following the definition of de-synchronized of [1]). The sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered, if it is registered with the clock then it is added to the party sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded; if the current time is $\tau_L > 0$, it is also added to $\mathcal{P}_{DS}$. Similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ (and therefore also from $\mathcal{P}_{DS}$ or $\mathcal{H}$). The ledger maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$.
For each party $p_i \in \mathcal{P}$ the functionality maintains a pointer $\texttt{pt}_i$ (initially set to the root of `state-tree`) which defines the current-state view $\texttt{state}_i$ of p_i. The functionality also keeps track of the timed honest-input sequence in a vector $\vec{\mathcal{I}}^T_H$ (initially $\vec{\mathcal{I}}^T_H := \varepsilon$).

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, \tau)$ set $\tau_L := \tau$ and do the following:

1. Let $\hat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered (continuously) since time $\tau' < \tau_L - \texttt{Delay}$ (to both ledger and clock). Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \hat{\mathcal{P}}$.
2. If I was received from an honest party $p_i \in \mathcal{P}$:
   (a) Set $\vec{\mathcal{I}}^T_H := \vec{\mathcal{I}}^T_H || (I, p_i, \tau_L)$;
   (b) Evaluate $\vec{R}$= $((\vec{N}_{\texttt{pt}_1}, \texttt{pt}_1, \ldots, (\vec{N}_{\texttt{pt}_k}, \texttt{pt}_k)) :=$ weakExtendPolicy$(\vec{\mathcal{I}}^T_H, \texttt{state-tree}, \texttt{NxtBC}, \texttt{buffer})$

(c) For each pointer $\texttt{pt}_i$ such that $\vec{N}_{\texttt{pt}_i} \neq \varepsilon$ add path $\mathsf{Blockify}(\vec{N}_{\texttt{pt}_i,1}), \dots, \mathsf{Blockify}(\vec{N}_{\texttt{pt}_i,\ell})$ to $\texttt{state-tree}$ starting at node $\texttt{pt}_i$.

(d) Reset $\texttt{NxtBC} := \varepsilon$.

3. Depending on the above input I and its sender's ID, $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$ executes the corresponding code from the following list:

- *Submiting a transaction:*
  If $I = (\text{SUBMIT}, \text{sid}, \texttt{tx})$ and is received from a party $p_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of a corrupted party p_i) do the following
  (a) Choose a unique transaction ID txid and set $\texttt{BTX} := (\texttt{tx}, \text{txid}, \tau_L, p_i)$
  (b) Set $\texttt{buffer} := \texttt{buffer} \cup \{\texttt{BTX}\}$ and send $(\text{SUBMIT}, \texttt{BTX})$ to $\mathcal{A}$.

- *Reading the state:*
  If $I = (\text{READ}, \text{sid})$ is received from a party $p_i \in \mathcal{P}$ return $(\text{READ}, \text{sid}, \texttt{state}_i)$ to the requestor. If the requestor is $\mathcal{A}$ then send $(\texttt{state-tree}, \texttt{buffer}, \vec{\mathcal{I}}^T_H)$ to $\mathcal{A}$.

- *Maintaining the ledger state:*
  If $I = (\text{MAINTAIN-LEDGER}, \text{sid}, \text{minerID})$ is received by an honest party $p_i \in \mathcal{P}$ and (after updating $\vec{\mathcal{I}}^T_H$ as above) $\mathsf{predict\text{-}time}(\vec{\mathcal{I}}^T_H) = \hat{\tau} > \tau_L$ then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. Else send I to $\mathcal{A}$.

- *The adversary proposing the next block:*
  If $I = (\text{NEXT-BLOCK}, \texttt{pt}, \text{hFlag}, (\text{txid}_1, \dots, \text{txid}_\ell))$ is sent from the adversary, update $\texttt{NxtBC}$ as follows:
  (a) Check that $\texttt{pt}$ points to a leaf of $\texttt{state-tree}$ and set listOfTxid $\leftarrow \varepsilon$ (otherwise, ignore command)
  (b) For $i = 1, \dots, \ell$ do: if there exists $\texttt{BTX} := (x, \text{txid}, \text{minerID}, \tau_L, p_i) \in \texttt{buffer}$ with ID txid $= \text{txid}_i$ then set listOfTxid := listOfTxid$||$txid$_i$.
  (c) Finally, set $\texttt{NxtBC}[\texttt{pt}] := \texttt{NxtBC}[\texttt{pt}]||(\text{hFlag}, \text{listOfTxid})$ and output $(\text{NEXT-BLOCK}, ok)$ to $\mathcal{A}$.

- *The adversary proposing a fork:*
  If $I = (\text{FORK}, \texttt{pt}, (\text{txid}_1, \dots, \text{txid}_\ell))$ is sent from the adversary, update $\texttt{NxtBC}$ as follows:
  (a) Set listOfTxid $\leftarrow \varepsilon$
  (b) For $i = 1, \dots, \ell$ do: if there exists $\texttt{BTX} := (x, \text{txid}, \text{minerID}, \tau_L, p_i) \in \texttt{buffer}$ with ID txid $= \text{txid}_i$ then set listOfTxid := listOfTxid$||$txid$_i$.
  (c) Finally, set $\texttt{NxtBC}[\texttt{pt}] := \texttt{NxtBC}[\texttt{pt}]||(0, \text{listOfTxid})$ and output (FORK, ok) to $\mathcal{A}$.

- *The adversary setting state-slackness:*
  If $I = (\text{SET-POINTER}, (p_{i_1}, \hat{\texttt{pt}}_{i_1}), \dots, (p_{i_\ell}, \hat{\texttt{pt}}_{i_\ell}))$, with $\{p_{i_1}, \dots, p_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$ do the following:
  (a) If for all $j \in [\ell] : \hat{\texttt{pt}}_{i_j}$ has greater distance than $\texttt{pt}_{i_j}$ from the root $\texttt{state-tree}$, set$\texttt{pt}_{i_j} := \hat{\texttt{pt}}_{i_j}$ for every $j \in [\ell]$ and return $(\text{SET-SLACK}, ok)$ to $\mathcal{A}$.
  (b) Otherwise set $\texttt{pt}_{i_j}$ to the leaf with greatest distance from the root of $\texttt{state-tree}$.

