A Rational Protocol Treatment of 51% Attacks

Christian Badertscher[1], Yun Lu[2], and Vassilis Zikas[3]

[1] IOHK, Zurich, Switzerland – christian.badertscher@iohk.io
[2] University of Edinburgh, Edinburgh, UK – Y.Lu-59@sms.ed.ac.uk
[3] Purdue University, West Lafayette, USA – vzikas@cs.purdue.edu

Abstract. Game-theoretic analyses of cryptocurrencies and—more generally—blockchain-based decentralized ledgers offer insight on their economic robustness and behavior when even their underpinning cryptographic assumptions fail. In this work we utilize the recently proposed blockchain adaptation of the rational protocol design (RPD) framework [EUROCRYPT '18] to analyze 51% double-spending attacks against Nakamoto-style proof-of-work based cryptocurrencies. We first observe a property of the originally proposed utility class that yields an unnatural conclusion against such attacks, and show how to devise a utility that avoids this pitfall and makes predictions that match the observable behavior—i.e., that renders attacking a dominant strategy in settings where an attack was indeed observed in reality. We then propose a generic remedy to the underlying protocol parameters that provably deter adversaries controlling a majority of the system's resources from attacks on blockchain consistency, including the 51% double-spending attack. This can be used as guidance to patch systems that have suffered such attacks, e.g., Ethereum Classic and Bitcoin Cash, and serves as a demonstration of the power of game-theoretic analyses.

1 Introduction

The classical cryptographic analysis of blockchain ledgers establishes worst-case guarantees on their security either by proving central security properties [GKL15, PSs17], such as *consistency/common-prefix*—the stable parts of the chains held by honest parties are prefixes of one-another—*liveness*—new blocks with recent transactions keep being added–or by proving that the protocol realizes an ideal ledger functionality [BMTZ17a]. Typically such analyses rely on an assumed limitation on the adversary's influence/presence in the system. In particular, the majority of an underlying resource—e.g., hashing power for proof-of-work (PoW)-based protocols such as Bitcoin [Nak08] and Ethereum [But13] (before version 2.0), or stake in Proof-of-Stake (PoS)-based protocols such as Algorand, Ouroboros, and Snow White [KRDO17, BGK+18, CM19, DPS19]—is owned/contributed by parties who honestly run the protocol.

Although such an analysis is instrumental for understanding the properties and limitations of the analyzed ledgers and gaining confidence in their security, it does not take into account a fundamental property of such systems, namely

that the ledger's state is often associated with some monetary value and therefore the protocol's security might rely on how profitable an attack might be. Thus, in addition to the classical cryptographic analysis of such systems, it is useful to analyze their so-called *economic robustness*, namely their level of protection or susceptibility to attacks by an incentive-driven (also called rational) attacker. Such an analysis can fortify the security of these systems by proving a fallback rational assumption, e.g., assuming an incentives model of the attacker, security is maintained even when certain cryptographic assumptions fail, or indicate that the proven security is fragile by pointing out natural incentives that lead to violating the security assumptions. Additionally, it can offer a higher resolution picture of the systems guarantees—e.g., its tendency to decentralize [BKKS20]—and/or more realistic estimates of the parameters associated with its security properties—e.g., relation between the density of honest blocks (that is, the chain-quality parameter [GKL15]) and the properties of the communication network [ES14, NKMS16]. Perhaps, even more interesting, it can offer insight on the system's behavior when the main (cryptographic) assumption fails, e.g., when the attacker controls a 51% fraction of the underlying resource of the blockchain protocol.

Motivated by the recent (repeated) 51% double-spending attacks that have drained millions of dollars from popular blockchain-based cryptocurrencies, we devise a game-theoretic analysis of such attacks for Nakamoto-style systems, e.g., Bitcoin, Bitcoin Cash/Gold, Ethereum (Classic), etc. We use the adaptation of the rational protocol design (RPD) framework by Garay *et al.* [GKM$^+$13] to blockchains, which was recently proposed by Badertscher *et al.* [BGM$^+$18], to analyze the utility of an attacker against these systems as a function of their basic parameters.

A central question to the relevance for practice of any game-theoretic analysis is to what extent the model and assumed utilities capture the incentives of real world attacks. Indeed, if the utilities are disconnected from reality, they can lead to counter-intuitive statements. We demonstrate an instance of such an artifact in [BGM$^+$18] and propose a different class of utilities which is both natural and avoids this artifact. We validate our utility against a range of security parameters matching those of Ethereum Classic, a PoW-based system that fell victim to 51% double-spending attacks. We observe that when the payoff for double-spending is high, attacking is indeed a dominating strategy. That is, predictions of our utility choice match reality. We then use our framework to devise a generic tuning of one of the core parameters of such blockchains—namely, the number `cutOff` of most-recent blocks needed to be dropped to achieve the so-called common-prefix property with parameter `cutOff` (cf. [BMTZ17a, BGM$^+$18, GKL15])—to deter any attacks on consistency by a rational attacker with our utility. Stated differently, we show how an incentive model can serve, possibly in addition to cryptographic assumptions, to find a robust protocol parameterization. This thereby demonstrates how our model and analysis can be used to improve the economic robustness of such blockchains, and offers a guide to how to "patch" such protocols to avoid future occurrences.

1.1 Related Literature

A number of works have focused on a rational analysis of decentralized ledgers and cryptocurrencies (e.g., [Ros11, CKWN16, ES14, Eya15, SBBR16, SSZ16, LTKS15, TJS16, NKMS16, PS17, GKW$^+$16] to mention some). Typically, these works abstract away the computational aspects of cryptographic tools (signatures, hash-functions, etc.) and provide a game which captures certain aspects of the execution that are relevant for the rational analysis. In contrast, RPD uses a cryptographic simulation-based framework to incorporate these computational considerations into the analyzed game, ensuring that predictions about attacker behavior hold for the actual protocol and not only for an idealized version (unless the idealization is obtained via a cryptographic composition argument such as UC). Incorporating such computational considerations within a rational treatment is highly non-trivial (see [GKM$^+$13, CCWrao20] for a discussion). We discuss the RPD framework in more detail in the following section.

The term *51% (double-spending) attack* is defined in [Inv] as an attack where the adversary gains any majority (not necessarily just 51%) of mining power and reverses transactions in order to double-spend its coins, often by creating a deep fork in the chain. The site CoinDesk keeps track of news of 51% attacks [Coia], of which there are quite many: most recently, Verge suffered an attack with 200 days worth of transactions erased in Feb, 2021. Also recently, Ethereum Classic suffered three 51% attacks in the same month of August, 2020, prompting a solution called MESS to mitigate such attacks which still may not provide robust security [Coib]. Other recent victims of such attacks include well-known coins such as Bitcoin Gold (Jan 2020), and Bitcoin Cash (May, 2019). A major avenue of 51% double-spending attacks is the use of rented hash power [For]. The site `https://www.crypto51.app/` gives rough estimates on the vulnerability of different coins, based on whether 51% of hashing power can be rented via a service called Nicehash. In some cases, e.g. Bitcoin Gold, it is estimated to only cost a few hundred dollars to have 51% of hashing power for 1 hour.

Previous works have considered the ability of blockchain protocols to recover from 51% attacks. In [AKWW19], conditioned on honest majority being satisfied on expectation, Bitcoin was proven to be resilient against a (temporary) dishonest majority. In [BGK$^+$20], no such condition is assumed and the authors give concrete recovery bounds as a function of the actual power of the adversary (captured as a budget to go over majority hashing power). We use the latter work for our analysis of the blockchain's security against incentive-driven attackers.

The profitability of 51% double-spending attacks have also been analyzed in previous works. The work of [Bud18] explores these attacks through an economics perspective, and leaving the cost of the attack as a parameter that is computed via simulations. The work of [JL20] computes probability of attack by modeling attacks as random walk of two independent Poisson counting processes (PCPs). In comparison, our rational analyses are done in the Rational Protocol Design (RPD) framework, where a fork is formally defined as a command in a UC ledger functionality. Another technique proposed is the Markov Decision Process (MDP) model, which is used by both [GKW$^+$16] and [HSY$^+$21]. In this model,

the adversary takes a series of actions relevant to double-spending: adopting or overriding the honest party's chain, waiting, or stopping. Solving the MDP allows these works to reason about the optimal double-spending adversary. While we do not analyze an optimal double-spending adversary, our model is more general. We do not restrict the actions of the adversary, which allows us to analyze conditions under which the protocol is secure against attacks on consistency by *any* incentive-driven adversary. Moreover, since standard MDP solvers cannot solve infinite state MDPs, the MDP is restricted to only consider situations where the chain length is less than some length c [GKW$^+$16].

1.2 Our Results

We start by devising a utility in RPD which naturally captures the incentives of an attacker to provoke a double-spending attack. To this direction, we observe that the utility considered in [BGM$^+$18] does not capture such an incentive. Intuitively, the reason is that the utility in [BGM$^+$18] essentially only considers incentives related to the consensus layer of the protocol. This means that an attacker is rewarded when successfully mining a block, but is not rewarded depending on the block contents—i.e. what kinds of transactions are in the block. Their extension to a utility function to include transaction fees does not apply to double-spending attacks. In this case, the (only) reason to attack the blockchain stems from the existence of a super-polynomial transaction fee, and assuming a moderate range of fees, no incentive to attack is present. We discuss why super-polynomial quantities are generally problematic in Section 4. It follows from [BGM$^+$18] that the attacker with these utility functions (and assuming moderate transaction fees) has no incentive to fork over mining honestly. Yet, looking at real-life double-spending attacks, this is clearly not the case. To capture double-spending, we introduce a special payoff that the attacker receives when successfully creating a deep-enough fork (i.e., orphans a sufficiently long valid chain). Intuitively, this payoff corresponds to the utility that the attacker receives when it double-spends by replacing the orphaned chain with his own.

Perhaps counter-intuitively, when analyzing Bitcoin [4] with this extended utility function, the attacker is still indifferent between forking and honest mining. We demonstrate this artifact and pinpoint the reason for it: Intuitively, the utility function from [BGM$^+$18] (with or without the extra payoff for forking) rewards the attacker by the same amount in all rounds in which it creates (mines) a block. This means that given any adversary that provokes a fork, there is always an honest-mining adversary who achieves more utility without forking by simply accumulating block rewards over a longer period of time. We distill the source of this issue in a property which we call *unbounded incentives*, and demonstrate that any utility which satisfies this property will make any deviation from passive mining a weakly *dominated* strategy.

[4] Our analysis uses Bitcoin as a representative example of Nakamoto-style blockchain ledgers, but similarly any blockchain protocol which realizes the ledger from [BMTZ17a, BGK$^+$18] could be analyzed.

We then devise a revision of this utility class which allows us to avoid the above counter-intuitive artifact. This utility, which satisfies a property we term *limited horizons*—a strong negation of unbounded incentives—has the property that the (actual) rewards of an adversary mining a block diminish with time. This is a natural way to avoid reasoning about extremely "long-lived" adversaries, i.e., that take decisions based on payoffs too far in the future, and captures features which are well-known in utility theory [Ber54]—intuitively, earning $10 today is more attractive than $1 million in 100 years, an example of the "St. Petersburg Paradox". We next turn in analyzing the profitability of 51% double-spending attacks, by showing how our revised utility can actually capture them. We provide a range of payoffs for double-spending which would incentivize an attack. Then we visualize our result using concrete parameters estimated from those of Ethereum Classic, for which performing the attack is indeed a dominant strategy. This demonstrates that the above result can explain, in a game-theoretic framework, how recent victims of 51% attacks are vulnerable.

Finally, we discuss whether and how the blockchain protocol can be tuned so that such 51% double-spending attacks are deterred. In fact, we provide a much stronger tuning, which deters attacks on consistency by any incentive-driven adversary. The tuning depends on the costs (e.g. electricity or cost to rent hashing power), positive payoffs (e.g. block rewards and payoff for causing a fork, from double-spending or otherwise), and protocol parameters (e.g. the difficulty of creating a block). Intuitively, for any combination of these parameters, we show how the window size of the underlying blockchain protocol can be adjusted so that it is not rational for the attacker to perform this attack. At the core of this results is a lemma that relates the incentive model to an attack pattern, which coupled with the self-healing properties of Nakamoto-style PoW, leads to the desired estimate of a safe parameter. We view this as a demonstration that game theory can aid us in fortifying blockchains even when assumptions made by the cryptographic analyses fail.

2 Preliminaries

2.1 The Bitcoin Backbone Protocol

The abstraction of the Bitcoin protocol that is used in the cryptographic literature is known as the *Bitcoin backbone protocol* [GKL15, PSs17, BMTZ17a] which we denote by $\Pi^{\bitcoin}$. In this abstraction, Bitcoin is modeled as a round-based protocol, where a number of participants (the miners) are connected via a multicast network with bounded delay Δ (unknown to the protocol). In every round, each party adopts the longest chain $\mathcal{C} = B_0||\ldots||B_k$ of block B_i (connected by hash-pointers) it has received so far, where B_0 is the unique genesis block of the system. Each party tries to extend this longest chain an by additional block, via running the PoW-lottery: an extension of chain $\mathcal{C}$ by a new block B_{k+1} can only be valid, if its hash $H(B_{k+1})$ belongs to a dedicated small portion of the output domain of the function (typically, the hash must have a lot of leading zeros). In

such analyses, the hash function is modeled using a random-oracle functionality $\mathcal{F}_{\mathrm{RO}}$ that returns uniform values upon each query. Therefore, when extending the chain, each party makes a certain number of *mining queries* per round (that is, RO-queries with candidate blocks B_{k+1} containing a random nonce to obtain the hash) and we call a mining query *successful*, if the output is below the threshold. In the setting with fixed PoW difficulty, we can assign a success probability p to each such mining query. Finally, if a miner is successful, it will send the new chain over the multicast network to all other miners.

Cryptographic security. The main security guarantee[5] proven for the Bitcoin protocol is eventual *consistency*: every block that is deep enough can be considered immutable and only the most recent, $\mathtt{cutOff}$ number of blocks might be transient. This $\mathtt{cutOff}$-consistency (where the cutoff parameter is often left implicit if clear from context) guarantee states that at any point in time, the prefix of $\mathcal{C}$ consisting of $|\mathcal{C}| - \mathtt{cutOff}$ blocks is common to all honest miners:

Definition 1 (Consistency). *Let $\mathcal{C}_1 \preccurlyeq \mathcal{C}_2$ denote the prefix-of relation, then the consistency guarantee (with parameter $\mathtt{cutOff}$) states that at any two points in time $a \leq b$ in an execution, where party P at round a holds chain $\mathcal{C}_1$ and party P' at round b holds chain $\mathcal{C}_2$, we have that $\mathcal{C}_1|_{\mathtt{cutOff}} \preccurlyeq \mathcal{C}_2$, where the notation $\mathcal{C}|_k$ denotes the prefix of $\mathcal{C}$ obtained by removing the most recent k blocks (and if k exceeds the length of $\mathcal{C}$, it is defined to correspond to the genesis block).*

In the cryptographic setting (without incentives), such a guarantee only holds if we restrict the adversary to have a minority of mining power. That is, given $n_a^{(r)}$ and $n_h^{(r)}$ denote the numbers of adversarial and honest mining queries in round r, respectively, then the protocol $\Pi^{\mathbb{B}}$ is secure if in any round r the inequality $n_a^{(r)} < \theta_{pow} \cdot n_h^{(r)}$ holds, with $\theta_{pow} := (1-p)^{(2\Delta+1)\mathtt{T}_{\mathrm{ub}}}$ being the well-established security threshold for Bitcoin (often stated in its linear approximation $1 - 2(\Delta+1)p\mathtt{T}_{\mathrm{ub}}$) [GKL15, PSs17, BMTZ17a], where the quantity $\mathtt{T}_{\mathrm{ub}}$ denotes the upper bound on the number of mining queries per round. Throughout this work, we work in the so-called *flat model* of Bitcoin for notational simplicity [GKL15, BGM$^+$18], where each miner gets one mining query per round (and the adversary's power is the number of corrupted miners). We note that sometimes it is convenient to assume a lower bound $\mathtt{T}_{\mathrm{lb}}$ on the number of mining queries (a.k.a. participation) per round, in particular when arguing about the guaranteed growth of the blockchain over time in combination with the security threshold. Finally, we point out that even if there are no adversarial players, an upper bound $\mathtt{T}_{\mathrm{ub}}$ on the number of queries is necessary for security in the fixed difficulty setting, when aiming for a common prefix guarantee for some target parameter $\mathtt{cutOff}$. As the failure probability of Bitcoin becomes negligible as a function of $\mathtt{cutOff}$ (more precisely, the relevant factor is of the order $2^{-\Omega(\mathtt{cutOff})}$), we often treat it as a (of course polynomial-bounded) function $\mathtt{cutOff}(\kappa)$ of a

[5] While other security guarantees exist, such as *chain quality*, our focus in this paper is consistency.

security parameter κ, and (in symbolic notation) $\texttt{cutOff} = \omega(\log(\kappa))$ is at least required to obtain a negligible probability of a failure.