- *The adversary setting the state for desychronized parties:*
  If $I = (\text{DESYNC-STATE}, (p_{i_1}, \texttt{state}'_{i_1}), \dots, (p_{i_\ell}, \texttt{state}'_{i_\ell}))$, with $\{p_{i_1}, \dots, p_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$, set $\texttt{state}_{i_j} := \texttt{state}'_{i_j}$ for each $j \in [\ell]$ and return $(\text{DESYNC-STATE}, ok)$ to $\mathcal{A}$.

In the following we describe the weakened extend policy for $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$.

Algorithm weakExtendPolicy for $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$

```
function WEAKEXTENDPOLICY($\vec{\mathcal{I}}^T_H$, state-tree, NxtBC, buffer)
    Let $\tau_L$ be current ledger time (computed from $\vec{\mathcal{I}}^T_H$)
    // The function must not have side-effects: Only modify copies of relevant values.
    Create local copies of the values buffer, state-tree, pt and $\vec{\tau}_{\texttt{state}}$.
    // First, create a default honest client block as alternative:
```

```
    Set $\mathtt{pt}_D$ to the leaf of the longest branch of $\mathtt{state\text{-}tree}$ and denote by $\mathtt{state}_D$ the corresponding state.
    Set $\vec{N}_{\mathtt{df}} \leftarrow \mathtt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
    Sort buffer according to time stamps.
    Let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_\ell)$ be the transactions in buffer
    Set $\mathtt{st} \leftarrow \mathsf{blockify}_{\text{₿}}(\vec{N}_{\mathtt{df}})$
    repeat
        Let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_\ell)$ be the current list of (remaining) transactions
        for $i = 1$ to $\ell$ do
            if $\mathsf{ValidTx}_{\text{₿}}(\mathtt{tx}_i, \mathtt{state}_D||\mathtt{st}) = 1$ then
                $\vec{N}_{\mathtt{df}} \leftarrow \vec{N}_{\mathtt{df}}||\mathtt{tx}_i$
                Remove $\mathtt{tx}_i$ from $\vec{\mathtt{tx}}$
                Set $\mathtt{st} \leftarrow \mathsf{blockify}_{\text{₿}}(\vec{N}_{\mathtt{df}})$
            end if
        end for
    until $\vec{N}_{\mathtt{df}}$ does not increase anymore
    // Now, parse the proposed block by the adversary
    // Possibly more than one block should be added, possibly at several places
    $\vec{R} \leftarrow \varepsilon$ // Result variable
    for each pt such that $\mathtt{NxtBC}[\mathtt{pt}] \neq \varepsilon$ do
        Set state to the state corresponding to pointer pt (i.e., a path in state-tree)
        Parse $\mathtt{NxtBC}[\mathtt{pt}]$ as a vector $((\text{hFlag}_1, \mathtt{NxtBC}_1), \cdots, (\text{hFlag}_n, \mathtt{NxtBC}_n))$
        $\vec{N}_{\mathtt{pt}} \leftarrow \varepsilon$     // Initialize Result
        for each list $\mathtt{NxtBC}_i$ of transaction IDs do
            // Compute the next state block
            $\vec{N}_i \leftarrow \varepsilon$
            // Verify validity of $\mathtt{NxtBC}_i$ and compute content
            Use the txid contained in $\mathtt{NxtBC}_i$ to determine the list of transactions
            Let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_{|\mathtt{NxtBC}_i|})$ denote the transactions of $\mathtt{NxtBC}_i$
            if $\mathtt{tx}_1$ is not a coin-base transaction then
                return $(\vec{N}_{\mathtt{df}}, \mathtt{pt}_D)$
            else
                $\vec{N}_i \leftarrow \mathtt{tx}_1$
                for $j = 2$ to $|\mathtt{NxtBC}_i|$ do
                    Set $\mathtt{st}_i \leftarrow \mathsf{blockify}_{\text{₿}}(\vec{N}_i)$
                    if $\mathsf{ValidTx}_{\text{₿}}(\mathtt{tx}_j, \mathtt{state}||\mathtt{st}_i) = 0$ then
                        return $(\vec{N}_{\mathtt{df}}, \mathtt{pt}_D)$
                    end if
                    $\vec{N}_i \leftarrow \vec{N}_i||\mathtt{tx}_j$
                end for
                Set $\mathtt{st}_i \leftarrow \mathsf{blockify}_{\text{₿}}(\vec{N}_i)$
            end if
            $\vec{N}_{\mathtt{pt}} \leftarrow \vec{N}||\vec{N}_i$
            $\mathtt{state} \leftarrow \mathtt{state}||\mathtt{st}_i$
            $\vec{\tau}_{\mathtt{state}} \leftarrow \vec{\tau}_{\mathtt{state}}||\tau_L$
        end for
        $\vec{R} \leftarrow \vec{R}||(\vec{N}_{\mathtt{pt}}, \mathtt{pt})$
        Update (the local copy of) state-tree to include the extended path state
    end for
    return $\vec{R}$
end function
```

B.1 On the secure realization of the weak ledger

It is easy to see that such a weak ledger can be constructed perfectly:

Lemma 4. *The weak Bitcoin ledger $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$ is UC-realized by Π^{B} for any fraction of dishonest parties. In particular, the realization is possible by means of a black-box simulator $\mathcal{S}_{weak}$.*

Proof. First, consider the case where the fraction of dishonest parties is limited as in Theorem 1. In this case, the protocol Π^{B} UC-realizes the stronger ledger $\mathcal{G}^{\text{B}}_{\text{LEDGER}}$. Hence, one can directly use the black-box simulator presented in [1]. If the fraction of dishonest parties is larger the adversary can control which (valid) blocks will enter the ledger state and can set the speed of the ledger. Observe, weakExtendPolicy does not (in contrast to ExtendPolicy) restrict the speed, nor the type of blocks entered into the ledger. This allows to simulate any block pattern which could occur in the real-world using the black-box simulator from the first case. Moreover, in the second case the adversary can fork the state in the real-world. However, this can be simulated using the FORK command of the $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$. In summary, one arrives at a straightforward extension, which we denote by $\mathcal{S}_{weak}$, of the black-box simulator from [1] to simulate the protocol execution of Π^{B} for any number of corrupted parties. □