Bitcoin backbone and UC. The RPD framework is based on the UC framework. As such, the above Bitcoin backbone protocol $\Pi^{\text{₿}}$ is seen as a UC protocol as in [BMTZ17a], where it is proven to UC-realize a strong transaction ledger functionality $\mathcal{G}_{\text{LEDGER}}$ under the honest majority assumption. We give here just the explanation of how the ideal consistency guarantee looks like: the functionality $\mathcal{G}_{\text{LEDGER}}$ ensures that at any point in time, there is only one unique ledger state (sequences of transactions packed in blocks), where the state is append-only (that is, whatever appears as a block in the state is immutable). Furthermore, different honest parties see different prefixes of this state, with the guarantee that these *views* are increasing and within a window of `windowSize` (a ledger parameter) blocks from the tip of the state. Note that the cut-off parameter of Bitcoin corresponds exactly to the size of that window in the realized ledger $\mathcal{G}_{\text{LEDGER}}$. More precisely, whenever Bitcoin satisfies Definition 1, then the above mentioned correspondence holds and the ledger state is a single chain of blocks [BMTZ17a].

In UC, the protocol $\Pi^{\text{₿}}$ assumes a couple of hybrid functionalities. First, the round-based structure is achieved using UC-synchronous tools (assuming a clock functionality), a network, and a random oracle, where restrictions on the mining queries can be captured by functionality wrappers restricting the number of RO evaluations, e.g. [BMTZ17a, GKO$^+$20]. One extremely helpful aspect of UC in the context of RPD is the compatibility with the composition theorem [GKM$^+$13]. In this work this is leveraged as follows. The Bitcoin backbone $\Pi^{\text{₿}}$ admits a modular structure that isolates the lottery aspect as a submodule of the system. Technically, the proofs in [BMTZ17a, PSs17] show that whenever the PoW-lottery UC-realizes the *state exchange* functionality $\mathcal{F}_{\text{StX}}$ (in [PSs17] the related concept is called $\mathcal{F}_{\text{tree}}$), the Nakamoto-style longest chain rule protocol (under the above honest-majority security threshold) realizes the ledger. This intermediate step is important due to two things: first, it models an idealized mining process where each mining query is an independent Bernoulli trial with success probability p (and hence abstracts away those real-life negligible probability events that would destroy independence), and second it abstracts away the low-level details of the chain structure (where e.g., "hash collisions" could cause disruptions). It is proven in [BMTZ17a] that the proof-of-work layer of Bitcoin (in the random oracle model) UC-realizes $\mathcal{F}_{\text{StX}}$. Moreover, since it only abstracts the lottery part of the system, this realization does not depend on any security threshold. We can therefore leverage composition when analyzing the utilities of Bitcoin and work with the idealized lottery directly.

2.2 Rational Protocol Design

The Rational Protocol Design framework (RPD) allows us to analyze the security of the blockchain without assuming honest majority. Although consistency and other security properties are lost if an attacker can arbitrarily break honest

majority, assuming attackers are *rational* offers an alternate method of limiting his actions. That is, although the attacker is free to act in any way (e.g. corrupt more than majority hashing power), he will only do so if it is profitable. Building on [BGM+18], our analysis is based on the Rational Protocol Design (RPD) framework introduced in [GKM+13]. RPD analyzes the security of protocols, such as Bitcoin, with respect to an incentive-driven adversary. In this model, a protocol designer $\mathtt{D}$ plays an *attack game* $\mathcal{G}$ with an attacker $\mathtt{A}$. First, the designer $\mathtt{D}$ comes up with a protocol Π. Then, the attacker $\mathtt{A}$—who is informed about Π—comes up with an adversarial strategy $\mathcal{A}$ to attack Π. The utility of the attacker (resp. designer) is then defined on the *strategy profile* $(\Pi, \mathcal{A})$, and is denoted $u_{\mathtt{A}}(\Pi, \mathcal{A})$ (resp. $u_{\mathtt{D}}(\Pi, \mathcal{A})$). In this work, we focus on the attacker's utility $u_{\mathtt{A}}(\Pi, \mathcal{A})$.

The game $\mathcal{G}$ is defined with respect to an attack model $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$. $\mathcal{F}$ is the functionality which the designer would like to implement such as a ledger that provides certain ideal guarantees as described above. However, when certain assumptions, e.g. honest majority for Bitcoin, are not met (which as stated above we explicitly do not want to demand *a priori*), we cannot hope to get $\mathcal{F}$. Instead, the designer $\mathtt{D}$'s protocol Π (in our case, the Bitcoin protocol $\Pi^{\dot{\mathbb{B}}}$) only implements a weaker functionality. This weaker functionality that Bitcoin implements when lifting the honest majority assumption is proven to be $\mathcal{G}^{\dot{\mathbb{B}}}_{\text{WEAK-LEDGER}}$ in [BGM+18] and provided in App. C.2 for completeness. Intuitively, the weak ledger is derived from the stronger version [BMTZ17a] by introducing a few weaknesses. For example, it allows the adversary to fork the ledger state and hence allows it to break consistency (this event corresponds to a deep reorganization of the blockchain in the real world). This is allowed by the FORK command in $\mathcal{G}^{\dot{\mathbb{B}}}_{\text{WEAK-LEDGER}}$. Given the views of the simulator and environment in an ideal world execution, the value functions $v_{\mathtt{A}}$ and $v_{\mathtt{D}}$ assign payoffs to the attacker and designer respectively, when certain events happen in the views, such as when the simulator forks the blockchain via $\mathcal{G}^{\dot{\mathbb{B}}}_{\text{WEAK-LEDGER}}$. Finally, utilities $u_{\mathtt{A}}$ and $u_{\mathtt{D}}$ are functions of payoffs (defined with $v_{\mathtt{A}}$ and $v_{\mathtt{D}}$) of simulators that can simulate $\mathcal{A}$ in Π in the environment $\mathcal{Z}$. Looking ahead, the goal of RPD is to find conditions under which a rational attacker would not invoke the weaknesses of $\mathcal{G}^{\dot{\mathbb{B}}}_{\text{WEAK-LEDGER}}$ (e.g., it is too costly to perform an attack). For example, if under a class of utilities, no rational attacker invokes the FORK command, then we essentially obtain a stronger ledger (i.e., the same except that this command is absent and hence the ledger state remains a unique chain) against attackers incentivized by this class of utilities.

2.3 Utility of the Attacker from [BGM+18]

We detail the attacker's utility in [BGM+18], which in the RPD framework captures the expected payoff of a particular adversarial strategy $\mathcal{A}$ in a given protocol Π (in our case $\Pi = \Pi^{\dot{\mathbb{B}}}$). This payoff is calculated based on different *events* that occur in the real execution and the corresponding ideal experiment where a black-box simulator is attempting to simulate this adversarial strategy.

Specifically, the work of [BGM+18] considers the following events:

1. Event $W^{\mathtt{A}}_{q,r}$, for each pair $(q, r) \in \mathbb{N}^2$: The simulator simulates q mining queries by the adversary in round r of the simulated execution.
2. Event $I^{\mathtt{A}}_{b,r}$, for each pair $(b, r) \in \mathbb{N}^2$: 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. Informally, this event occurs when an honest party views these blocks as "confirmed" (part of his own ledger state).

A different payoff is associated with each event. In order to make q mining queries and invoke event $W^{\mathtt{A}}_{q,r}$, the attacker must pay $q \cdot \mathtt{mcost}$, where $\mathtt{mcost}$ is the cost of making a mining query (e.g. electricity cost per hash query). When b blocks made by the adversary are inserted into the ledger and event $I^{\mathtt{A}}_{b,r}$ occurs, the attacker receives payoff $b \cdot \mathtt{breward} \cdot \mathtt{CR}$. Here $\mathtt{breward}$ is the reward for making a block in the currency of the blockchain (e.g. Bitcoins), and $\mathtt{CR}$ is an exchange rate to the same currency used for $\mathtt{mcost}$ (e.g. USD).

Then, [BGM+18] defines the following attacker's utility for a strategy profile $(\Pi, \mathcal{A})$. Let $\mathcal{C}_{\mathcal{A}}$ denote the set of simulators that can emulate an adversary $\mathcal{A}$ in the ideal world with access to the weaker ledger functionality $\mathcal{G}^{\check{\mathtt{B}}}_{\text{WEAK-LEDGER}}$, and $\mathcal{Z}$ denote an environment. The *real payoff* of an adversary $\mathcal{A}$ attacking the protocol is defined as the minimum payoff over all simulators in $\mathcal{C}_{\mathcal{A}}$. If $\mathcal{C}_{\mathcal{A}} = \emptyset$ (there are no simulators that can simulate $\mathcal{A}$) then $u_{\mathtt{A}}(\Pi, \mathcal{A}) = \infty$ by definition. Then, the utility $u_{\mathtt{A}}(\Pi, \mathcal{A})$ is the real payoff, maximized over all possible environments $\mathcal{Z}$ (we assume for simplicity that environments are closed and run in polynomial time in the security parameter [Can01]).

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) := \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} \left(b \cdot \mathtt{breward} \cdot \mathtt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}] \right) \right. \right. \tag{1}$$
$$\left. \left. - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}] \right\} \right\}.$$

The work of [GKM+13] introduces the following notion of security against incentive-driven adversaries: No matter the utility achieved by an adversary $\mathcal{A}$ running the protocol Π in the real world, there exists an adversary $\mathcal{A}'$ running the dummy protocol with access to the ideal functionality $\mathcal{F}$ that achieves the same or better utility. In other words, even the best adversary attacking Π, cannot achieve better utility than one who does not invoke any of the "bad events" in $\langle \mathcal{F} \rangle$. Note that here $\mathcal{F}$ can be any strengthening of its weaker version. For example, the weak ledger without the option to break consistency would be a strengthening of $\mathcal{G}^{\check{\mathtt{B}}}_{\text{WEAK-LEDGER}}$ in which case attack-payoff security implies that there is no incentive (even for a majority-controlling adversary) to create a fork (that is, a deep reorganization) even though he technically could be able to.

Strictly speaking, the utilities are also functions in the security parameter κ (the environment obtains the parameter as input in UC) but we omit it for notational simplicity. We note that as functions in the security parameter κ, the asymptotic behavior of the involved functions is the relevant aspect.

Definition 2 (Attack payoff security [GKM+13]). *Let* $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$ *be an attack model inducing utility* $u_{\mathtt{A}}$*, 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* $\mathcal{A}$*, there is an* $\mathcal{A}'$ *such that* $u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u_{\mathtt{A}}(\Phi^{\mathcal{F}}, \mathcal{A}') + \mathsf{negl}(\kappa)$

This notion of attack-payoff security does not necessarily mean an incentive-driven adversary will honestly follow the protocol—there is no restriction on the honestly of the actions of $\mathcal{A}'$ in the above definition. To capture this stronger requirement in the context of Bitcoin, we also consider a stronger notion introduced by [BGM+18]: the attacker is incentivized to always choose a *front-running, passive-mining* adversary over any (potentially malicious) strategy. Informally, this passive adversary behaves exactly like an honest party (mining with all his hashing power and releasing a block he has found immediately), except the adversary's messages are always delivered before the honest parties' (front-running). Front-running gives the adversary an advantage since if an adversary's block is concurrently competing with an honest party's block to be appended to the longest chain, the adversary always wins.

Definition 3 (Front-running, passive-mining adversary [BGM+18]). *The front-running adversarial strategy* $\mathcal{A} \in \mathbb{A}_{fr}$ *is specified as follows: Upon activation in round* $r > 0$*,* $\mathcal{A}$ *activates in a round-robin fashion all its (passively) corrupted parties, say* $p_1, \ldots, p_t$*. When corrupt party* p_i *generates some new message to be sent through the network,* $\mathcal{A}$ *immediately delivers it to all its recipients. In addition, upon any activation, any message submitted to the network* $\mathcal{F}_{\text{N-MC}}$ *by an honest party is maximally delayed.*

$\Pi^{\mathbb{B}}$ was proved to be strongly attack-payoff in [BGM+18] for the utility in Equation 1. Informally, a protocol is strongly attack-payoff secure if there is always a passive adversarial strategy that is at least as good as any malicious strategy. In this work, we are also interested in the case where security does not hold: we say an adversary $\mathcal{A}$ *breaks* strong attack-payoff security if $u_{\mathtt{A}}(\Pi, \mathcal{A})$ exceeds $u_{\mathtt{A}}(\Pi, \mathcal{A}')$ for any $\mathcal{A}' \in \mathbb{A}_{\mathtt{fr}}$, by a non-negligible amount.

Definition 4 (Strongly attack-payoff secure [BGM+18]). *A protocol* Π *is* strongly attack-payoff secure *for attack model* $\mathcal{M}$ *if there is a* $\mathcal{A}' \in \mathbb{A}_{fr}$ *such that for all* $\mathcal{A}$*,* $u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u_{\mathtt{A}}(\Pi, \mathcal{A}') + \mathsf{negl}(\kappa)$

In our work, we will follow the approach from [BGM+18] that simplifies the proofs when analyzing the utilities from mining in the protocol $\Pi^{\mathbb{B}}$ by utilizing the composition theorem of RPD. As explained above, instead of analyzing the probabilities of payoff-inducing events for $\Pi^{\mathbb{B}}$ which uses the random oracle as the lottery, one can analyze probabilities for the *modular* ledger protocol w.r.t. an idealized lottery that makes use of the state exchange functionality $\mathcal{F}_{\text{StX}}$ (Figure C.1, App. C.1). In more detail: when a party (or the adversary in the name of a corrupted party) wishes to extend a chain, they would invoke $\mathcal{F}_{\text{StX}}$ with a SUBMIT-NEW command, which performs a coin toss and informs him whether he is successful. If the party is successful, the functionality includes this

new chain into a tree data structure and allows the party to multicast this new chain with a SEND command; this multicasting is done automatically for honest parties. Due to the correspondence of RO queries in the Bitcoin protocol and the SUBMIT-NEW-commands in the modularized Bitcoin protocol [BMTZ17a], the events defined for $u_{\mathtt{A}}^{\mathbb{B}}(\Pi, \mathcal{A})$ (for the full Bitcoin protocol) above remain valid and meaningful also in this hybrid world, because the black-box simulator for the overall Bitcoin protocol simulates one RO-query (as a reaction to an input by a corrupted party) whenever the (black-box) simulator for the modular ledger protocol simulates one SUBMIT-NEW-command,as a reaction to the corresponding input by the same party [BGM+18].

3 Artifacts of Unbounded Incentives

In this section, we discuss an artifact of the utility function Equation 1, which we will eliminate in the next section. Concretely, we prove that this RPD utility is inappropriate to capture the most realistic situation of attackers that attack the system, e.g., attempt a fork to profit from double-spending. To do so, we prove Lemma 1 and 2, which roughly show this surprising fact: if running the protocol (semi-)honestly is profitable in expectation, then there is no incentive for an adversary to fork. The intuitive reason for this is clear: Any fixed payoff for forking incurred by the adversary can be offset by an adversary who runs slightly longer (and still polynomially long) but does not fork. This, however, is an artifact of the asymptotic definition and does not reflect real-world incentive-driven attack scenarios, where mining is anticipated to be profitable—otherwise no one would mine—but attackers still perform forking attacks (in particular, in order to double-spend coins). We distill a property of the utility from [BGM+18] that is the reason this artifact, which we call *unbounded incentives*, and prove that any utility satisfying this property will suffer from the same artifact. Looking ahead to the following section, we will propose a natural adaptation of this utility function that does not suffer from the above artifact (and where in particular the duration of an attack actually starts to matter).