C The State-Exchange hybrid world of Bitcoin

State Exchange Functionality A convenient step in showing that the Bitcoin protocol realizes the ledger, an important step is to observe that the Bitcoin protocol can be modularized: Π^{B} can be modularized by showing that the sub-process that corresponds to the typical mining-process (or PoW step) securely realizes a functionality that models a lottery: the lottery decides which miner gets to advance the state and additionally the process of propagating this state to other miners. This lottery is captured by the state exchange functionality $\mathcal{F}_{\text{STX}}$ which allows parties to submit ledger states which are accepted with a certain probability p (that corresponds to the probability of finding a PoW). Only accepted states are then multicast to all parties and considered as possible ledger states. A detailed description of $\mathcal{F}_{\text{STX}}$ is found in Section 4.3 in [1]. The realization of $\mathcal{F}_{\text{STX}}$ by employing a proof-of-work protocol (StateExchange-Protocol in Appendix D.2 [1]) is unconditional, i.e., no restrictions on any available resource is made. In particular, the following statement holds for any number of corrupted parties and the involved simulation is black-box.

Lemma 5. *[1] The protocol StateExchange-Protocol UC-realizes functionality $\mathcal{F}_{\text{STX}}$ in the $(\mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{N-MC}})$-hybrid model.*

D Proofs of the Main Statements

D.1 Proof of Lemma 1

Lemma. *Given any adversarial strategy, there is a front-running, honest-mining adversary $\mathcal{A}$ that achieves better utility. In particular, the adversarial strategy $\mathcal{A}$ makes as many RO-queries per round as allowed by the real-world restrictions, and one environment that maximizes its utility is the environment $\mathcal{Z}$ that activates $\mathcal{A}$ as the first ITM in every round until $\mathcal{A}$ halts.*

Proof. To simplify the argument, we make use of Lemma 5 to analyze the Bitcoin protocol in a simpler hybrid world. This lemma is proven in [1] and depicted in Section C for completeness. The lemma says that instead of looking at the Bitcoin protocol, where the PoW are solved by queries to a random oracle $\mathcal{F}_{\text{RO}}$, we can equivalently look at a hybrid world where the parties and the adversary interact with a functionality $\mathcal{F}_{\text{STX}}$ which implements a lottery and is described in Section C: when submitting some new state to this functionality, with probability p it is accepted (each query by any party is treated independently). All states that are accepted by the lottery are by definition valid. Recall that the Bitcoin protocol only considers valid states as possible ledger states. Hence, a necessary condition for a block to be added to the ledger state is that the block is a valid state extension, i.e., the new state including the new block is first accepted by $\mathcal{F}_{\text{STX}}$. The modular construction step that realizes the functionality is unconditional (in the random-oracle model) and fails only with negligible probability in the security parameter κ. Looking at the simulator of this construction step given in [1] we further observe that it is a black-box simulator and that one query to the state-exchange functionality corresponds to one query to the random oracle of any real-world-adversary $\mathcal{A}$. By [10, Theorem 6], this functionality replacement does not affect the payoff except with negligible probability in κ. For simplicity, we thus still denote this hybrid world by "real world", in contrast to "ideal world" which denotes an execution of the environment, the weak ledger functionality $\mathcal{G}^{\text{B}}_{\text{WEAK-LEDGER}}$ and a simulator.

In the following we first look at the real-world UC-execution, i.e., all random variables are defined as functions on the transcript of the real UC random experiment, i.e., the execution of a protocol, the adversary, and the environment. Let X_i be the Bernoulli random variable that equals 1 if the ith query (of the transcript) to the state-exchange functionality was successful (and 0 otherwise). Let $\mathcal{A}_1$ be a front-running, honest-mining adversary making exactly q^* queries in total. By the definition of the strategy as in Definition 2, we directly see that whenever the adversary mines a block, the honest miners' protocol demands that they continue mining on the longest chain. Given that the adversary, in some round r, finds at least one block that extends the longest chain, then $\mathcal{A}_1$ always delivers the longest chain to the honest miners *immediately*. Hence, the honest miners keep on mining on the adversarial chain in this case and the adversarial blocks will eventually go into the state of the ledger as defined by the protocol. Hence, for all i with $X_i = 1$ implies that the adversary extends the state by one state block. For this adversary, we therefore get the real-world payoff function defined by

$$R^{\text{Real}}_{\mathcal{A}_1} := \texttt{breward} \cdot \texttt{CR} \cdot \sum_{i=1}^{q^*} X_i - q^* \cdot \texttt{mcost}$$

and hence

$$E\big[R^{\text{Real}}_{\mathcal{A}_1}\big] = q^* \cdot p \cdot \texttt{breward} \cdot \texttt{CR} - q^* \cdot \texttt{mcost}.$$

For sake of brevity, let us define the define the random variable $X = \sum_{i=1}^{q^*} X_i$. Note that the q^* queries are distributed among $r^* := \frac{q^*}{t}$ number of rounds. The best environment $\mathcal{Z}$ runs the environment at least r^* number of rounds until $\mathcal{A}_1$ halts, and activates the honest parties until all adversarially mined blocks appear as part of the ledger state as output by (at least) one honest party.

We now proceed with the proof of the statement. First, consider the case $(1-\delta)\cdot p\cdot\texttt{breward}\cdot\texttt{CR}-\texttt{mcost} > 0$ for some constant $0 < \delta < 1$. Let $\mathcal{A}_2$ denote any adversary making at most q queries to the state-exchange functionality, i.e., let Q denote the number of queries in an execution of the protocol, the environment $\mathcal{Z}$, and the adversary $\mathcal{A}_2$ and let P_Q be the associated distribution, then we define $q := \max \mathsf{support}(P_Q)$. The expected utility of $\mathcal{A}_2$ in this execution is loosely upper bounded by

$$E\big[R^{\text{Real}}_{\mathcal{A}_2}\big] \leq q \cdot \texttt{breward} \cdot \texttt{CR}.$$

The reason is that it is impossible to add more blocks to the ledger state (except with negligible probability). This holds by definition of $\mathcal{F}_{\text{STX}}$: one query can at most extend one state by at most one block. Consequently, if a black-box simulator $\mathcal{S}_2$ for $\mathcal{A}_2$ contradicts this observation, then it is trivially distinguishable and hence $\mathcal{S}_2 \notin \mathcal{C}_{\mathcal{A}_2}$.