3.1 Demonstrating the Artifact

Let us first consider the straightforward adaptation of the utility from Equation 1 to model the payoff (e.g. double-spending) an adversary gains by forking the ledger. Define the event K as: There is a round r where the simulator uses the FORK command of the weak ledger functionality $\mathcal{G}_{\text{WEAK-LEDGER}}^{\mathbb{B}}$ (App. C.2) that allows the simulator to invoke a fork. Let `fpayoff` be the payoff for invoking the fork. Then, the utility $u_{\mathtt{f}}$ becomes:

$$u_{\mathtt{f}}(\Pi,\mathcal{A}) := \sup_{\mathcal{Z}\in\mathtt{ITM}}\left\{\inf_{\mathcal{S}^{\mathcal{A}}\in\mathcal{C}_{\mathcal{A}}}\left\{\sum_{(b,r)\in\mathbb{N}^2} b\cdot\mathtt{breward}\cdot\mathtt{CR}\cdot\Pr[I^{\mathtt{A}}_{b,r}]\right.\right.$$
$$\left.\left. - \sum_{(q,r)\in\mathbb{N}^2} q\cdot\mathtt{mcost}\cdot\Pr[W^{\mathtt{A}}_{q,r}] + \mathtt{fpayoff}\cdot\Pr[K]\right\}\right\}. \quad (2)$$

Below, we show that for the the utility function $u_{\mathtt{f}}$ above, the Bitcoin protocol $\Pi^{\ddot{\mathcal{B}}}$ is strongly attack-payoff secure as long as mining is profitable. Our proof takes advantage of the artifact of unbounded incentives: informally, first we show that the payoff of any polynomial-run-time adversary $\mathcal{A}$ is bounded by a polynomial $p(\kappa)$ of the security parameter; then, we show that there is a passive, front-running adversary whose run-time is also polynomial (albeit bigger than that of $\mathcal{A}$), and who achieves at least $p(\kappa)$ utility.[6]

Lemma 1 (Attack payoff security with forking). *Let $\mathcal{T}_{\mathrm{ub}} > 0$ be the upper bound on total number of mining queries per round, $p \in (0,1)$ be the probability of success of each mining query, and $\mathtt{cutOff} = \omega(\log(\kappa))$ be the consistency parameter. Let $\mathcal{M}$ be a model whose induced utility $u_{\mathtt{f}}$ has parameters $\mathtt{fpayoff}, \mathtt{breward}, \mathtt{CR}, \mathtt{mcost} \geq 0$. The Bitcoin protocol $\Pi^{\ddot{\mathcal{B}}}$ is strongly attack-payoff secure in $\mathcal{M}$ if $p\cdot\mathtt{breward}\cdot\mathtt{CR} - \mathtt{mcost} > 0$.*

3.2 A First Attempt to Eliminate the Artifact

Although we proved that Bitcoin is strongly attack payoff secure even with a payoff for forking, this is actually not a good sign, as this result does not reflect reality. In reality, attackers do fork blockchains to gain profit via e.g. double-spending transactions. Thus, the fact that we can prove Lemma 1 means that there must be a problem with our assumptions.

Why were we able to prove Lemma 1? It turns out the utility function we used has the weakness that it considers an attacker who does not care about the ephemeral payoff for forking—he can simply obtain more utility via block rewards if he just put in a bit more hashing power for mining. Thus, somewhat counter-intuitively, to model incentives for forking attacks, we must consider utilities that limit the amount of mining an attacker can do.

A first natural instinct may be to incorporate in the utility the (often substantial) initial investment (e.g. cost of buying mining rigs) an attacker must make before being able to participate in the blockchain protocol. This turns out to be not only a natural extension, but also a very simple one. Concretely, we capture this investment as *cost of party corruption*: in order to use party for

[6] We note that for the simple utility function presented in [BGM+18] other proof techniques could conclude attack-payoff security without the runtime-extension argument. The main point here is to demonstrate the importance of considering the attack duration in the utility function.

mining, the adversary needs to corrupt him, which corresponds to acquiring its mining equipment. Formally, for each $g \in \mathbb{N}$ define $C_g^{\mathtt{A}}$ as follows: The maximum number of corrupted parties at any round is g. Let $\mathtt{ccost}(g)$ be the cost of event $C_g^{\mathtt{A}}$, i.e. corrupting g parties. Then we define the utility function:

$$u_{\mathtt{f},\mathtt{c}}(\Pi, \mathcal{A}) := \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \mathtt{breward} \cdot \mathtt{CR} \cdot \Pr[I_{b,r}^{\mathtt{A}}] \right. \right. \\ - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W_{q,r}^{\mathtt{A}}] \\ + \mathtt{fpayoff} \cdot \Pr[K] \\ \left. \left. - \sum_{g \in \mathbb{N}} \mathtt{ccost}(g) \cdot \Pr[C_g^{\mathtt{A}}] \right\} \right\}. \quad (3)$$

Interestingly, as we see below, this natural extension is still insufficient to align the model with the reality that forking attacks occur. Indeed, even with this additional cost, we can still prove a result similar Lemma 1. Concretely, the following lemma shows that for $u_{\mathtt{f},\mathtt{c}}$ above, we can prove the statement as the one in Lemma 1 about $\Pi^{\not\Diamond}$ being attack-payoff secure by again exploiting the artifact of unbounded incentives.

Lemma 2 (Attack payoff security with forking, with cost of corruption). *Let $T_{\mathrm{ub}} > 0$ be the upper bound on total number of mining queries per round, $p \in (0,1)$ be the probability of success of each mining query, and $\mathtt{cutOff} = \omega(\log(\kappa))$ be the consistency parameter. Let $\mathcal{M}$ be the model whose induced utility $u_{\mathtt{f},\mathtt{c}}$ has parameters $\mathtt{fpayoff}, \mathtt{breward}, \mathtt{CR}, \mathtt{mcost} \geq 0$, $\mathtt{ccost}(\cdot) : \mathbb{N} \to \mathbb{R}^+$. The Bitcoin protocol is strongly attack-payoff secure in $\mathcal{M}$ if $p \cdot \mathtt{breward} \cdot \mathtt{CR} - \mathtt{mcost} > 0$.*

3.3 The Source of the Artifact: Unbounded Incentives

Distilling the issue in above lemmas, we observe that that as long as the adversary keeps accumulating rewards as rounds are added to the protocol—i.e., mining remains profitable—he does not care about the payoff for forking: there always exists a polynomial-time, passively mining strategy that simply gains the same amount of utility by mining a bit more. However, not only do real-life attackers in fact profit from forks, even the assumption on the profitability of mining forever is unrealistic: any attacker is at least limited in time by e.g. the anticipated age of the universe, and cannot, in practice, keep accumulating utility in perpetuity.

Thus, to make accurate prediction about the attackability of a blockchain protocol the utility function must exclude the eternal profitability of passive mining. We generalize this intuition, by defining the notion of *unbounded incentives*: a utility function has *unbounded incentives* if there is an adversarial strategy $\mathcal{A} \in \mathbb{A}_{\mathtt{fr}}$ such that for any polynomial $h(\kappa)$, $\mathcal{A}$ can gain better payoff

than $h(\kappa)$. (Conversely, we will say that a utility has bounded incentives if there there is no such passive adversary.).

It is straightforward to verify that the utilities we have seen so far have unbounded incentives, which explains the effect of the artifact exploited in the above lemmas. In fact, in the following there is a simple argument for a generic statement about the strong attack-payoff security of utility functions that have unbounded incentives.

Lemma 3. *Let $\mathcal{M}$ be a model inducing a utility function $u_{\mathtt{A}}$. Assume for any adversary $\mathcal{A}$, in any real execution of the protocol his payoff is polynomially-bounded.* [7] *If $u_{\mathtt{A}}$ has unbounded incentives for a protocol Π, then Π is strongly attack-payoff secure for $\mathcal{M}$.*

Proof. Suppose $u_{\mathtt{A}}$ has unbounded incentives. Then let $\mathcal{A} \in \mathbb{A}_{\mathtt{fr}}$ be the adversary in the definition of unbounded incentives. Then $\mathcal{A}$ is a witness for strong attack-payoff security: For every real execution of another adversary $\mathcal{A}'$, the passive adversary $\mathcal{A}$ can gain better payoff than $\mathcal{A}'$. □

4 An RPD Analysis of Forks

In this section, we will tune our utility function to avoid the issue of *unbounded incentives* isolated in the previous section. A straw man approach would be to make `fpayoff` a super-polynomial function of the security parameter. But this would imply a very unnatural assumption, which, intuitively, corresponds to ensuring that the polynomially-bounded adversaries are *always* incentivized to fork. This would have the opposite effect and introduce a different artifact: it would make attack-payoff security impossible, and making a 51% attack always a dominant strategy no matter the systems parameters, contradicting the observable fact that many blockchains have not fallen to 51% attacks.

Instead, we make `breward` a function of time, which captures e.g., inflation, or simply that the adversary only plans to stay in the system for a limited amount of time. We refer to this adaptation of $u_{\mathtt{f,c}}$ as $u_{\mathtt{buy}}$:

$$u_{\mathtt{buy}}(\Pi, \mathcal{A}) := \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \begin{aligned} &\sum_{(b,r)\in\mathbb{N}^2} b \cdot \mathtt{breward}(r) \cdot \mathtt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}] \\ &\quad - \sum_{(q,r)\in\mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}] \\ &\quad + \mathtt{fpayoff} \cdot \Pr[K] \\ &\quad - \sum_{g\in\mathbb{N}} \mathtt{ccost}(g) \cdot \Pr[C^{\mathtt{A}}_{g}] \end{aligned} \right\} \right\}. \tag{4}$$

[7] This is true for the utility function $u^{\hat{\mathtt{B}}}_{\mathtt{A}}$ in Equation 1 (as well as the utility functions we will consider)—no adversary can get payoff that is superpolynomial in the run time of the execution.

We also define a version of this utility $u_{\texttt{rent}}$, formally defined in Appendix A.4, which models the attacker renting hashing queries by replacing $\texttt{mcost}$ with parameter $\texttt{rcost}$ (rent cost) and setting $\texttt{ccost}(\cdot) = 0$. Renting especially has been observed in real attacks, such as the August 2020 attacks on Ethereum Classic [For].

Note that while $\texttt{breward}$ is a function of time, we let the cost of a mining query, that is $\texttt{mcost}/\texttt{rcost}$, remain constant. We do so to model the attacker's anticipated monetary budget to launch and maintain an attack, such as the costs for renting a certain amount of hashing power (which are generally paid upfront), or cost of electricity (which realistically appears to be relatively stable). Further, the parameter $\texttt{fpayoff}$ should be seen as an abstract excess payoff for the attacker arising from forking that is able to capture various use-cases. In the prototypical (double-spend) example where the attacker sells some coins for fiat currency and later tries to regain the coins with a successful attack, it corresponds to this extra fiat inflow gained prior to attacking the blockchain. We note that the utility functions could be tweaked to allow for all parameters to be time-dependent without changing the results qualitatively as long as the relations among the parameters required by the definitions and theorems (which are time-dependent in our treatment already) still hold.

To capture realistic utilities, we restrict to instances of our utility function which satisfy what we call *limited horizons* (Definition 5). Roughly, limited horizons constrains utilities by requiring that passive mining eventually becomes unprofitable. Recall that in light of the St. Petersburg Paradox discussed in the introduction, rational parties become increasingly reluctant to invest some monetary budget for potential rewards gained only later in a randomized process (e.g. due to uncertainty about the future or other specific utility-relevant considerations like relative inflation between several quantities). We cast this general idea as a rather simple condition based on our utility function.

After defining limited horizons, in Section 4.1, we will first address a technical challenge imposed when payoff-parameters in the utility functions are non-constant. Then, in Section 4.2 we show that limited horizons implies bounded incentives (i.e., the opposite of unbounded incentives) through Lemma 5. More precisely, limited horizon is a strong negation[8] of unbounded incentives. Looking ahead, we will prove that when utilities have limited horizons, there is always a large enough payoff for forking such that (strong) attack-payoff security is broken. Informally, a utility function $u_{\texttt{buy}}$ (resp. $u_{\texttt{rent}}$) has limited horizons if there is a time limit after which passive mining becomes unprofitable.

Definition 5 (Limited Horizons). *We say $u_{\texttt{buy}}$ in Equation 4 (resp. $u_{\texttt{rent}}$, formally defined in Equation 5), parameterized by $\texttt{breward}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$, $\texttt{mcost}, \texttt{fpayoff} \geq 0$, and non-decreasing function $\texttt{ccost}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ (resp. $\texttt{breward}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$, $\texttt{rcost}, \texttt{fpayoff} \geq 0$) satisfies* limited horizons *(resp.* limited horizons with renting*) if $\texttt{breward}(\cdot)$ is a non-increasing function such that $\exists x \in \mathbb{N} : \; p \cdot \texttt{CR} \cdot \texttt{breward}(x) < \texttt{mcost}$.*

[8] Note that the *strong negation* of an assertion A is one which implies $\neg A$, but is not necessarily implied by $\neg A$.

Remark. Technically, $u_{\texttt{rent}}$ is a special case of the case of $u_{\texttt{buy}}$ (since the utilities are the same if we set $\texttt{mcost} = \texttt{rcost}$ and set $\texttt{ccost}(\cdot) = 0$); however semantically they are different: $\texttt{rcost}$ represents the cost of renting a hashing query, which usually is much higher than $\texttt{mcost}$ which represents the cost (e.g. electricity) of an adversary mining with his own equipment. Nevertheless, to reduce redundancies in the technical sections, we will analyze the utility $u_{\texttt{buy}}$ in Equation 4 (with a general $\texttt{ccost}(\cdot)$, including when $\texttt{ccost}(\cdot) = 0$), and state the results for the renting case as corollaries.

4.1 Addressing Technical Issue of Non-Constant Payoff for Block Rewards

In this section, we address a technical issue with considering a non-constant $\texttt{breward}$—recall that in limited horizons, $\texttt{breward}$ is a non-increasing function of time/round number. By our definition (which follows that of [BGM+18]), the event $I^{\mathtt{A}}_{b,r}$ happens when b blocks are placed into the ledger of some honest party. This is intuitive—the block reward should be given only when the block is "confirmed" to be in the ledger. However, there is a delay between when a block is broadcasted, and when it makes it into the common prefix of an honest chain. This delay is a random variable which depends on the amount of (honest and corrupt) hashing power in the protocol, the network delay, and the adversary's strategy. Fortunately, we can lower and upper bound such a delay (which we denote by t_{lb}, t_{ub} respectively), as we show in the following lemma. This will in turn allow us to avoid the complication of analyzing when blocks enter the ledger state and instead analyze when locks broadcasted by the adversary to honest parties (whose events are easier to analyze). Note that we choose to analyze time-of-block-broadcast, instead of time-of-block-creation, since the adversary may choose to withhold successfully-mined blocks instead of broadcasting them immediately, making time-of-broadcast more suitable for incorporating such adversarial strategies.

We first define a useful quantity $t^{\Delta}_{\delta}(q)$. As we will see, this quantity, which is derived from the *chain growth* property of Nakamoto-style blockchains, is the maximum time for honest chains to grow by $\texttt{cutOff}$ blocks, given that in each round there are at least q honest mining queries.

Definition 6 (Maximum time to grow $\texttt{cutOff}$ blocks). *For network delay Δ, and $p, \delta \in (0,1)$, we denote $t^{\Delta}_{\delta}(q) := \frac{\texttt{cutOff}}{(1-\delta)\gamma}$, where $\gamma := \frac{h}{1+h\Delta}$ and $h := 1-(1-p)^q$.*

Let $t_{lb} := 0$ and $t_{ub} := t^{\Delta}_{\delta}(\mathtt{T}_{\mathrm{ub}})$. Let $B^{\mathtt{A}}_{b,r}$ denote the event: At round r, the adversary broadcasts b blocks made by parties that are corrupted at the time of the blocks' creation, and which are part of the longest chain at round r. Let $u^{h}_{\texttt{buy}}$ be $u_{\texttt{buy}}$ except $\sum_{(b,r)\in\mathbb{N}^2} b \cdot \texttt{breward}(r) \cdot \texttt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}]$ (which considers time of block confirmation) is replaced with $\sum_{(b,r)\in\mathbb{N}^2} b \cdot \texttt{breward}(r+t_{lb}) \cdot \texttt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,r}]$ $= \sum_{(b,r)\in\mathbb{N}^2} b \cdot \texttt{breward}(r) \cdot \texttt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,r}]$ (which considers time of block broadcast).

Similarly, let $u^{l}_{\mathtt{buy}}$ replace the same term in $u_{\mathtt{buy}}$ with $\sum_{(b,r)\in\mathbb{N}^2} b \cdot \mathtt{breward}(r + t_{ub}) \cdot \mathtt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,r}]$. We define them formally in App. A.5.