For our front-running and semi-honest adversary, we can now set $q^* := \frac{q\cdot\texttt{breward}\cdot\texttt{CR}\cdot\kappa}{(1-\delta)\cdot p\cdot\texttt{breward}\cdot\texttt{CR}-\texttt{mcost}}$ and the Chernoff-Bound yields

$$\Pr[R^{\text{Real}}_{\mathcal{A}_1} < E\big[R^{\text{Real}}_{\mathcal{A}_2}\big]] \leq \Pr[R^{\text{Real}}_{\mathcal{A}_1} < q \cdot \texttt{breward} \cdot \texttt{CR}] \tag{5}$$

$$\leq \Pr[X \; < \; (1-\delta)\cdot q^* \cdot p] < \exp(\frac{-\delta^2 \cdot q^* \cdot p}{2}) \tag{6}$$

which is a negligible in κ.

Recall that the attacker's utility is actually defined on the ideal-world transcript, i.e., on the transcript of an execution of the ideal functionality, the environment, and the simulator $\mathcal{S} \in \mathcal{C}_{\mathcal{A}_1}$ that minimizes the ideal world utility defined as $U^{\langle\mathcal{F}\rangle}_{I^{\text{D}}}(\mathcal{S},\mathcal{Z}) = E(v^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}_{\text{A}})$ on the ideal-world payoff function v, where in our case $\langle\mathcal{F}\rangle := \mathcal{G}^{\text{₿}}_{\text{WEAK-LEDGER}}$. However, a general argument shows that our analysis of the real-world is actually sufficient. First, we have that $\mathcal{C}_{\mathcal{A}_1} \neq \emptyset$. To see this, consider our concrete simulator $\mathcal{S}_{weak}$ described in Lemma 4 in Section B. This simulator is black-box and UC-realizes the weak Bitcoin-ledger. Second, the number q^* of queries by $\mathcal{A}_1$ is fixed and due to black-box simulation, any simulator $\mathcal{S}$ has to answer q^* number of queries by the adversary. Third, the expected number of solutions to the proof-of-work cannot deviate by too much from the expected number in the real world. The reason is that this is observable by the environment $\mathcal{Z}$. More formally, fix an arbitrary PPT environment $\mathcal{Z}$ and let X_{ideal} be the random variable describing the number of successful returns upon querying the (simulated) state-exchange functionality. Given q submit queries, the expected number of successes in the real world is $q \cdot p$. In addition, by a Chernoff-Bound, given q submit-queries, the probability that more than $(1+\epsilon)\cdot q \cdot p$ or less than $(1-\epsilon)\cdot q\cdot p$ return with a success is negligible in the security parameter κ. Assume for the sake of contradiction that

$$E[X_{\text{ideal}} \mid Q_{ideal} = q^*] < (1-\epsilon)\cdot q^* \cdot p$$

for some $\epsilon > 0$. Since the event is observable (it corresponds to the number of state blocks of the ledger assigned to a corrupted party), we can construct an environment $\mathcal{Z}'$, which first executes $\mathcal{Z}$ (until it halts) and then outputs 1 if the deviation from the expected value of the real-world is too significant. For this environment, we would have $\text{EXEC}_{\Pi^{\text{₿}},\mathcal{A},\mathcal{Z}'} \not\approx \text{EXEC}_{\mathcal{G}^{\text{₿}}_{\text{WEAK-LEDGER}},\mathcal{S},\mathcal{Z}'}$ contradicting our assumption that $\mathcal{S}_1 \in \mathcal{C}_{\mathcal{A}_1}$. Thus, to achieve a utility exceeding $q \cdot \texttt{breward} \cdot \texttt{CR}$ with overwhelming probability, we directly get from equation 5 that

$$\Pr[v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S}_1,\mathcal{Z}_1} < q\cdot\texttt{breward}\cdot\texttt{CR}] < \exp(\frac{-\delta'^2 \cdot q^* \cdot p}{2}) = \text{negl}(\kappa), \tag{7}$$

for an appropriate constant δ' and a sufficiently large (still polynomial) number of queries q^*. Since for this choice of parameter, we have that $E[v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S}_1,\mathcal{Z}_1}] \geq q\cdot\texttt{breward}\cdot\texttt{CR} \geq E[v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S}_2,\mathcal{Z}_2}]$, we can directly conclude the following: Let

$$x_1^* := \sup_{\mathcal{Z}_1\in\texttt{ITM}}\{\inf_{\mathcal{S}_1\in\mathcal{C}_{\mathcal{A}_1}}\{E[v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S}_1,\mathcal{Z}_1}]\}\}$$
$$x_2^* := \sup_{\mathcal{Z}_2\in\texttt{ITM}}\{\inf_{\mathcal{S}_2\in\mathcal{C}_{\mathcal{A}_2}}\{E[v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S}_2,\mathcal{Z}_2}]\}\}.$$

Assume for the sake of contradiction that $x_2^* > x_1^*$. In this case, there has to exist a $\mathcal{Z}_2$ such that $\inf_{\mathcal{S}_2\in\mathcal{C}_{\mathcal{A}_2}}\{E[v_{\mathtt{A}}^{\mathcal{S}_2,\mathcal{Z}_2}]\} > \inf_{\langle\mathcal{F}\rangle,\mathcal{S}_1\in\mathcal{C}_{\mathcal{A}_1}}\{E[v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S}_1,\mathcal{Z}_1}]\}$ for all PPT environments $\mathcal{Z}_1$, which is a contradiction. Hence, for all adversarial strategies $\mathcal{A}_2$, $u_{\mathtt{A}}(\Pi^{\text{₿}},\mathcal{A}_2) \leq u_{\mathtt{A}}(\Pi^{\text{₿}},\mathcal{A}_1) + \text{negl}(\kappa)$, for some $\mathcal{A}_1 \in \mathbb{A}_{\texttt{fr}}$.

Let us now assume that $p\cdot\texttt{breward}\cdot\texttt{CR} - \texttt{mcost} \leq 0$. In this case, we will show that any adversary $\mathcal{A}$ has at most utility zero. Hence, the adversary which corrupts no party is guaranteed to be at least as good as any other strategy. Note that corrupting no party means that the adversary receives no block reward but incurs no query cost.

Let again Q be the random variable denoting the number of queries to the state-exchange functionality in a UC execution, and let the random variables X_i as defined above. We first rewrite

$$E\big[R_{\mathcal{A}}^{\text{Real}}\big] = \sum_{q=0}^{\sup_{\mathcal{Z}}\max \mathsf{support}(P_Q)} \Pr[Q=q]\cdot E\big[v \mid Q=q\big]$$

and observe that

$$\begin{aligned} E\big[R_{\mathcal{A}}^{\text{Real}} \mid Q=q\big] &= E[B \mid Q=q]\cdot\texttt{breward}\cdot\texttt{CR} - q\cdot\texttt{mcost} \\ &= \texttt{breward}\cdot\texttt{CR}\cdot\sum_{i=1}^{q} X_i \cdot - q\cdot\texttt{mcost}. \end{aligned}$$

Since the any query in the real world is an independent Bernoulli-trial with success probability p (by the definition of the state-exchange functionality), we have that the expected value gives

$$E\big[R_{\mathcal{A}}^{\text{Real}} \mid Q = q\big] = q \cdot (p \cdot \texttt{breward} \cdot \texttt{CR} - \texttt{mcost}) \leq 0.$$

This holds for any execution with any environment.

Again, for this case, we need to argue why the analysis of the real world is sufficient. This is easier to see than the first case by the following observation: the simulator $\mathcal{S}_{weak}$ described in Lemma 4 in Section B is a valid black-box simulator of any adversary $\mathcal{A}$, hence $\mathcal{C}_{\mathcal{A}} \neq \emptyset$. In particular, its simulation strategy is to perfectly mimic the real world perfectly, inserting a block into the ledger state whenever needed (since the ledger is weak, this is always possible by definition). Since this is one particular simulator of $\mathcal{C}_{\mathcal{A}}$, it constitutes an upper bound on the utility of the adversary. Since in addition, the transcript of the real and the ideal worlds are identically distributed with this simulator (i.e., $\text{EXEC}_{\Pi^{\text{₿}},\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}_{\mathcal{G}^{\text{₿}}_{\text{WEAK-LEDGER}},\mathcal{S}_{weak},\mathcal{Z}}$, the real-world analysis applies to the payoff function defined on the ideal-world transcript. We therefore get directly that for all adversarial strategies $\mathcal{A}_2$, $u_{\mathtt{A}}(\Pi^{\text{₿}}, \mathcal{A}_2) \leq u_{\mathtt{A}}(\Pi^{\text{₿}}, \mathcal{A}_1) + \text{negl}(\kappa)$, for some $\mathcal{A}_1 \in \mathbb{A}_{\mathtt{fr}}$ (the one that does not corrupt any party). □

D.2 Proof of Theorem 3

Theorem. *For $n > 0$ and $\texttt{breward} \cdot \texttt{CR} < \frac{\texttt{mcost}}{p}$ the Bitcoin protocol is not incentive compatible.*

Proof. In the following we look the real-world UC-execution in the state-exchange functionality hybrid-world. The $n > 0$ honest parties run the Bitcoin protocol $\Pi^{\text{₿}}$ in presence of an front-running, honest-mining adversary which corrupts $t < n$ parties. Each party makes one mining query per round. As the adversary is front-running any adversarial blocks mined in the current round are inserted before the honest parties fetch for new blocks. In particular, the adversary does not prevent honest parties from mining a block in the same round. Moreover, at most one honest block per round is added to the state. The probability that an honest block is added to the state in an arbitrary round is $1-(1-p)^{n-t}$. The honest parties thus insert an expected $(1-(1-p)^{n-t})$ blocks per round into the state.

Analogously to the proof of Lemma 2, the number of honest blocks inserted into the ledger state in the ideal world is negligibly close to the number of honest blocks in the real-world ledger state (which is defined as the prefix of the longest chain). Since the payoff in case of a fork is negative, the designer's utility (per round) is upper bounded by

$$u_{\mathtt{D}}(\Pi^{\text{₿}}, \mathcal{A}) \leq (1-(1-p)^{n-t}) \cdot \texttt{breward} \cdot \texttt{CR} - (n-t) \cdot \texttt{mcost}.$$

If the utility is strictly negative, i.e., $u_{\mathtt{D}}(\Pi^{\text{₿}}, \mathcal{A}) < 0$ we have that $\texttt{breward} \cdot \texttt{CR} < \frac{n-t}{1-(1-p)^{n-t}} \cdot \texttt{mcost}$. Consider the following lower bound

$$\frac{n-t}{1-(1-p)^{n-t}} \cdot \texttt{mcost} \geq \frac{(n-t) \cdot \texttt{mcost}}{1-(1-p \cdot (n-t))} = \frac{(n-t) \cdot \texttt{mcost}}{p \cdot (n-t)} \cdot \texttt{mcost} = \frac{\texttt{mcost}}{p}.$$

This implies for $\texttt{breward} \cdot \texttt{CR} < \frac{\texttt{mcost}}{p}$ (and thus $u_{\mathtt{D}}(\Pi^{\text{₿}}, \mathcal{A}) < 0$) that it is not profitable for honest parties to participate in the Bitcoin protocol. □

D.3 Proof of Theorem 4

Theorem. *Consider the real world consisting of the random oracle functionality $\mathcal{F}_{\text{RO}}$, the diffusion network $\mathcal{F}_{\text{N-MC}}$, and the clock $\mathcal{G}_{\text{CLOCK}}$, and let $\mathcal{W}_{\text{flat}}(\cdot)$ be the wrapper that formalizes the restrictions of the flat model.*[38] *Consider the class $\Pi_{\mathit{isvalidchain}_{H,d}}(\cdot)$ of protocols that are defined for the $\mathcal{W}_{\text{flat}}(\mathcal{G}_{\text{CLOCK}}, \mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{N-MC}})$-hybrid world and which are compatible with the Bitcoin network, i.e., which obey the following two restrictions:*

[38] Recall from [1] that we model restrictions by using functionality-wrappers. The above implemented restrictions correspond to the so-called flat model of Bitcoin, where each party gets one query to the random oracle per round and can send and receive one vector of messages in each round.