The following lemma tells us that instead of analyzing the utility function defined on when a block is confirmed in the ledger we can instead approximate by only analyzing when a block is broadcasted. This will be helpful in our proof of Lemma 5 on the utility of the optimal front-running, passive adversary.

Lemma 4 (Translating time-of-block-confirmation to time-of-block-broadcast: $u^{h}_{\mathtt{buy}}$ and $u^{l}_{\mathtt{buy}}$). *For any utility function satisfying limited horizons (in fact, we only require that* $\mathtt{breward}(\cdot)$ *is a non-increasing function), satisfies the following: For all adversaries $\mathcal{A}$, and front-running, passive $\mathcal{A}'$,*

$$u_{\mathtt{buy}}(\Pi^{\not\Bbbk}, \mathcal{A}) \leq u^{h}_{\mathtt{buy}}(\Pi^{\not\Bbbk}, \mathcal{A}) + \mathsf{negl}(\kappa) \quad \textit{and}$$
$$u_{\mathtt{buy}}(\Pi^{\not\Bbbk}, \mathcal{A}') + \mathsf{negl}(\kappa) \geq u^{l}_{\mathtt{buy}}(\Pi^{\not\Bbbk}, \mathcal{A}').$$

Proof. The first inequality is obvious: By limited horizons, giving block rewards using time-of-block-broadcast (i.e., $u^{h}_{\mathtt{buy}}$) gives the attacker a higher payoff.

The second inequality: Let the environment be one which maintains $\mathtt{T}_{\mathrm{ub}}$ parties in each round after r. The bound follows then from the chain-growth lower bound which states the minimum chain length increase during a time period, depending on the honest parties' hashing power and the network delay (cf. [BMTZ17b, PSs17]). This concludes the proof. □

4.2 Optimal Utility of Front-running, Passive Adversaries

We show in this section if a utility satisfies limited horizons, then it also satisfies bounded incentives. We do so by proving the following optimal utility of a passive, front-running adversary. We define $u^{\mathrm{h}}_{\mathrm{honest}}$ and $u^{\mathrm{l}}_{\mathrm{honest}}$ which, as we will see in Lemma 5 below, are the upper and lower bounds on the optimal utility obtained by a front running, passive adversary in $\Pi^{\not\Bbbk}$.

Definition 7 (Bounds $u^{\mathbf{h}}_{\mathbf{honest}}$ and $u^{\mathbf{l}}_{\mathbf{honest}}$ for optimal front-running, passive adversary). *We define the quantity*

$$u^{\mathrm{h}}_{\mathrm{honest}}(\mathtt{breward}, \mathtt{CR}, \mathtt{mcost}, \mathtt{ccost}) := g \cdot p \cdot \mathtt{CR} \cdot \sum_{x=1}^{t} [\mathtt{breward}(x + t_{lb}) - \mathtt{mcost}] - \mathtt{ccost}(g)$$

with

$$t := \arg\max_{x\in\mathbb{N}} (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{lb}) \geq \mathtt{mcost}),$$
$$g := \arg\max_{g\in[0,T_{\mathrm{ub}}]} \left(mg - \mathtt{ccost}(g)\right),$$
$$\textit{for } m := \sum_{x=1}^{t} \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{lb}) - \mathtt{mcost}\right),$$

and the quantity

$$u^{\mathrm{l}}_{\mathrm{honest}}(\mathtt{breward}, \mathtt{CR}, \mathtt{mcost}, \mathtt{ccost}) := g \cdot p \cdot \mathtt{CR} \cdot \sum_{x=1}^{t} [\mathtt{breward}(x + t_{ub}) - \mathtt{mcost}] - \mathtt{ccost}(g)$$

with

$$t := \arg\max_{x \in \mathbb{N}} (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{ub}) \geq \mathtt{mcost}),$$
$$g := \arg\max_{g \in [0, T_{\mathrm{ub}}]} \left(mg - \mathtt{ccost}(g)\right),$$
$$\textit{for } m := \sum_{x=1}^{t} \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{ub}) - \mathtt{mcost}\right).$$

We simplify the above upper and lower bounds on the optimal front-running, passive adversaries as $u^{\mathrm{h}}_{\mathrm{honest}}$ and $u^{\mathrm{l}}_{\mathrm{honest}}$, when the parameters to the utility function are clear from context. As discussed before, although we prove the optimal passive adversary for $u_{\mathtt{buy}}$, the renting case for utility $u_{\mathtt{rent}}$ is a direct corollary by setting $\mathtt{ccost}(\cdot) = 0$ and $\mathtt{mcost} = \mathtt{rcost}$.

Intuitively, the following lemma is established by proving that (1) due to limited horizons, there is a fixed time t after which an optimal passive adversary will not mine, and (2) it is optimal for a passive adversary to corrupt parties statically. Then, we can re-write the utility of a front-running, passive adversary as a function of his running time t, and the number of parties he corrupts g. Optimizing for t and g gives us the optimal utility of this passive adversary.

Lemma 5 (Optimal utility of a front-running passive adversary, for incentives with limited horizons). *Let $T_{\mathrm{ub}} > 0$ be the upper bound on total number of mining queries per round, $p \in (0, 1)$ be the probability of success of each mining query, and $\mathtt{cutOff} = \omega(\log(\kappa))$ be the consistency parameter. Given parameters such that $u_{\mathtt{buy}}$ satisfies limited horizons and protocol $\Pi^{\mathbb{B}}$, for $\mathcal{A}$ the optimal adversary in $\mathbb{A}_{fr}$, $u_{\mathtt{buy}}(\Pi^{\mathbb{B}}, \mathcal{A}) \leq u^{\mathrm{h}}_{\mathrm{honest}} + \mathsf{negl}(\kappa)$ and $u_{\mathtt{buy}}(\Pi^{\mathbb{B}}, \mathcal{A}) + \mathsf{negl}(\kappa) \geq u^{\mathrm{l}}_{\mathrm{honest}}$*

This lemma directly implies that any utility with limited horizons also has bounded incentives.

Proof. We show the proof for the upper bound. The lower bound can be proven in the exact same way by constructing an optimal front-running, passive adversary for $u^{l}_{\mathtt{buy}}$, except replacing $\mathtt{breward}'(x)$ with $\mathtt{breward}''(x) := \mathtt{breward}(x + t_{ub})$.

Let $\mathtt{breward}'(x) = \mathtt{breward}(x + t_{lb})$. By Lemma 4, we can prove the upper bound on utility in the lemma by constructing an optimal front-running, passive adversary $\mathcal{A}$ for the utility function $u^{h}_{\mathtt{buy}}$.

We first show that, at any round, the optimal passive adversary $\mathcal{A}$ for $u^{h}_{\mathtt{buy}}$ is the one who either mines with all his corrupted parties, or does not mine at all.

Claim (1). There is a round t where it is optimal for a passive adversary to do the following: For all rounds $x \leq t$, the adversary does not de-register any party. That is, he mines with all parties corrupted at round t. At round $t+1$, the adversary de-registers all his corrupted parties and stops execution.

Proof of Claim (1): Let g_x be the number of corrupted parties at round x in a given protocol execution. Suppose the adversary mines (queries the random oracle with blocks) with $g \leq g_x$ corrupted parties. Then the payoff for mining at round x is (up to negligible difference)

$$\begin{aligned}\sum_{b\in\mathbb{N}} b \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,x}] - \sum_{q\in\mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,x}] \\ = g \cdot p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - g \cdot \mathtt{mcost} \\ = g \cdot (p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - \mathtt{mcost})\,.\end{aligned}$$

Since we assume $\mathtt{cutOff} = \omega(\log(\kappa))$ and the adversary is passive, the probability and thus payoff for a fork is negligible. The first equality holds since when the adversary is front-running and passive, all successfully-mined blocks will be added to the ledger. Thus, both $B^{\mathtt{A}}_{b,x}$ and $W^{\mathtt{A}}_{q,x}$ only depend on the number of queries made at round x.

We see that if $(p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - \mathtt{mcost}) \geq 0$ he gains the optimal utility by mining with all his rigs. On the other hand, when $(p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - \mathtt{mcost}) < 0$, then he obtains optimal utility by not mining at all. Thus, he does not lose utility by de-registering his parties. There exists a round t described in the claim by assumption of the utility function parameters satisfying bounded mining incentives. ■

Our second claim says that the optimal adversary is the one who statically corrupts parties at the first round. That is, adaptive corruption does not increase his payoffs.

Claim (2). Let G be the total number of parties corrupted by an adversary in a protocol execution and P_G be the associated distribution. Then, (1) given any front-running, passive adversary where $g = \max Supp(G)$, an environment $\mathcal{Z}$ which spawns $g+1$ parties gives the optimal payoff. Moreover, (2) given an environment which spawns at least $g+1$ parties at the first round, the optimal front-running, passive adversary who corrupts at most a total of g parties, is to statically corrupt them at the first round.

Proof of Claim (2): The first statement (1) is given by the fact that the payoff of the passive adversary only depends on the number of successful blocks inserted into the ledger. Since the adversary is front-running, it does not matter how many honest parties there are, as long as the environment spawns enough parties to allow for at least one honest party (for the technical reason that the ledger state is accepted by some honest party). The second statement follows from the fact that Claim (1) implies the adversary's mining strategy at any round does not depend on the number of corrupt parties nor view of the protocol execution. Thus, an adversary corrupting a party at some round x always gains equal or

better payoff by corrupting the party at the first round. It is possible for the adversary to corrupt statically first round since we assumed the environment spawns enough parties. ■

Finally, we show that the optimal front-running, passive adversary $\mathcal{A}$ for $u^h_{\mathtt{buy}}$ will corrupt a deterministic number of parties at the first round. By Claims 1 and 2, it suffices to analyze the utility of a front-running, passive adversary $\mathcal{A}$ who corrupts some g parties statically at the first round and mines for t rounds.

$$
\begin{aligned}
&u_{\mathtt{buy}}(\Pi^{\mathbb{B}}, \mathcal{A}) \leq u^h_{\mathtt{buy}}(\Pi^{\mathbb{B}}, \mathcal{A}) \\
&= \sum_{b\in\mathbb{N}} \sum_{x\geq 1} \left(b \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,x}]\right) - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= \mathtt{CR} \sum_{b\in\mathbb{N}} \left(b \cdot \Pr[B^{\mathtt{A}}_{b,1}] \cdot \sum_{x=1}^{t} \mathtt{breward}'(x)\right) - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= \mathtt{CR} \left(\sum_{x=1}^{t} \mathtt{breward}'(x)\right) \left(\sum_{b\in\mathbb{N}} b \cdot \Pr[B^{\mathtt{A}}_{b,1}]\right) - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= \mathtt{CR} \left(\sum_{x=1}^{t} \mathtt{breward}'(x)\right) \cdot g \cdot p - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= g \cdot p \cdot \mathtt{CR} \cdot \sum_{x=1}^{t} \left(\mathtt{breward}'(x) - \mathtt{mcost}\right) - \mathtt{ccost}(g) + \mathsf{negl}(\kappa)
\end{aligned}
$$

To go from line 2 to line 3: for $x \in [1, t]$, $\Pr[B^{\mathtt{A}}_{b,x}] = \Pr[B^{\mathtt{A}}_{b,x'}] = \Pr[B^{\mathtt{A}}_{b,1}]$, and for $x > t$, $\Pr[B^{\mathtt{A}}_{b,x}] = 0$. To go from line 4 to line 5: we see $\sum_{b\in\mathbb{N}} b \cdot \Pr[B^{\mathtt{A}}_{b,1}]$ is the expected number of blocks made in each round, which is gp. We next find the g, t that maximize the above expression. We see that from Claim 1, and that $u_{\mathtt{buy}}$ satisfies limited horizons:

$$
\begin{aligned}
t &= \arg\max_{t\in\mathbb{N}} \left(\sum_{x=1}^{t} \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}'(x) - \mathtt{mcost}\right)\right) \\
&= \arg\max_{x\geq 1} \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}'(x) \geq \mathtt{mcost}\right)
\end{aligned}
$$

Let m be the mining profit defined by

$$
m = \sum_{x=1}^{t} \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}'(x) - \mathtt{mcost}\right)
$$

Then the optimal number of corruptions g is the following (recall $\mathtt{T}_{\mathrm{ub}}$ is the maximum hashing power per round).

$$g = \arg \max_{g \in [0, \mathtt{T}_{\mathrm{ub}}], g \in \mathbb{N}} (mg - \mathtt{ccost}(g))$$

□

5 Analyzing 51% Attacks

We can now utilize our above framework to analyze one of the most common types of forking attacks, known as 51% *double-spending* attack [Inv]. We analyze a range of parameters for utility functions with limited horizons, for which a 51% double-spending adversary breaks the strong attack-payoff security of protocol $\Pi^{\mathbb{B}}$ (formalized by Theorem 1). In more detail, first we will show a general lemma relating the number of honest/adversarial hashes per round, to the time it takes to fork with a 51% double-spending attack (Lemma 6). Then, in Theorem 1 we will show that if the payoff for a successful attack (`fpayoff`) satisfies certain conditions, then an adversary performing a 51% double-spending attack achieves better utility than any passive-mining strategy. This `fpayoff` is quantified as a function of the parameters of the protocol and the utility function.

We call the following strategy a *51% double-spending attack*: The adversary obtains any majority fraction ("51%" is just a colloquial name) of the hashing power, and uses it to secretly mine an extension of the currently longest chain (i.e., keeping successful blocks private to himself), and which he will release after some time. We say that a 51% double-spending attack is *successful* if, when released, the adversary's secret chain is at least as long as the honest chain, and causes the ledger state of some honest party to fork (which in reality corresponds to a roll-back of more than `cutOff` blocks, in order to adopt the released attack chain). If this happens, some transactions on the reversed blockchain ledger state may become orphaned (no longer part of the ledger state), thus allowing the attacker to double-spend his coins.

5.1 Time to Fork

We start by showing a general lemma that relates the amount of honest and adversarial hashing power in a system, to the time to cause a fork via a 51% double-spending attack. That is, how long it takes for an adversary with majority hashing power to secretly create a chain that, when released, would cause an honest party to roll back, or discard, more than `cutOff` blocks of his own chain in order to adopt the new one.

Definition 8. *We say that an adversary $\mathcal{A}$* causes a fork *in a protocol Π if, except with negligible probability in κ, all simulators $\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}$ (i.e. those which in fact simulate $\mathcal{A}$ according to UC emulation) use the* FORK *command*[9].

[9] If $\mathcal{C}_{\mathcal{A}} = \emptyset$, then in any case by definition the utility of $\mathcal{A}$ is infinite.

The FORK command, which allows forking the confirmed ledger state (and hence corresponds to rolling back more than `cutOff` blocks in the real world world), is necessary and sufficient to simulate an adversary who succeeds in a 51% double-spending attack. We compute the (upper bound) time for a 51% double-spending adversary to fork, which is obtained by the time for honest parties to grow their chain by `cutOff` blocks (for which we can use guaranteed chain-growth of Nakamoto-style blockchains. Since the adversary has more hashing power (and thus more random oracle queries that can be issued sequentially) than the honest party, and since we assume $\texttt{cutOff} = \omega(\log(\kappa))$ and that the adversary does not interfere with the honest parties' mining, this implies that the adversary's secretly-mined chain will be longer than the honest parties' chain, and be the only source for a large rollback, with overwhelming probability in κ.

Lemma 6 (Time to Fork with 51% Attack). *Let* $\texttt{cutOff} = \omega(\log(\kappa))$, $[r, r+t]$ *be any time interval (starting from some round* $r \geq 0$*) of* $t \geq 1$ *rounds,* $\Delta \geq 1$ *be the network delay,* $p \in (0,1)$ *the success probability of one mining query.*

Then for all $\delta, \delta' \in (0,1)$, $\alpha \geq \frac{1+\delta}{1-\delta}$, *and* $q \geq 1$ *such that* $t \geq t_{\delta'}^{\Delta}(q)$ *(Definition 6) the following holds. Suppose in time interval* $[r, r+t]$*, (1) the honest parties make at least* q *mining queries per round, and (2) in total they make at most* q_t *queries. Then, the adversary* $\mathcal{A}$ *who performs a 51% double-spending attack for at least* αq_t *queries during the time interval and then releases his secretly-mined chain, causes a fork in the protocol* $\Pi^{\mathcal{B}}$*.* [10]

Proof. As usual we analyze in the state exchange functionality ($\mathcal{F}_{\text{STX}}$) hybrid-world, which ignores negligible-probability bad event like hash collisions that break the tree structure of chains (and in any case can only serve to help the adversary as the honest parties do not take advantage of such bad events).

Let the adversary's strategy be the following: He does nothing before some round r. Let $\mathcal{C}$ be the longest known chain at the beginning of round r (since the adversary knows all the chains, he knows the longest chain). During $[r, r+t]$, the adversary mines secretly (i.e., mines without releasing any successful blocks), starting from $\mathcal{C}$ (for αq_t queries). After mining in round $r+t$, the adversary releases his secretly-mined chain.

Let c_a be the random variable (r.v.) that is the number of blocks made by the adversary's secret mining, and c_h the r.v. that is the length of the longest chain fragment made by honest parties, during $[r, r+t]$.

First we claim that the adversary causes a fork if, except with negligible probability in κ, the following conditions hold:

(1) $c_a \geq c_h$ and (2) $c_h > \texttt{cutOff}$.

[10] More concretely, he succeeds except with probability at most $\exp\left(-\frac{\delta^2 \alpha \mu}{2+\delta}\right) + \exp\left(-\frac{\delta^2 \mu}{2}\right) + \exp\left(-\frac{\delta'^2 t\gamma}{2+\delta}\right)$.

By (1) and the fact $\mathcal{C}$ is the longest chain at round r, the adversary's secret chain is at least as long as the longest honest chain, so his chain will be adopted by honest parties when he releases it. In addition, by (2), the honest parties must roll back more than $\texttt{cutOff}$ blocks to adopt the adversary's secret chain after it is released, which breaks consistency. In this case, any simulator successfully simulating $\mathcal{A}$ must use the command FORK.

Let b_h and b_a be the total number of blocks made by the honest and corrupt parties during the time interval. Then $c_h \leq b_h$ and $E(c_h) \leq E(b_h) = \mu = q_t p$. Moreover $c_a = b_a$ since the adversary does mine his own blocks sequentially, and $E(c_a) = E(b_a) = \alpha\mu$.

We recall the Chain Growth Lemma [BMTZ17a, GKL15, PSs17] (formally defined in App A.3) that relates the time it takes an honest chain to grow by some T blocks with the honest parties' hashing power and the network delay. We are in particular interested in $T = \texttt{cutOff}$—that is, how long it takes for the honest chain to grow by $\texttt{cutOff}$ blocks. This lemma tells us that if $t \geq \frac{\texttt{cutOff}}{(1-\delta)\gamma}$ (implied by our assumption) then $c_h \geq \texttt{cutOff}$ except with probability at most $\mathsf{negl}(\gamma t)$. Since $\texttt{cutOff} = \omega(\log(\kappa))$, we have $\mathsf{negl}(\gamma t) = \mathsf{negl}(\kappa)$.

We analyze below the probability that either condition (1) or (2) above fail. To get from line 1 to line 2: By $\alpha \geq \frac{1+\delta}{1-\delta}$, we have $\alpha q p_t(1-\delta) \geq q_t p(1+\delta)$; moreover $c_h \leq b_h$. To get line 3 to line 4: We apply Chernoff bound and Chain Growth Lemma.

$$
\begin{aligned}
&\Pr\left[(c_a < c_h) \vee (c_h \leq \texttt{cutOff})\right] \\
&\leq \Pr\left[(c_a \leq \alpha q_t p(1-\delta)) \vee (b_h \geq q_t p(1+\delta)) \vee (c_h \leq \texttt{cutOff})\right] \\
&\leq \Pr\left[c_a \leq \alpha q_t p(1-\delta)\right] + \Pr\left[b_h \geq q_t p(1+\delta)\right] + \Pr\left[c_h \leq \texttt{cutOff}\right] \\
&\leq \exp\left(-\frac{\delta^2 \alpha\mu}{2+\delta}\right) + \exp\left(-\frac{\delta^2\mu}{2}\right) + \exp\left(-\frac{\delta'^2 t\gamma}{2+\delta}\right)
\end{aligned}
$$

where $\mu = q_t p$. Since we assume $\mu = q_t p \geq w = \omega(\log(\kappa))$, $\exp(-c\mu) = \exp(-\omega(\log(\kappa)))$ is negligible in κ for any constant c.[11] □

A visualization. In Figure 1, the (upper-bound) time to fork with exactly 51% corruption, is graphed against the total number of rigs in the system. The graph uses the formula from Lemma 6. We use current parameters for Ethereum Classic as the source of the concrete parameters for this figure, and refer the reader to App. B for more details.

5.2 Payoff of 51% Double-Spending Attacks

In this section, we prove Theorem 1 and its corollary Theorem 2, which quantify the size of the payoff for double-spending, under which a 51% double-spending attack can break strong attack-payoff security. That is, the attacker achieves

[11] To prove that it is negligible, for contradiction suppose there is a polynomial x^d that is asymptotically larger than e^g for some $g = \omega(\log(\kappa))$. But $x^d = e^{d\log(x)}$ which implies $g = O(\log(x))$ which is a contradiction.

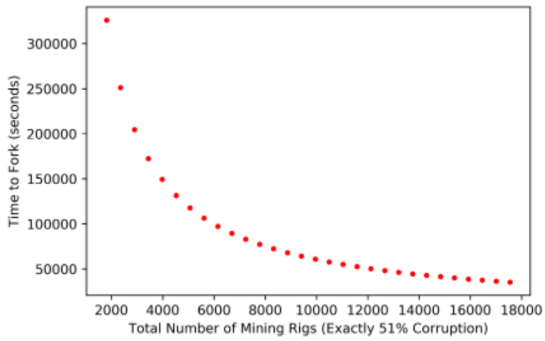

Fig. 1. Time to create a fork via 51% attack, versus the total number of mining rigs. Here the adversary corrupts exactly 51%.

better utility than *any* passive mining strategy. While one may think that it is always profitable to attack if there is no assumption on the honest majority of hashing power, there are a few things that may deter an attacker. For example, the costs of buying or renting mining equipment for the attack may become too high compared to the diminished block rewards as time goes on. Our statement below quantifies an amount of payoff for forking (e.g. how much an attacker can double-spend) to incentivize a 51% double-spending attack. Intuitively, the result below says that as long as the payoff for forking ($\texttt{fpayoff}$) is larger than the loss of utility from withholding blocks and corrupting a large number of parties to perform the attack, then there is a 51% attack strategy that is more profitable than *any* front-running, passive adversary.

Theorem 1 (51% Double-Spending Attacks that Break (Strong) Attack-Payoff Security ($u_{\texttt{buy}}$)). *Let $T_{\text{ub}} > 2$ be the upper bound on total number of mining queries per round, $p \in (0,1)$ be the probability of success of each mining query, and $\texttt{cutOff} = \omega(\log(\kappa))$ be the consistency parameter. Then, the protocol $\Pi^{\mathring{\mathrm{B}}}$ is not attack-payoff secure/strongly attack-payoff secure in any attack model $\mathcal{M}$ whose induced utility function $u_{\texttt{buy}}$ satisfies limited horizons, if for some $\delta \in (0,1)$, $\alpha > 1$ and $g = \frac{T_{\text{ub}}}{1+\alpha}$ the following holds:*

$$\texttt{fpayoff} > u^{\text{h}}_{\text{honest}} - \alpha \cdot g \cdot t^{\Delta}_{\delta}(g)\left(p \cdot \texttt{CR} \cdot \texttt{breward}(t^{\Delta}_{\delta}(g) + t_{ub}) - \texttt{mcost}\right) - \texttt{ccost}(\alpha g).$$

Proof. First we show an upper bound on the utility of an optimal passive adversary $\mathcal{A}_1$. Then, we show that there is a 51% attacking adversary $\mathcal{A}_2$ who achieves better utility than $\mathcal{A}_1$.

We show the former, by showing the optimal payoff of any pair $(\mathcal{Z}, \mathcal{A}_1)$, where $\mathcal{Z}$ is the environment and $\mathcal{A}_1$ a front-running and passive adversary in $\Pi^{\mathring{\mathrm{B}}}$. This is directly from Lemma 5, which shows that $u_{\texttt{buy}}(\Pi^{\mathring{\mathrm{B}}}, \mathcal{A}_1) \leq u^{\text{h}}_{\text{honest}} + \mathsf{negl}(\kappa)$ for any utility function $u_{\texttt{buy}}$ that satisfies limited horizons.

Then, we show that there is an environment under which an (malicious) adversary $\mathcal{A}_2$ obtains non-negligibly more utility than $\mathcal{A}_1$. Consider the adversary that corrupts $\alpha g = \frac{\alpha \mathtt{T}_{\mathrm{ub}}}{1+\alpha}$ parties, and mines a secret chain for $t_\delta^\Delta(g)$ rounds (so he will make $\alpha \cdot g \cdot t_\delta^\Delta(g)$ mining queries) before releasing his secret chain and then deregisters his corrupted parties. By Lemma 6, he successfully causes a fork with overwhelming probability in κ. However, by withholding blocks (in order to create a fork), he also loses some block rewards. This is because his blocks would end up in an honest party's ledger at a later time than if he just broadcasted successful blocks immediately after creating them. Consider an environment which spawns $\mathtt{T}_{\mathrm{ub}}$ parties (which are honest as the adversary does not corrupt them) after he releases his secret chain, which will reduce the time between when the adversary broadcasts his secret chain and when the chain becomes confirmed as part of the ledger to at most t_{ub} (Lem. 4). This means the adversary $\mathcal{A}_2$ achieves utility of:

$$u_{\mathtt{buy}}(\Pi^{\not\Bbbk}, \mathcal{A}_2) \geq \alpha \cdot g \cdot t_\delta^\Delta(g) \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}(t_\delta^\Delta(g) + t_{ub}) - \mathtt{mcost}\right) - \mathtt{ccost}(\alpha g) + \mathtt{fpayoff}.$$

We ignore negligible probability/utility loss from an unsuccessful attack. We also recall that t_{ub} is the upper bound from Lemma 4 on the time between a block is created, and the block enters a honest party's ledger (we also recall the utility is computed using the optimal environment for the adversary). Since $\mathtt{breward}(\cdot)$ is a non-increasing function, this lower-bounds the payoff the adversary obtains from creating blocks. Given the conditions on $\mathtt{fpayoff}$ it then holds that $\mathcal{A}_2$ breaks strong attack-payoff security. Because the attack provokes the fork-command, also attack-payoff security cannot hold. □

We state the case where the adversary mines with rented equipment (and uses utility function $u_{\mathtt{rent}}$), as a direct corollary to Theorem 1.

Theorem 2 (51% Double-Spending Attacks that Break (Strong) Attack-Payoff Security ($u_{\mathtt{rent}}$)). *Let $\mathtt{T}_{\mathrm{ub}} > 2$ be the upper bound on total number of mining queries per round, $p \in (0,1)$ be the probability of success of each mining query, and $\mathtt{cutOff} = \omega(\log(\kappa))$ be the consistency parameter. Then, the protocol $\Pi^{\not\Bbbk}$ is not attack-payoff secure/strongly attack-payoff secure in any attack model $\mathcal{M}$ whose induced utility function $u_{\mathtt{rent}}$ satisfies limited horizons, if for any $\delta \in (0,1)$, $\alpha > 1$ and $g = \frac{\mathtt{T}_{\mathrm{ub}}}{1+\alpha}$ the following holds:*

$$\mathtt{fpayoff} > u_{\mathrm{honest}}^{\mathrm{h}} - \alpha \cdot g \cdot t_\delta^\Delta(g) \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}(t_\delta^\Delta(g) + t_{ub}) - \mathtt{mcost}\right).$$

5.3 Visualizations with Concrete Values

We will visualize Theorems 1 and 2 through Figures 2 and 3. We consider two utility functions, one where the adversary buys mining equipment, and one where

the adversary rents. We then graph the utilities of passive/non-passive adversaries, against the maximum fraction of corrupted parties. The concrete parameters are based on current (as of writing, Feb. 2021) parameters for Ethereum Classic. The outline is given in Appendix B.

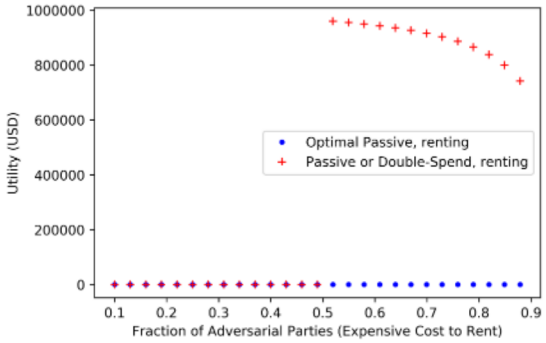

Fig. 2. Utility of the passive/51% double-spending attacker who rents hashing power, versus the fraction of adversarial parties. Here we consider an expensive cost to rent hashing power (1.96 BTC/TH/day, at $\$50,000$/BTC).

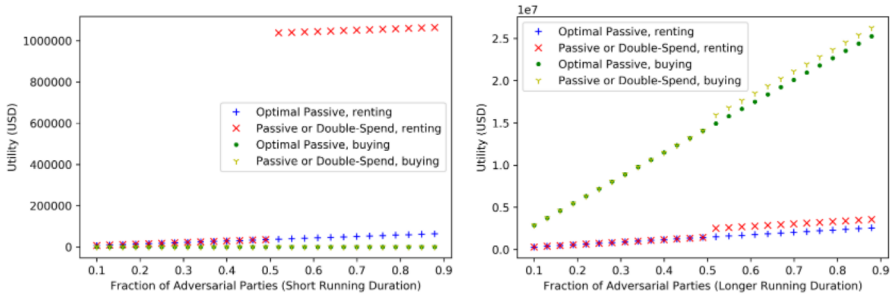

Fig. 3. Utility of the passive/51% double-spending attacker versus the fraction of adversarial parties. We consider an attacker who runs for a short duration (1 week) and a long duration (40 weeks).

In Figure 2, we consider the incentives of a 51% attacker who rents his hashing power, using the price for renting of 1.96 BTC/TH/day (Bitcoin per terahash per day), at $\$50,000$/BTC. In this case, it is in fact not profitable to mine passively (and thus the optimal passive strategy is to not mine at all). However, when the adversary corrupts more than majority of hashing power, it may become

profitable to mine in order to create a fork. It is less profitable for the adversary to corrupt a larger fraction of the parties, as cost of renting becomes too high. We remark that even when it is not profitable to mine (passively) using rented rigs, this does not exclude incentivizing honest parties from mining with e.g., bought equipment.

In the next two examples in Figure 3, we compare the utility of the attacker who mines with purchased rigs, and one who mines with rented rigs. For the attack who buys hashing power, each rig costs $3000, and mining (electricity) costs $0.000047/s. For the attacker who rents, for more interesting comparisons we consider a cheaper cost to rent hashing power (1.96 BTC/TH/day, at a cheaper $\$22,000$/BTC). We consider two scenarios: the attacker either (1) only plans to mine for a short duration of one week, or (2) plans to mine for a longer duration of 40 weeks (time is expressed in seconds in the code). For the purposes of the graphs, we account for the possible variance of Bitcoin-USD exchange rates by using an average exchange rate over the relevant period of time. In either case, to more closely model reality, we restrict the duration of the attack, where the adversary may obtain a majority of hashing power, to 3 days (which, in the code, simply means we do not show attacks that last longer than 3 days). We see a big difference between the two scenarios. In the short duration case, it is much more profitable to mine or attack with rented rigs. In fact, it is not even profitable to fork using purchased rigs, as the cost of purchase is higher than the payoff for double-spending. The long duration case is the opposite. Although it may be profitable to mine in both cases, it is vastly more profitable to mine and attack with purchased rigs than rented rigs. This agrees with our intuition and reality: the high initial investment of buying mining equipment is offset in the long run by the lower cost of mining. Moreover, an attacker who is only interested in mining in order to perform a 51% attack for a short time is incentivized to use hash renting services.

6 Mitigating 51% Attacks

In previous sections, we studied utility functions with limited horizons, in which an attacker is incentivized to perform a 51% double-spending attack and break (strong) attack-payoff security. In this section, we turn to analyzing how to *defend* against 51% attacks. Specifically, given an attacker's utility function with limited horizons, and a cut-off parameter `cutOff` that achieves security in the honest majority setting, we show a way to amplify `cutOff` to obtain security against a rational (and possibly dishonest majority) attacker.