1. *With probability* 1*, the real-world transcript (i.e., the real-world UC-execution of Π, any environment and adversary) does not contain a chain $\mathcal{C}$ with $\mathsf{isvalidchain}_{H,\mathtt{d}}(\mathcal{C}) = 0$ and this chain was an output to the network from an uncorrupted protocol instance running Π.*
2. *Upon input* (READ, sid) *to a protocol instance, the return value is* $(\text{READ}, sid, \vec{st}^{\lceil T})$ *(for some integer T), where $\vec{st}^{\lceil T}$ denotes the prefix of the state $\vec{st}$ encoded in the longest valid chain $\mathcal{C}$ received by this protocol instance.*

With respect to the class $\sqcap_{\mathsf{isvalidchain}_{H,\mathtt{d}}(\cdot)}$, the Bitcoin protocol is an incentive-compatible choice for the protocol designer if Bitcoin is profitable as of Lemma 3, i.e., if we are in the region $\mathtt{breward} \cdot \mathtt{CR} > \frac{n \cdot \mathtt{mcost}}{p}$*, and if,*

$$\mathtt{breward} \cdot \mathtt{CR} > \frac{\mathtt{mcost}}{p \cdot (1-p)^{n-1}}. \tag{8}$$

Proof. Let us first recall some of the basic properties of our model. First, note that the protocol designer's utility decreases super-polynomially if he proposes a protocol which, even when no corruptions occur, fails to implement one consistent ledger state (i.e., the designer is punished when proposing a solution where forks could happen with non-negligible probability in $T > \mathsf{polylog}(\kappa)$), and hence such as solution cannot be incentive compatible (the loss is greater than what he could ever earn). Second, the model restricts the number of hash queries per round, which is modeled as a restriction on the number of random-oracle queries. So, each protocol instance can invoke the random oracle once in a round. Finally, the messages from the network can be fetched exactly once per round.

We present a sequence of claims and finally conclude the statement.

Claim (A). Without loss of generality, we can assume that Π only sends messages via $\mathcal{F}_{\text{N-MC}}$.

Proof: Given a protocol Π, we design another protocol Π' as follows: whenever an instance of Π with session identifier sid sends a message m to some other instance dest, we define $m' \leftarrow (\mathsf{Network}, m, \text{sid}, \text{dest}$ and send it via $\mathcal{F}_{\text{N-MC}}$ (where $\mathsf{Network}$ is a unique starting sequence of messages). Analogously, upon receiving such a message, Π' processes it as a standard incoming message just as Π would do if the destination address matches the protocol instance. Since $\mathcal{F}_{\text{N-MC}}$ is a stronger resource than the insecure network, and since the adversary is free in delivering a message to anyone of its choice, whatever ideal functionality $\mathcal{F}$ is realized by Π, it is also realized by Π' and Π' sends all messages via $\mathcal{F}_{\text{N-MC}}$. ■

Claim (B). Let $r > 0$ be some round and let Π be a protocol. In the synchronous model for protocol execution, the output to the network $\mathcal{F}_{\text{N-MC}}$ in round r of any uncorrupted protocol instance of Π is independent of the outputs of other uncorrupted protocol instances in round r if the adversarial strategy is front-running as per Definition 2.

Proof: Recall the definition of a front-running adversarial strategy. In particular, the strategy implies that the adversary delays any honest message by at least one round. Hence, a message output by an honest party (i.e., an uncorrupted protocol instance) in round r is received not before the beginning of round $r+1$ by any other honest party. This implies the claim. ■

The above claim essentially says that protocol instances cannot coordinate within one round. Note that throughout the paper, we treat the hash function H as an instance of a random oracle functionality (sampled at the onset of the execution). The following claim says that if the goal of the protocol is to generate chains $\mathcal{C}$ such that $\mathsf{isvalidchain}_{H,\mathtt{d}}(\mathcal{C}) = 1$, then, except with negligible probability, a random oracle query can produce a block whose pointer points to at most one other block, i.e., its predecessor is unique.

Claim (C). Let p_i be a protocol instance and let $r > 0$ and let $\{\mathcal{C}_i = \mathbf{G}||\mathbf{B}_{i,1}||\mathbf{B}_{i,2}||\dots||\mathbf{B}_{i,n_1}\}$ be the set of all valid chains that have been input to $\mathcal{F}_{\text{N-MC}}$ in round r. Let $\mathbf{B} = \langle \mathtt{s}, \mathtt{st}, \mathtt{n}\rangle$ be the value of p_i queried to the random oracle in round r and define the set $S_{\mathbf{B}}^{(i,r)}$ as follows:

$$S_{\mathbf{B}}^{(i,r)} := \emptyset \qquad\qquad\qquad\qquad \text{if } \mathsf{H}[\mathbf{B}] \geq \mathtt{d},$$
$$S_{\mathbf{B}}^{(i,r)} := \{\mathbf{B}_{i,j} \mid \mathtt{s} = \mathsf{H}[\mathbf{B}_{i,j}]\} \qquad \text{if } \mathsf{H}[\mathbf{B}] < \mathtt{d}.$$

Then, the probability of the event $|S_{\mathbf{B}}^{(i,r)}| > 1$ is at most negligible in κ.

In addition, except with negligible probability, $|S_{\mathbf{B}}^{(i,r)}|$ does not contain a value that was given as input to $\mathcal{F}_{\text{RO}}$ for the first time in round r by another honest party.