To show attack payoff security, one must show that for *any* adversarial strategy attacking the protocol, there is another adversary who attacks the dummy protocol with access to the ideal ledger functionality $\mathcal{G}_{\text{LEDGER}}$[12], which achieves the same or better utility.

[12] Recall this is the ledger functionality that ensures consistency and liveness, but since we will amplify the cut-off parameter `cutOff`, we only achieve worse overall chain growth and chain quality parameters for the ledger (cf. [BMTZ17a, BGK+20]).

Even more difficult, we place very few restrictions on the adversary: he may corrupt any fraction of parties (e.g. more than majority) and perform any currently known (e.g. block withholding) or unknown strategy. The only restriction we place on the attacker is that he is incentive-driven. Fortunately, a rational attacker is limited by his utility function. As we show, given a utility function satisfying limited horizon, we are able to bound the amount of mining an incentive-driven adversary will do, even in presence of a payoff for forking. Then, by choosing a large enough consistency parameter $\texttt{cutOff}$, we ensure that attackers are disincentivized from creating a fork.

More specifically: We first present in Section 6.1 a result that shows that if an adversary's hashing resources are limited by a budget B, then there is a bound on the interval of rounds where the blockchain is at risk of a consistency failure (Lemma 7). For this, we apply a result from [BGK+20] that, roughly, shows how fast a blockchain's consistency can recover after an attack by an adversary with a given budget (the self-healing property of Bitcoin). Based on this fundamental property, we present in Section 6.2, the main result of the section: Given a utility function with limited horizons, we show a condition on the parameter $\texttt{cutOff}$, depending only on the utility function and protocol parameters, such that $\Pi^{\hat{\mathcal{B}}}$ is attack-payoff secure. To do so, we show that an adversary who spends too much budget will begin to lose utility (Lemma 8), and then combine this result with that of Section 6.1.

6.1 Budget to Vulnerability Period

Assume an instance of the Bitcoin backbone protocol with cut-off parameter ℓ. We distinguish here between ℓ and $\texttt{cutOff}$ for clarity, since we will eventually amplify ℓ to obtain our final cut-off parameter $\texttt{cutOff}$.

Under the honest majority condition, we know that a consistency failure (expressed as the probability that blocks which are ℓ deep in an honest parties adopted chain can be reverted) appears with probability negligible in ℓ (and consequently also in any security parameter κ as long as $\ell = \omega(\log(\kappa))$. We now recall (and state a simple corollary from) the result from [BGK+20], which defines a relationship between an adversary's violation of honest-majority (measured as a so-called budget B by which it can violate the honest-majority condition) and the time until Bitcoin (and more generally, Nakamoto-style PoW chains) self-heals after the adversary returns to below 50% hashing power. That is, until Bitcoin can again guarantee consistency for the part of the chain that is at least ℓ blocks deep in a longest chain held by an honest party. The self-healing time depends on the budget and the parameter ℓ. Recall that θ_{pow} is the usual security threshold for Bitcoin as explained in Section 2.1.

Definition 9 ($(\theta_{pow}, \epsilon, \mathtt{T}_{\mathrm{lb}}, \mathtt{T}_{\mathrm{ub}}, B)$-adversary [BGK+20]). *Let $\theta_{pow}, \epsilon \in (0,1)$, $\mathtt{T}_{\mathrm{lb}}, \mathtt{T}_{\mathrm{ub}}, B \in \mathbb{N}$. A $(\theta_{pow}, \epsilon, \mathtt{T}_{\mathrm{lb}}, \mathtt{T}_{\mathrm{ub}}, B)$-adversary is an adversary[13] satisfying the following: At every round i, let n_a^i and n_h^i be the mining queries made by*

[13] Here the environment is also included in this statement.

corrupt and honest parties in this round. Then, (1) For all i, $\mathtt{T}_{\mathrm{lb}} \leq n_a^i + n_h^i \leq \mathtt{T}_{\mathrm{ub}}$, and (2) For all i, $n_a^i \leq (1-\epsilon) \cdot \theta_{pow} \cdot n_h^i + B_i$, where $B_i \geq 0$ and $\sum_i B_i = B$.

We say the adversary attacks between rounds $a < b$ *if $B_i = 0$ for any $i < a$ or $i > b$ (i.e. he spends all his budget between rounds a and b).*

We say an adversary spends budget B over t rounds, *if the adversary has budget B, and only spends it in rounds $r_1 < r_2 < \cdots < r_t$, such that $\sum_i B_{r_i} = B$.*

The behavior of a blockchain protocol under an attack by an $(\theta_{pow}, \epsilon, \mathtt{T}_{\mathrm{lb}}, \mathtt{T}_{\mathrm{ub}}, B)$-adversary is described by a *vulnerability period*. The vulnerability period is an upper bound on number of rounds before and after an adversary performs the attack, such that protocol is still at risk of a (non-negligible) consistency failure.

Definition 10 (Consistency self-healing property and vulnerability period [BGK$^+$20]). *A protocol is self-healing with vulnerability period (τ_l, τ_h) with respect to consistency, and against a $(\theta_{pow}, \epsilon, \mathtt{T}_{\mathrm{lb}}, \mathtt{T}_{\mathrm{ub}}, B)$-adversary who attacks between rounds (a, b), if the consistency failure event* $\mathrm{ConsFail}_\ell(r)$ *occurs except with at most negligible probability unless $r \in [\rho_\alpha - \tau_l, \rho_\beta + \tau_h]$.* $\mathrm{ConsFail}_\ell(r)$ *is defined as the event that ℓ-consistency is violated in an execution for rounds (r, r'), w.r.t. some round $r' > r$, and any two pairs of honest parties.*

In other words, outside of these "dangerous" rounds $[a - \tau_l, b + \tau_h]$, chains adopted by honest parties are guaranteed to diverge by at the most recent ℓ blocks. Below, [BGK$^+$20] gives a characterization of the vulnerability period in terms of the budget B.

Theorem 3 ([BGK$^+$20]). *A Nakamoto-style PoW blockchain with an upper bound $\mathtt{T}_{\mathrm{ub}}$ of hashing queries per round, maximum network delay Δ, success probability p, and cut-off parameter ℓ satisfies the* consistency self-healing property *with vulnerability period $(\tau_l, \tau_h) = (O(B), O(B) + O(\ell))$ against any $(\theta_{pow}, \epsilon, \mathtt{T}_{\mathrm{lb}}, \mathtt{T}_{\mathrm{ub}}, B)$-adversary, for any $\epsilon, \mathtt{T}_{\mathrm{lb}} > 0$.*

The vulnerability period only bounds the number of "bad" rounds before the attack, and after *all* the budget is spent. For our treatment, we consider a more applicable version of the vulnerability period. In Lemma 7, we show the maximum number of consecutive rounds where ConsFail may occur, by applying the above theorem in a piece-wise fashion. For example, if the adversary spends his budget over a long period of time (e.g., spend a bit of the budget, wait for 2 years, then spend more of his budget), the theorem is not directly suitable for our needs, but it is possible to isolate those "spending" rounds and applying the theorem to each such region. Then, since the total hashing power in the system is bounded, we can use this maximum consecutive "bad" rounds to bound the maximum number of blocks that can be rolled back at any given round.

Lemma 7 (Max consecutive consistency failure rounds and associated number of blocks and rollback). *In the same setting as above in Theorem 3, except with negligible probability the following holds: for any adversary with budget B, spent over t rounds (that is, for t different rounds i it holds that $B_i > 0$),*

there is a maximum number $R(B,t,\ell) = O(B) + O(\ell t)$ *of consecutive rounds* r_j *where* $\text{ConsFail}_\ell(r_j)$ *occurs, during which at most* $W(B,t,\ell) = 2\mathcal{T}_{\text{ub}} p \cdot R(B,t,\ell)$ *blocks are created.*

Proof. To show $R(B,t,\ell)$ we note that the rounds where ConsFail_ℓ occurs, are either (1) rounds where adversary spends his budget (i.e., when adversary has majority) (2) some number of rounds before some amount of budget is spent, bound by τ_ℓ in Theorem 3 (3) some number of rounds after some amount of budget is spent, bound by τ_h in Theorem 3. For (1), there are exactly t rounds by definition. For (2), we observe that this is at most $O(B)$, regardless of how the adversary spends his budget. For (3), we see that this is at most $O(B) + O(\ell t)$, since τ_h gives the adversary at most $O(\ell)$ rounds (where ConsFail_ℓ may occur) each time the budget is spent, and there are exactly t such rounds. Summing the above, we get $R(B,t,\ell) = O(B) + O(\ell t)$. We obtain $W(B,t,\ell)$ through Chernoff bound (and for simplicity using 2 instead of $(1+\delta)$), given that there are at most $\mathtt{T}_{\text{ub}}$ hashes per round. □

Looking ahead, this means that at any point in time, prefixes of honest parties' chains must agree (except with negligible probability) when dropping the most recent $W(B,t,\ell)+\ell$ blocks. Here, we omit the dependency on p because we treat it as a constant parameter of the protocol.

6.2 Attack-Payoff Security

In this section we will show the following: For any utility function with limited horizons, we give a characterization of how to adjust the consistency parameter (depending on the protocol parameters and those of the utility function) such that $\Pi^{\mathbb{B}}$ is attack payoff secure. To do so, we will first upper bound the utility of adversaries who spends a total budget B over some time t, given a utility function $u_{\mathtt{buy}}$ with limited horizons (Lemma 8, and Corollary 1 for utility function $u_{\mathtt{rent}}$). In Theorem 4, we then combine this lemma with the result of the previous subsection, and present our characterization of parameters for which $\Pi^{\mathbb{B}}$ is attack-payoff secure—i.e. for which forks are disincentivized.

Below, we quantify an upper bound $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ on the utility of *any* adversary spending budget of at least B over exactly t rounds, assuming the utility function satisfies limited horizons. Why are we interested in this quantity? Recall $W(B,t)$—which informally represents an interval of blocks where consistency might fail—increases with B and t. Looking ahead, we will find a large enough $W(B,t)$ that disincentivizes attacks (i.e., $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) < 0$). To show that the upper-bound $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ is useful, later we will show that it is possible use it to derive a maximum B,t, which we denote by $\bar{B},\bar{t}$.

Lemma 8 (Upper bound utility of adversary spending budget at least B, over time t).

Suppose $u_{\mathtt{buy}}(\Pi^{\mathbb{B}},\mathcal{A})$ *satisfies limited horizons. Then an adversary* $\mathcal{A}$ *with budget at least* $B > 0$*, and who spends it over exactly* $t \geq \frac{B}{\mathcal{T}_{\text{ub}} - \bar{n}_a}$ *rounds, achieves*

utility at most $u_{\mathtt{buy}}(\Pi^{\not\Bitcoin}, \mathcal{A}) \leq u^{\mathtt{ub}}_{\mathtt{buy}}(B, t)$ *where*

$$\begin{aligned} u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) := &\sum_{x=1}^{t_h} T_{\mathrm{ub}} \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \\ &+ \sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \\ &- \mathtt{ccost}\left(\bar{n}_a + \frac{B}{t}\right) + \mathtt{fpayoff} \end{aligned}$$

and where $t_h := \arg\max_{x \in \mathbb{N}}(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{mcost})$, $\bar{n}_a := \frac{(1-\epsilon)\cdot\theta_{pow}\cdot \mathtt{T}_{\mathrm{lb}}}{1+(1-\epsilon)\cdot\theta_{pow}}$.

If $t < \frac{B}{T_{\mathrm{ub}} - \bar{n}_a}$ *(in this case it is not possible to spend budget* B *over* t *rounds) or* $B \leq 0$*, then* $u^{\mathtt{ub}}_{\mathtt{buy}}(B, t)$ *is undefined.*

Proof. We first note that if $t < \frac{B}{\mathtt{T}_{\mathrm{ub}} - \bar{n}_a}$ then it is by definition not possible for an adversary to spend at least budget B over t rounds: In each round, at most $\mathtt{T}_{\mathrm{ub}}$ hash queries can be made, and given a lower bound of $\mathtt{T}_{\mathrm{lb}}$ mining queries in each round, the adversary must make more than $\bar{n}_a$ queries in order to spend budget.

Thus, let $\mathcal{A}$ be any adversary who spends at least budget B, over exactly t rounds for $t > \frac{B}{\mathtt{T}_{\mathrm{ub}} - \bar{n}_a}$. Let T be the maximum running time of $\mathcal{A}$ in some environment $\mathcal{Z}$. Let $q_1, \cdots, q_T$ be the random variables where q_i is the number of queries $\mathcal{A}$ makes in round i. Let $g = \max_{i \in [1,T]} q_i$ be the maximum number of parties $\mathcal{A}$ corrupts at any given round. Let $t_h = \max_{x \in \mathbb{N}} x : p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{mcost}$, which exists as $u_{\mathtt{buy}}$ satisfies limited horizons. Then,

$$\begin{aligned} u_{\mathtt{buy}}(\Pi^{\not\Bitcoin}, \mathcal{A}) &\leq \sum_{x=1}^{T} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}(g) + \mathtt{fpayoff} \\ &= \sum_{x=1}^{t_h} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \\ &\quad + \sum_{x=t_h+1}^{T} q_i (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}(g) + \mathtt{fpayoff}. \end{aligned}$$

Now by construction $p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq 0$ for $x \leq t_h$ so it is optimal to make as many queries as possible in these rounds. Thus, $\sum_{x=1}^{T} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \leq \sum_{x=1}^{T} g \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost})$. On the other hand, $p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) < 0$ for $x > t_h$ so it is optimal to make as few queries as possible in these rounds. Thus, $\sum_{x=t_h+1}^{T} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \leq \sum_{x=t_h+1}^{t} \bar{n}_a \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost})$. The latter statement is because by assumption that $\mathcal{A}$ spends his budget for t rounds, we have $T \geq t$ and he must make at least $\bar{n}_a + 1$ queries when he spends his budget. Note we are ignoring whether a fork can be obtained this way, as in the upper-bound we give $\mathtt{fpayoff}$ to the adversary "for free".

In addition, it must be that $g \in [\bar{n}_a + \frac{B}{t}, \mathtt{T}_{\mathrm{ub}}]$—he must corrupt enough parties to spend B budget in t time, but corrupting more than $\mathtt{T}_{\mathrm{ub}}$ (the upper bound on total hashing power in the system) parties does not increase his hashing power. This gives us the bound in the lemma (due to limited horizons, $\mathtt{ccost}$ is a non-decreasing function of g). □

As a corollary, by setting $\mathtt{ccost}(\cdot) = 0$ and $\mathtt{mcost} = \mathtt{rcost}$, we obtain an upper bound on the utility of any adversary who spends at least budget B, assuming the utility function satisfies limited horizons with renting.

Corollary 1. *Suppose $u_{\mathtt{rent}}(\Pi^{\not B}, \mathcal{A})$ satisfies limited horizons with renting. Then an adversary $\mathcal{A}$ who spends budget of at least $B > 0$ over exactly $t \geq \frac{B}{T_{\mathrm{ub}} - \bar{n}_a}$ rounds, achieves utility at most $u_{\mathtt{rent}}(\Pi^{\not B}, \mathcal{A}) \leq u^{\mathtt{ub}}_{\mathtt{rent}}(B, t)$ where*

$$u^{\mathtt{ub}}_{\mathtt{rent}}(B,t) := \sum_{x=1}^{t_h} T_{\mathrm{ub}} \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{rcost}) + \sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{rcost}) + \mathtt{fpayoff}$$

and where $t_h := \arg\max_{x \in \mathbb{N}}(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{rcost})$, $\bar{n}_a := \frac{(1-\epsilon)\cdot\theta_{pow}\cdot \mathtt{T}_{\mathrm{lb}}}{1+(1-\epsilon)\cdot\theta_{pow}}$.

A natural question is whether the upper bound $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ and $u^{\mathtt{ub}}_{\mathtt{rent}}(B,t)$ will be useful for bounding B, t. We remark that it is not even trivially clear whether they bounded, as the budget does not limit how many rounds (when honest majority is satisfied) the adversary can mine. Below, we show that there indeed exist a maximum B, t for which $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) \geq 0$ (resp. $u^{\mathtt{ub}}_{\mathtt{rent}}(B,t) \geq 0$, which we denote by $\bar{B}, \bar{t}$.