Proof: The event $|S_{\mathbf{B}}^{(i,r)}| > 1$ indicates that party p_i successfully found a proof of work, which is simultaneously valid for more than one chain block. Assuming that no collisions occur among output values of the random oracle functionality, the pointer $\mathtt{s}$ of block $\mathbf{B}$ uniquely defines a block $\mathbf{B}_{i,j}$ with $\mathtt{s} = \mathsf{H}[\mathbf{B}_{i,j}]$ and thus $|S_{\mathbf{B}}^{(i,r)}| \leq 1$. Thus, the probability of the event $|S_{\mathbf{B}}^{(i,r)}| > 1$ is upper bounded by the collision probability and hence is no more than $\frac{\mathsf{poly}(\kappa)}{2^\kappa}$ by the union bound. Finally, by the previous claim, the protocol actions of party p_i cannot depend on outputs of other protocol instances in round r. Hence, $|S_{\mathbf{B}}^{(i,r)}|$ does not contain any block $\mathbf{B}'$ which was input to $\mathcal{F}_{\text{RO}}$ for the first time in round r unless $\mathtt{s} = \mathsf{H}[\mathbf{B}']$. But this happens only with negligible probability since from the point of view of p_i, the return value $\mathsf{H}[\mathbf{B}']$ is independent and uniformly distributed. ■

The above claim formalizes that if two honest users simultaneously find an valid block, then those blocks cannot be part of the same chain. This will be important in the sequel.

Claim (D). Let $r > 0$. The protocol Π instructs any party p_i to extend the longest chain it receives at the onset of round r. This can only increase the utility of the protocol designer if Bitcoin the ratio between block reward and mining cost are as given in the theorem statement.

Proof: Recall that incentive compatibility requires to prove optimality of the Bitcoin protocol against a front-running adversary. By Definition 2 and Lemma 2 we can assume that (1) the adversary gets activated at the onset of each round (2) it makes all its queries (sequentially) to the random oracle, (3) it delivers all its messages in the same round, (4) it delays the honest messages by one round, and (5) it executes the honest mining protocol.

By properties (1), (2), (3), the honest miners receive the the potentially newly found blocks of the adversary already in round r, thus, for some block $\hat{\mathbf{B}}$ found by the adversary in round r, $\hat{\mathbf{B}} \in |S_{\mathbf{B}}^{(i,r)}|$ is not excluded. By property (5), the adversary was extending the longest chain known to him from round $r-1$. In addition, the adversary will continue extending the longest chain in round $r+1$ and not consider shorter ones.

By property (3),(4), and (5), the chain received by the honest parties at the onset of round r is the longest chain known to any other honest party. In case there are multiple candidates of equal length a tie-breaking rule decides. The exact rule does not matter here, as long as it leaves the probability of violating the common-prefix property negligible[39] in the parameter T.[40] Let us denote the length of the longest chain after the adversary has mined in round r by ℓ_r.

[39] Recall that for standard Bitcoin we need to choose $T > \mathsf{polylog}(\kappa)$ to make this probability negligible in κ.

[40] For standard Bitcoin, the selection rule is first come, first served and giving preference to the local longest chain of a party. Against a passive adversary the probability of creating a length-T fork using this rule is negligible in T, as it would require at least two honest miners to be simultaneously successful for an extended period of rounds (without the adversary being successful). We also observe that employing a strategy that yields a unique result, such as taking the lexicographically smallest chain among the candidates, would give probability 0 of creating a fork. This supports an observation made in [12] on the dependency of the Bitcoin network stability on a particular chain selection rule.

Now, denote by S_r^h the set of blocks mined by honest parties in round r and assume for now that $S_r^h \neq \emptyset$. By Claim C, no two blocks $\mathbf{B}_1, \mathbf{B}_2 \in S_r^h$ can be part of the same chain. Therefore, the length of the chain the front-running adversary is mining on in round $r+1$ has length at most $\ell_r + 1$ and at least ℓ_r, except with negligible probability in κ. Thus, $S_r^h \neq \emptyset$ implies that the block at position $\ell_r + 1$, belongs to one of the honest parties.

By the same argument as in the proof of Lemma 2, we arrive at an equivalence between the occurrence of a successful round of honest parties and the received payoff in the form of one block reward. By Claim C, this ratio can only be negligibly larger in the ideal world as otherwise, this would immediately be observable by an environment and would yield a distinguisher distinguishing the real world and the ideal world with an arbitrary simulator $\mathcal{S}$.

To conclude the statement, it is thus left to argue that it is optimal that each miner issues exactly one query to the random oracle. While the maximum probability of success is achieved if $n-t$ different queries are made to the random oracle, we need to argue that this yields an optimal payoff. Let Q_r^h be the random variable describing the number of honest queries to the random oracle in round r (i.e., the associated distribution of Q_r^h is a distribution on the set $\{0, \ldots, n-t\}$. We get as expected reward in one round that

$$\sum_{q=0}^{n-t} \Pr[Q_r^h = q] \cdot \underbrace{(\texttt{breward} \cdot \texttt{CR} \cdot (1-(1-p)^q) - q \cdot \texttt{mcost})}_{=:g(q)}.$$

By a straightforward calculation, we see $g(q+1) - g(q) > 0$ if $\texttt{breward} \cdot \texttt{CR} > \frac{\texttt{mcost}}{p \cdot (1-p)^q}$ $(q \in \{0, \ldots, n-t-1\})$. Hence, it is optimal for the protocol designer to choose that $P[Q_r^h = n-t] = 1$ as long as

$$\texttt{breward} \cdot \texttt{CR} > \frac{\texttt{mcost}}{p \cdot (1-p)^{n-t-1}}.$$

The largest lower bound is achieved at $t = 0$ given in the theorem statement. Note that the probability $p \cdot (1-p)^{n-t-1}$ is the probability that a fixed honest miner, e.g., the last one activated in a round, is successful and all the other honest miners are not. ■

In summary, we proved in a previous section that the Bitcoin protocol is attack-payoff secure, and in this section, we prove that the Bitcoin protocol is actually the best choice given the above condition on the value of a block reward in contrast to the mining costs. It thus holds that within this profitable region, the Bitcoin protocol is in addition incentive compatible, i.e., it constitutes an equilibrium. We hence established

$$\forall \Pi \in \mathbb{\Pi}_{\mathsf{isvalidchain}_{H,\mathrm{d}}(\cdot)} : u_{\mathtt{D}}(\Pi, \mathcal{A}_{fr}) \leq u_{\mathtt{D}}(\Pi^{\text{₿}}, \mathcal{A}_{fr}) + \mathrm{negl}(\kappa) \quad \text{and}$$
$$\forall \mathcal{A} : u_{\mathtt{A}}(\Pi^{\text{₿}}, \mathcal{A}) \leq u_{\mathtt{A}}(\Pi^{\text{₿}}, \mathcal{A}_{fr}) + \mathrm{negl}(\kappa) \text{ for some } \mathcal{A}_{fr} \in \mathbb{A}_{\mathtt{fr}},$$

as required by the definition of incentive-compatibility. This concludes the proof. □

D.4 Proof of Lemma 2

Lemma. *If* $\texttt{breward} \cdot \texttt{CR} > \frac{n \cdot \texttt{mcost}}{p}$ *then the Bitcoin protocol yields, with overwhelming probability, a positive utility for the protocol designer in the presence of front-running adversaries, i.e., the Bitcoin protocol is profitable in such a setting.*

Proof. In the following we look the real-world UC-execution in the state-exchange functionality hybrid-world. Honest parties run the Bitcoin protocol $\Pi^{\text{₿}}$ in the presence of a front-running, honest-mining adversary which corrupts $t < n$ parties. Each party makes one mining query per round, as otherwise, the synchronous computation does not proceed (and we consider all environments that maximize the $\mathcal{A}$'s utility and hence the computation has to proceed). Regarding the designer's utility, which is measured in the ideal-world, the best simulator[41] for him will yield the same distribution in payoffs and thus $u_{\mathtt{D}}(\Pi^{\text{₿}}, \mathcal{A}) = 1 - (1-p)^{n-t}) \cdot \mathtt{breward} \cdot \mathtt{CR} - (n-t) \cdot \mathtt{mcost}$ and $u_{\mathtt{D}}(\Pi^{\text{₿}}, \mathcal{A}) > 0$ we get for the interval

$$\mathtt{breward} \cdot \mathtt{CR} > \frac{n-t}{1-(1-p)^{n-t}} \cdot \mathtt{mcost}.$$

The right hand side can be upper bounded as follows:

$$\frac{n-t}{1-(1-p)^{n-t}} \cdot \mathtt{mcost} \leq \frac{n \cdot \mathtt{mcost}}{1-(1-p)^{n-t}} \leq \frac{n \cdot \mathtt{mcost}}{1-(1-p)} = \frac{n \cdot \mathtt{mcost}}{p}$$

Hence, for $\mathtt{breward} \cdot \mathtt{CR} > \frac{n \cdot \mathtt{mcost}}{p}$ that it gives positive utility to run the Bitcoin protocol. □

D.5 Proof of Theorem 5

Theorem. *Consider arbitrary environments and let the sum of the transaction fees per block be bounded by* $\mathsf{max}_{\text{block}} > 0$*. Then, the Bitcoin protocol is strongly attack-payoff secure in the attack model* $\hat{\mathcal{M}}^{\text{₿}}$*. It is further incentive compatible with respect to the class of protocols that are compatible with the Bitcoin network under the same conditions as in Theorem 4), i.e., if*

$$\mathtt{breward} \cdot \mathtt{CR} > \frac{\mathtt{mcost}}{p \cdot (1-p)^{n-1}}.$$

Proof. The proof is a direct consequence of Theorem 2 and Theorem 4. First, the best environment for the attacker is the one that always submits transactions that allow the adversary to create state blocks $\mathtt{st}$ such that $\hat{f}(\mathtt{st}) = \mathsf{max}_{\text{block}}$, which is constant and we can invoke Theorem 2 to conclude strong attack-payoff security. For the second part of the statement, it is easy to see that the best strategy for the protocol designer to let a protocol always mine blocks with the maximum number of total fees possible (i.e., available to the honest parties). By Theorem 4, this is incentive compatible as soon as the guaranteed reward is at least $\frac{\mathtt{mcost}}{p \cdot (1-p)^{n-1}}$ and Bitcoin mining is profitable (as in Theorem 4). Since $\mathtt{breward}$ is the minimum reward per block that can be guaranteed, the statement follows analogous to Theorem 4 . □

D.6 Proof of Theorem 6

Theorem. *Consider distributions* $\mathcal{D}$ *with the following property: in every round,* $\mathcal{D}$ *outputs a vector of transactions such that any party gets as input a list of transactions to build a valid next state block* $\mathtt{st}$ *to extend the longest chain and such that* $\hat{f}(\mathtt{st}) = \mathsf{max}_{\text{block}}$ *holds (where* $\mathsf{max}_{\text{block}} > 0$*). Then, with respect to* $\mathcal{D}$*-respecting environments, the Bitcoin protocol is strongly attack-payoff secure in the attack model* $\hat{\mathcal{M}}^{\text{₿}}$*. It is further incentive compatible with respect to the class of protocols that are compatible with the Bitcoin network (as defined in Theorem 4) if*

$$\mathsf{max}_{\text{block}} \cdot \mathtt{CR} > \frac{\mathtt{mcost}}{p \cdot (1-p)^{n-1}}.$$

Proof. The statement directly follows along the lines of the proof of Theorem 5 by observing that we can give an exact guaranteed block reward. Thus, we can again invoke Theorem 2 and Theorem 4 (with $\mathtt{breward} := \mathsf{max}_{\text{block}}$). Note that both theorems hold w.r.t. arbitrary environments, and hence also hold for the above type of environment (which is a subset of all environments). □

[41] Note that the best simulator is no worse than the simulator $\mathcal{S}_{weak}$ described in Lemma 4 in Section B which is a black-box simulator of any adversary $\mathcal{A}$ that mimics the real world perfectly.