Lemma 9. *$\bar{B} := \arg\max_{B>0}\left(u^{\mathtt{ub}}_{\mathtt{buy}}(B,\cdot) \geq 0\right)$ and $\bar{t} := \arg\max_{t>0}\left(u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot,t) \geq 0\right)$ exist, or $\forall t > 0, u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot,t) < 0$. The same is true when replacing $u^{\mathtt{ub}}_{\mathtt{buy}}$ with $u^{\mathtt{ub}}_{\mathtt{rent}}$ in the statement.*

Proof. We prove for $u^{\mathtt{ub}}_{\mathtt{buy}}$ then the case for $u^{\mathtt{ub}}_{\mathtt{rent}}$ follows. Let $b_{\max} = \sum_{x=1}^{t_h} \mathtt{T}_{\mathrm{ub}} \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost})$ where $t_h = \arg\max_{x \in \mathbb{N}}(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{mcost})$ (exists by limited horizons), Let $\bar{n}_a = \frac{(1-\epsilon)\cdot\theta_{pow}\cdot \mathtt{T}_{\mathrm{lb}}}{1+(1-\epsilon)\cdot\theta_{pow}}$.

Suppose there is a $t > 0$ such that $u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot,t) \geq 0$; we will show that (1) There is a $t \geq t_h$ such that $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) \geq 0$, and (2) $\bar{t} := \arg\max\left(u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot,t) \geq 0\right)$ exists. For (1),

$$\begin{aligned} & u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) \\ & \leq b_{\max} + \sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}\,(\bar{n}_a + 1) + \mathtt{fpayoff} \\ & \leq b_{\max} + \sum_{x=t_h+1}^{t=t_h} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}\,(\bar{n}_a + 1) + \mathtt{fpayoff}. \end{aligned}$$

The second line is by limited horizons: $\texttt{ccost}$ is a non-decreasing function, so $\texttt{ccost}\left(\bar{n}_a + \frac{B}{t}\right) > \texttt{ccost}\left(\bar{n}_a + 1\right)$. The third line is by assumption $p \cdot \texttt{CR} \cdot \texttt{breward}(x) - \texttt{mcost} < 0$ for $x > t_h$, and taking $t = t_h$ makes this term zero. Since there are no other terms with t, and by assumption that there exists $t > 0$ such that $u^{\texttt{ub}}_{\texttt{buy}}(\cdot, t) \geq 0$, this means $u^{\texttt{ub}}_{\texttt{buy}}(\cdot, t) \geq 0$ for some $t \geq t_h$.

For (2), We bound $\bar{t}$ first, then $\bar{B} < \bar{t}\,(\texttt{T}_{\text{ub}} - \bar{n}_a)$ follows from condition $t < \frac{B}{\texttt{T}_{\text{ub}} - \bar{n}_a}$. By limited horizons, there is a constant (in the security parameter) $\delta > 0$ such that for all $x \geq t_h + 1$, $\texttt{mcost} - p \cdot \texttt{CR} \cdot \texttt{breward}(x) > \delta$. Thus,

$$\begin{aligned}
\bar{t} &:= \arg\max_{t>0} \left(u^{\texttt{ub}}_{\texttt{buy}}(\cdot, t) \geq 0\right) \\
&\leq \arg\max_{t>0} \Big(b_{\max} + \texttt{fpayoff} - \delta \cdot (\bar{n}_a + 1)(t - t_h - 1) - \texttt{ccost}(\bar{n}_a + 1) \geq 0\Big) \\
&= \Big(b_{\max} + \texttt{fpayoff} + \delta \cdot (\bar{n}_a + 1)(t_h + 1) - \texttt{ccost}(\bar{n}_a + 1)\Big) \Big/ \Big(\delta \cdot (\bar{n}_a + 1)\Big).
\end{aligned}$$

The second line is from minimizing the term $\texttt{ccost}(\bar{n}_a + B/t)$ as $\texttt{ccost}(\bar{n}_a + 1)$ (by limited horizons, $\texttt{ccost}$ is a non-decreasing function) and minimizing $\sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \texttt{CR} \cdot \texttt{breward}(x) - \texttt{mcost}) = -\sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (\texttt{mcost} - p \cdot \texttt{CR} \cdot \texttt{breward}(x))$ as $-\delta \cdot (\bar{n}_a + 1)(t - t_h - 1)$ (given what we showed in (1), we have $t \geq t_h$). The third line is positive since we showed there is a $t > t_h$ such that $u^{\texttt{ub}}_{\texttt{buy}}(\cdot, t) > 0$. □

Concrete examples of $\bar{B}$, $\bar{t}$. We give example estimate values of $\bar{t}$, which we recall is the upper bound on the period of time where the attacker is incentivized to spend budget. These values are obtained using the equation from proof of Lemma 9, and from them $\bar{B}$ can be computed easily as $\bar{B} = \bar{t} \cdot (\texttt{T}_{\text{ub}} - \bar{n}_a)$.

In more detail: In the case there is no t such that $u^{\texttt{ub}}_{\texttt{buy}}(\cdot, t) \geq 0$ (resp. $u^{\texttt{ub}}_{\texttt{rent}}(\cdot, t) \geq 0$) ; i.e., it is not profitable to perform a 51% attack at all), we display $\bar{t} = 0$. We use $\texttt{T}_{\text{lb}} \cdot (1 - \epsilon) \approx 1$ to compute parameter $\bar{n}_a$, which requires an attacker to corrupt at least $\sim 50\%$ of the total hashing power. For simplicity we assume that the total hashing power is relatively stable during the period of attack, and thus we set $\texttt{T}_{\text{lb}} = \texttt{T}_{\text{ub}}$. We refer to App. B for details on the other parameters used (e.g., electricity costs, cost per mining rig, block rewards, etc.), which are estimates from Ethereum Classic at the time of writing (Feb. 2021).

In the case of bought mining rigs, we see that $\bar{t} = 0$ for $t_h \leq 89$ days (recall that t_h is the length of time an attacker expects passive-mining to be profitable). Then, we estimate $\bar{t} = 96$ days if $t_h = 90$ days, and $\bar{t} = 196$ days if $t_h = 100$ days. However, the estimated cost to buy a majority of mining rigs can be in the order of tens of millions in USD (using our estimated per-rig cost, around $27 million). This high cost of attack, coupled with the difficulty of purchasing and maintaining such a large number of rigs, may suggest why a 51% attacker might be incentivized to rent mining rigs instead.

In the case of rented mining rigs, we see that the estimated cost of attack becomes more reasonable. Below, we show estimated values of $\bar{t}$, varying both

over t_h (Table 1) and the price of renting `rcost` (Table 2, Fig. 4). In particular, we see that for a realistic (see App. B for source of parameters) renting price of \$0.00045/unit of hashing power, the estimated $\bar{t}$ is around 3 days—on par with the 2-day interval of attack on Ethereum Classic, which we base our numbers on, in August, 2020.

t_h (days)	$\bar{t}$ (days)
0.33	5.9
0.66	6.3
1	6.6
2	7.7
3	8.7
6	11.9
9	15.1

Table 1. $\bar{t}$ varying over anticipated profitable mining time t_h, price/rented rig 0.00023/unit of hashing power (= 435MH), which is \$179,592 per day.

`rcost` (USD)	$\bar{t}$ (days)	Cost/day (USD)
\$0.0001	24.0	\$78,084
\$0.0002	10.5	\$156,167
\$0.0003	4.3	\$234,251
\$0.0004	3.2	\$312,334
\$0.0005	2.6	\$390,418
\$0.0006	2.1	\$468,501

Table 2. Estimated $\bar{t}$ and cost of attack per day, varying over price per rented mining rigs per second. If passive-mining is profitable at all, we set $t_h = 3$ days, and otherwise $t_h = 0$ by default.

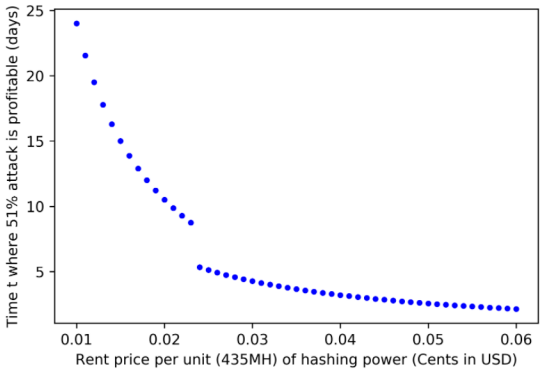

Fig. 4. Graph representation of Table 2.

Finally, we can make a general statement about the attack payoff security of protocol $\Pi^{\text{฿}}$ for any utility function satisfying limited horizons. Informally: our utility function limits how much budget B (spent over how many rounds t) any incentive-driven attacker could reasonably have. Then, if the the window size is large enough to accommodate the largest of such budgets, the protocol is

attack-payoff secure. In the following, $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$, $u^{\mathtt{l}}_{\text{honest}}$, and $W(B,t)$ are from Lem. 8, Lem. 5, and Lem. 7 respectively. The equivalent statement for utility function $u_{\mathtt{rent}}$ can be obtained by using the corollaries of the results above for the case of renting.

Theorem 4. *Let $\mathtt{T}_{\text{ub}}, \mathtt{T}_{\text{lb}} > 0$ be the upper and lower bounds on total number of mining queries per round and $p \in (0,1)$ be the probability of success of each mining query and let $\ell = \omega(\log(\kappa))$. Then, $\Pi^{\mathcal{D}}$ with consistency parameter* `cutOff` *is attack-payoff secure in any model $\mathcal{M}$, whose induced utility $u_{\mathtt{buy}}$ satisfies limited horizons, whenever the condition holds that*

$$\mathtt{cutOff} \quad > \quad \ell \; + \max_{(B,t):u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)>u^{\mathtt{l}}_{\text{honest}}} W(B,t,\ell).$$

The same statement is true replacing $u_{\mathtt{buy}}$ with $u_{\mathtt{rent}}$ and $u^{\mathtt{ub}}_{\mathtt{buy}}$ with $u^{\mathtt{ub}}_{\mathtt{rent}}$.

Proof. We prove for $u_{\mathtt{buy}}$ and $u^{\mathtt{ub}}_{\mathtt{buy}}$, and the renting case follows. We showed that for any (B,t) there is an upper bound $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ on the utility of any adversary who spends at least budget B over exactly t rounds (Lemma 8). From Lemma 5, we obtain a lower bound on the utility $u^{\mathtt{l}}_{\text{honest}}$ of an optimal passive adversary. Since $u^{\mathtt{l}}_{\text{honest}} \geq 0$, by Lemma 9 the maximum in the theorem statement exists. Then, we eliminate any incentive for adversaries to fork by letting $\mathtt{cutOff} > \ell + \max_{(B,t):u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)>u^{\mathtt{l}}_{\text{honest}}} W(B,t,\ell)$, since no adversary can create a fork of more than ℓ blocks except within a period of $R(B,t,\ell)$ rounds (Lemma 7), without his utility being lower than an passive adversary's. We achieve consistency by using the consistency parameter of $W(B,t,\ell)$ blocks, the number of blocks created within this period of rounds. Liveness (that is, chain growth and chain quality ensured by the ledger) follows from [BGK+20]. Now the only parameters affecting our adversary's utility are the block rewards, corruption costs, and mining costs, as we have eliminated forks. Again we work in the $\mathcal{F}_{\text{STX}}$ hybrid which abstracts mining for blocks as an ideal lottery. The cost of playing such a lottery is `mcost` and the reward for winning the lottery *and* inserting the block into the ledger at round r is $\mathtt{breward}(r) \cdot \mathtt{CR}$. Importantly, one cannot get this block reward twice, as attacks that cause forks have been eliminated. In this case, all payoff-inducing events made by the adversary in the real world can be matched by an adversary who invokes an equivalent event in the dummy world with the same payoffs (corruption, mining, and block reward). This implies attack-payoff security. □

We remark the protocol is not *strong* attack-payoff secure: Recall a front-running adversary always maximally delays honest parties' messages. Intuitively, this reduces the mining power in the system, which delays the time between a block is broadcasted, and when it becomes part of the ledger. In effect, this reduces the payoff for block rewards for utilities with limited horizons.

Acknowledgments Yun Lu and Vassilis Zikas acknowledge support from Sunday Group Inc.

References

AKWW19. Georgia Avarikioti, Lukas Käppeli, Yuyi Wang, and Roger Wattenhofer. Bitcoin security under temporary dishonest majority. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 466–483. Springer, Heidelberg, February 2019.

Ber54. Daniel Bernoulli. Exposition of a new theory on the measurement of risk. *Econometrica*, 22(1):23–36, 1954.

BGK+18. Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, October 2018.

BGK+20. Christian Badertscher, Peter Ga?i, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Consensus redux: Distributed ledgers in the face of adversarial supremacy. Cryptology ePrint Archive, Report 2020/1021, 2020. `https://eprint.iacr.org/2020/1021`.

BGM+18. Christian Badertscher, Juan A. Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? A rational protocol design treatment of bitcoin. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 34–65. Springer, Heidelberg, April / May 2018.

BKKS20. Lars Brünjes, Aggelos Kiayias, Elias Koutsoupias, and Aikaterini-Panagiota Stouka. Reward sharing schemes for stake pools. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 256–275. IEEE, 2020.

BMTZ17a. 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.

BMTZ17b. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. Cryptology ePrint Archive, Report 2017/149, 2017. `http://eprint.iacr.org/2017/149`.

Bud18. Eric Budish. The economic limits of bitcoin and the blockchain. Technical report, National Bureau of Economic Research, 2018.

But13. Vitalik Buterin. A next-generation smart contract and decentralized application platform, 2013. `https://github.com/ethereum/wiki/wiki/White-Paper`.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CCWrao20. Kai-Min Chung, T-H. Hubert Chan, Ting Wen, and Elaine Shi (random author ordering). Game-theoretically fair leader election in o(log log n) rounds under majority coalitions. Cryptology ePrint Archive, Report 2020/1591, 2020. `https://eprint.iacr.org/2020/1591`.

CKWN16. 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 2016*, pages 154–167. ACM Press, October 2016.

CM19. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

Coia. 51% attacks archives. url=https://www.coindesk.com/tag/51-attack, publisher=CoinDesk.

Coib. Ethereum classic's mess solution won't provide 'robust' security against 51% attacks: Report. url=https://www.coindesk.com/ethereum-classic-mess-security-51-attacks-report, publisher=CoinDesk.

DPS19. Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 23–41. Springer, Heidelberg, February 2019.

ES14. 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.

Eya15. Ittay Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy*, pages 89–103. IEEE Computer Society Press, May 2015.

For. Rented hash power for 51% attacks is a 'huge vulnerability' for proof-of-work blockchains, says etc labs ceo. url=https://forkast.news/hash-power-51-attack-rent-huge-vulnerability-proof-of-work-blockchain/, publisher=Forkast.

GKL15. 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.

GKM+13. 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.

GKO+20. Juan A. Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2020.

GKW+16. 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 2016*, pages 3–16. ACM Press, October 2016.

HSY+21. Runchao Han, Zhimei Sui, Jiangshan Yu, Joseph Liu, and Shiping Chen. Fact and fiction: Challenging the honest majority assumption of permissionless blockchains. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 817–831, New York, NY, USA, 2021. Association for Computing Machinery.

Inv. 51% attack. url=https://www.investopedia.com/terms/1/51-attack.asp, publisher=Investopedia.

JL20. Jehyuk Jang and Heung-No Lee. Profitable double-spending attacks. *Applied Sciences*, 10(23):8477, 2020.

KRDO17. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017,*

Part I, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

LTKS15. 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 2015*, pages 706–719. ACM Press, October 2015.

Nak08. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `http://bitcoin.org/bitcoin.pdf`.

NKMS16. Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *S&P*, 2016.

PS17. 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.

PSs17. 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, April / May 2017.

Ros11. Meni Rosenfeld. Analysis of bitcoin pooled mining reward systems. *CoRR*, 2011.

SBBR16. 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.

SSZ16. 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.

TJS16. 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.

A Deferred Proofs and Definitions

A.1 Proof of Lemma 1

Proof (Proof of Lemma 1). Similar to [BGM+18] we analyze the modular Bitcoin protocol with access to the state exchange functionality $\mathcal{F}_{\text{StX}}$ (Appendix C.1), which abstracts the proof-of-work puzzle of Bitcoin. Instead of parties querying the hash function, they mine by querying $\mathcal{F}_{\text{StX}}$, which tells them whether the mining attempt was successful.

Consider a real world execution with a front-running, passive adversarial strategy $\mathcal{A}_1$, who makes q^* queries to $\mathcal{F}_{\text{StX}}$ in total (note that the code of this adversary is in fact static, and the number of queries increases as it is executed for more rounds by the environment), and who does not attempt to fork the chain. Let X_i be the random variable where $X_i = 1$ if and only if his ith query successfully mines a block. Then, the real world payoff of $\mathcal{A}_1$ is

$$R_{\mathcal{A}_1} = \left(\sum_{i=1}^{q^*} X_i\right) \mathtt{breward} \cdot \mathtt{CR} - q^* \cdot \mathtt{mcost}$$

Now, consider any adversary $\mathcal{A}_2$ in the real world. Let Q denote the number of queries made by $\mathcal{A}_2$ in an execution of the protocol under the environment $\mathcal{Z}$, and let P_Q be the associated distribution. Then we define $q := \max Supp(P_Q)$ (where $Supp$ is the support). The expected real world payoff of $\mathcal{A}_2$ is

$$E(R_{\mathcal{A}_2}) \leq q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}$$

We want to show that $\mathcal{A}_1$ gains more payoff than $\mathcal{A}_2$. We do this by showing that for an appropriate choice of q^*, the payoff of $\mathcal{A}_1$ exceeds the expected payoff of $\mathcal{A}_2$ with overwhelming probability.

Now, let $X = \sum_{i=1}^{q^*} X_i$, then $E(X) = q^*p$. Let δ be such that $(1-\delta)p \cdot \mathtt{breward} \cdot \mathtt{CR} - \mathtt{mcost} > 0$ (this exists by assumption of $p \cdot \mathtt{breward} \cdot \mathtt{CR} - \mathtt{mcost} > 0$ and that $p, \mathtt{breward}, \mathtt{CR}, \mathtt{mcost}$ are constants). We have

$$\Pr(R_{\mathcal{A}_1} < E(R_{\mathcal{A}_2})) = \Pr\left(X < \frac{q^*\mathtt{mcost} + q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}}{\mathtt{breward} \cdot \mathtt{CR}}\right)$$

We can use Chernoff bound to upper-bound this probability, if $\frac{q^*\mathtt{mcost} + q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}}{\mathtt{breward} \cdot \mathtt{CR}} < (1-\delta)E(X) = (1-\delta)q^*p$. This inequality is satisfied if we set $q^* = \frac{(q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff})\kappa}{(1-\delta)\mathtt{breward} \cdot \mathtt{CR} - \mathtt{mcost}}$, which is still a polynomial in κ. Thus, by Chernoff bound,

$$\Pr\left(X < \frac{q^*\mathtt{mcost} + q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}}{\mathtt{breward} \cdot \mathtt{CR}}\right) \leq \exp\left(\frac{-\delta^2 q^* p}{2}\right) = \mathsf{negl}(\kappa)$$

To show that analysis of the real world utility is sufficient, we have to prove that this utility is the payoff in the ideal world, minimized over simulators that can simulate the adversary ($\mathcal{A}_1$ or $\mathcal{A}_2$), and maximized over all environments. This

is true following the same argument as [BGM+18]: A successful simulator must answer the same number of queries as the adversary. Moreover, the number of mining successes must be the same (up to a negligible difference). Otherwise, the environment can distinguish the real and ideal world. □

A.2 Proof of Lemma 2

Proof (Proof of Lemma 2). The proof goes the same way as in Lemma 1 except here we consider the front-running, passive adversary $\mathcal{A}_1$ who corrupts exactly one party. Let X_i be the random variable where $X_i = 1$ means the ith query yields a succesfully-mined block. Then, the payoff of $\mathcal{A}_1$ is

$$R_{\mathcal{A}_1} = \left(\sum_{i=1}^{q^*} X_i\right) \mathtt{breward} \cdot \mathtt{CR} - q^* \cdot \mathtt{mcost} - \mathtt{ccost}(1)$$

We want to show that the passive strategy gains more payoff than any adversarial strategy $\mathcal{A}_2$. For any adversary $\mathcal{A}_2$: If he corrupts no one, then he has exactly the same payoff as an front-running, passive adversary who corrupts no one so the lemma trivially holds. Thus, let us consider the case if the adversary $\mathcal{A}_2$ corrupts at least one party. Let Q denote the number of queries made by $\mathcal{A}_2$ in an execution of the protocol under the environment $\mathcal{Z}$, and let P_Q be the associated distribution. Then we define $q := \max Supp(P_Q)$. Then, the expected payoff of $\mathcal{A}_2$ is

$$E(R_{\mathcal{A}_2}) \leq q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff} - \mathtt{ccost}(1)$$

Then, the probability

$$\Pr(R_{\mathcal{A}_1} < E(R_{\mathcal{A}_2})) = \Pr\left(X < \frac{q^*\mathtt{mcost} + q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}}{\mathtt{breward} \cdot \mathtt{CR}}\right)$$

which is the same negligible probability as in the proof of Lemma 1 and the proof follows. □

A.3 Chain Growth Lemma

The above proof referred to the chain-growth lemma, we state one version here for completeness:

Lemma 10 (Chain growth, Lemma 7.11 [BMTZ17a]). *For any miner p_i, round number $r \geq 0$, $t \geq 1$, success probability of one mining query $p \in (0,1)$, network delay Δ, and $\delta \in (0,1)$, let $\gamma := \frac{h}{1+h\Delta}$ where $h := 1-(1-p)^q$ and q is the minimum total number of honest mining queries in any round during the interval $[r, r+t]$.*

Suppose p_i is honest in round r, and the longest state received or stored by p_i in round r has length ℓ. Then, in round $r+t$, it holds, except with probability at most $\mathsf{negl}(\gamma t)$[14], that the length of the longest state (received or stored) of at least one honest miner p_j in that round has length at least $\ell + T$ if $t \geq \frac{T}{(1-\delta)\cdot\gamma}$.

[14] In fact, at most $\exp\left(-\frac{\delta^2 t\gamma}{2+\delta}\right)$.

A.4 Formal Definitions of $u_{\texttt{rent}}$ from Section 4

$$u_{\texttt{rent}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z} \in \texttt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \texttt{breward}(r) \cdot \texttt{CR} \cdot \Pr[I^{\texttt{A}}_{b,r}] - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \texttt{rcost} \cdot \Pr[W^{\texttt{A}}_{q,r}] + \texttt{fpayoff} \cdot \Pr[K] \right\} \right\}. \tag{5}$$

A.5 Formal Definitions of $u^{h}_{\texttt{buy}}$ and $u^{l}_{\texttt{buy}}$ from Section 4

$$u^{h}_{\texttt{buy}}(\Pi^{\hat{\mathbb{B}}}, \mathcal{A}) = \sup_{\mathcal{Z} \in \texttt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \texttt{breward}(r + t_{lb}) \cdot \texttt{CR} \cdot \Pr[B^{\texttt{A}}_{b,r}] - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \texttt{mcost} \cdot \Pr[W^{\texttt{A}}_{q,r}] + \texttt{fpayoff} \cdot \Pr[K] - \sum_{g \in \mathbb{N}} \texttt{ccost}(g) \cdot \Pr[C^{\texttt{A}}_{g}] \right\} \right\}.$$

and

$$u^{l}_{\texttt{buy}}(\Pi^{\hat{\mathbb{B}}}, \mathcal{A}) = \sup_{\mathcal{Z} \in \texttt{ITM}} \left\{ \inf_{\mathcal{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \texttt{breward}(r + t_{ub}) \cdot \texttt{CR} \cdot \Pr[B^{\texttt{A}}_{b,r}] - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \texttt{mcost} \cdot \Pr[W^{\texttt{A}}_{q,r}] + \texttt{fpayoff} \cdot \Pr[K] - \sum_{g \in \mathbb{N}} \texttt{ccost}(g) \cdot \Pr[C^{\texttt{A}}_{g}] \right\} \right\}.$$

B Concrete Values Used in Graphs

We discuss the parameters used in the figures. Utility is computed in US dollars (USD). While our results are stated for the Bitcoin protocol, our analyses are in fact quite general, and work for Nakamoto-style PoW blockchains such as Ethereum (Classic) among others. Our graphs use numbers for the popular currency Ethereum Classic, as it has been the victim of several major 51%

double-spending attacks whereas Bitcoin was not. For the numbers to be more accurate, we use recent (as of writing, Feb. 2021) data.

Let time be measured in seconds and fiat currency be measured in USD for simplicity. Even basic mining rigs can make millions of hashes per second, thus we model each party/mining rig as capable of 435MH/s (mega hashes per second), the approximate hashing power of a rig with eight AMD RX 5700 XT graphic cards, which costs about $3000 and uses about 1300W. We use a electricity cost of $0.13/kWh (average cost in the US) to compute the cost of mining with this rig as $\texttt{mcost}$ = $0.000047. From the Nicehash marketplace we obtain the cost to rent 435MH (our basic unit of hashing power per party), which is about 1.96 BTC/TH/day (TH is terahash) at the time of writing. Since the price of Bitcoin (BTC) fluctuates wildly, we consider two cases: the "expensive" case, of $50,000 USD/BTC ($\texttt{rcost}$ = $0.00045), and the "cheap" case, of $22,000 USD/BTC ($\texttt{rcost}$ = $0.00022). We consider utility functions with limited horizons; here consider an attacker who is only incentivized to make blocks before some time t. Thus, we set the payoff for making blocks as $\texttt{breward}(x)$ = $54.4 (which is the block reward given price of ETC at $17) if $x < t$ and $\texttt{breward}(x) = 0$ otherwise. We set the payoff for double-spending to $1 million, a relatively modest amount (e.g., the equivalent of 5.6 million dollars in Ethereum Classic had been stolen from an exchange on August 1, 2020).

In our figures we will consider a maximum total hashing power in the system of 18091 mining rigs (each with hashing power 435MH/s), which is the approximate current hashrate of Ethereum Classic. The probability of mining success for one rig per second is $p = 0.00000435$, computed via the current difficulty of the blockchain. We set network delay $\Delta = 100$ms and set window size $\texttt{cutOff} = 500$ blocks using the deposit and withdrawal confirmation time of the OKEx exchange for ETC (100 and 400 blocks respectively).

C Details on the Bitcoin RPD Model

We refer to [BGM+18] for the full details. Here we recap the core elements of the model for the sake of self-containment.

C.1 Ideal Lottery: The State Exchange Functionality $\mathcal{F}_{\mathbf{StX}}$

For sake of completeness, we include the state exchange functionality $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ as defined in [BMTZ17a]. Here, Δ is the network delay, p_H and p_A are the probabilities for successfully creating a block for each mining attempt—in the flat model, as is our case, $p_H = p_A = p$ (which is a protocol parameter that relates to the *difficulty* of the blockchain). Roughly, the state exchange functionality manages a tree structure containing successfully-mined chains (thus eliminating issues with hash collisions when using hash pointers in the chain), and abstracts mining with random oracle (RO) queries as coin tosses with probability of success p. It also manages the multicasting of successfully-mined chains, and allows the adversary to influence the network via network delay.

Note that in this hybrid world, defining restrictions on the range of the allowed number of mining queries in the system is straightforward by introducing some wrapper $\mathcal{W}_{\mathtt{T}_{\mathrm{lb}}}^{\mathtt{T}_{\mathrm{ub}}}$ that will enforce an upper bound on the queries SUBMIT-NEW and ensure that the system only advances to the next round if at least $\mathtt{T}_{\mathrm{lb}}$ queries have bee submitted. We refer to [BMTZ17a, GKO+20] for details.

Functionality $\mathcal{F}_{\mathrm{StX}}^{\Delta,p_H,p_A}$

The functionality is parametrized with a set of parties $\mathcal{P}$. Any newly registered (resp. deregistered) party is added to (resp. deleted from) $\mathcal{P}$. For each party $p \in \mathcal{P}$ the functionality manages a tree $\mathcal{T}_p$ where each rooted path corresponds to a valid state the party has received. Initially each tree contains the genesis state. Finally, it manages a buffer $\boldsymbol{M}$ which contains successfully submitted states which have not yet been delivered to (some) parties in $\mathcal{P}$. It also manages a buffer $\mathbf{N}_{\mathrm{net}}$ of adverbially injected chunk messages (that might not correspond to valid states).

Submit/receive new states:

- Upon receiving $(\text{SUBMIT-NEW}, \mathrm{sid}, \mathtt{st}, \mathtt{st})$ from some participant $p_s \in \mathcal{P}$, if $\mathsf{isvalidstate}(\mathtt{st}||\mathtt{st}) = 1$ and $\mathtt{st} \in \mathcal{T}_p$ do the following:
  1. Sample B according to a Bernoulli-Distribution with parameter p_H (or p_A if p_s is dishonest).
  2. If $B = 1$, set $\mathtt{st}_{new} \leftarrow \mathtt{st}||\mathtt{st}$ and add $\mathtt{st}_{new}$ to $\mathcal{T}_{p_s}$. Else set $\mathtt{st}_{new} \leftarrow \mathtt{st}$.
  3. Output $(\text{SUCCESS}, \mathrm{sid}, B)$ to p_s.
  4. On response $(\text{CONTINUE}, \mathrm{sid})$ where $\mathcal{P} = \{p_1, \ldots, p_n\}$ choose n new unique message-IDs $\mathrm{mid}_1, \ldots, \mathrm{mid}_n$, initialize n new variables $D_{\mathrm{mid}_1} := D_{\mathrm{mid}_1}^{MAX} := \ldots := D_{\mathrm{mid}_n} := D_{\mathrm{mid}_n}^{MAX} := 1$ set $\boldsymbol{M} := \boldsymbol{M}||(\mathtt{st}_{new}, \mathrm{mid}_1, D_{\mathrm{mid}_1}, p_1)|| \ldots ||(\mathtt{st}_{new}, \mathrm{mid}_n, D_{\mathrm{mid}_n}, p_n)$, and send $(\text{SUBMIT-NEW}, \mathrm{sid}, \mathtt{st}_{new}, p_s, (p_1, \mathrm{mid}_1), \ldots, (p_n, \mathrm{mid}_n))$ to the adversary.
- Upon receiving $(\text{FETCH-NEW}, \mathrm{sid})$ from a party $p \in \mathcal{P}$ or $\mathcal{A}$ (on behalf of p), do the following:
  1. For all tuples $(\mathtt{st}, \mathrm{mid}, D_{\mathrm{mid}}, p) \in \boldsymbol{M}, \mathbf{N}_{\mathrm{net}}$ update value $D_{\mathrm{mid}} := D_{\mathrm{mid}} - 1$.
  2. Let $\boldsymbol{M}_0^p$ denote the subvector of $\boldsymbol{M}$ including all tuples of the form $(\mathtt{st}, \mathrm{mid}, D_{\mathrm{mid}}, p)$ where $D_{\mathrm{mid}} = 0$ (in the same order as they appear in $\boldsymbol{M}$). For each tuple $(\mathtt{st}, \mathrm{mid}, D_{\mathrm{mid}}, p) \in \boldsymbol{M}_0^p$ add $\mathtt{st}$ to $\mathcal{T}_p$. Delete all entries in $\boldsymbol{M}_0^p$ from $\boldsymbol{M}$ and send $\boldsymbol{M}_0^p$ to p. If p is corrupted, provide additionally $\mathbf{N}_{\mathrm{net}}$ to the adversary.
- Upon receiving $(\text{SEND}, \mathrm{sid}, \mathtt{st}, p')$ from $\mathcal{A}$ on behalf some *corrupted* $p \in \mathcal{P}$, if $p' \in \mathcal{P}$ and $\mathtt{st} \in \mathcal{T}_p$, choose a new unique message-ID mid, initialize $D := 1$, add $(\mathtt{st}, \mathrm{mid}, D_{\mathrm{mid}}, p')$ to $\boldsymbol{M}$, and return $(\text{SEND}, \mathrm{sid}, \mathtt{st}, p', \mathrm{mid})$ to $\mathcal{A}$. If $\mathtt{st} \notin \mathcal{T}_p$, then conduct the same steps except that $(\mathtt{st}, \mathrm{mid}, D_{\mathrm{mid}}, p')$ is added to $\mathbf{N}_{\mathrm{net}}$.

Further adversarial influence on the network:

- Upon receiving $(\text{SWAP}, \text{sid}, \text{mid}, \text{mid}')$ from $\mathcal{A}$, if mid and mid$'$ are message-IDs registered in the current message buffers, swap the corresponding tuples in the buffer. Return $(\text{SWAP}, \text{sid})$ to $\mathcal{A}$.
- Upon receiving $(\text{DELAY}, \text{sid}, T, \text{mid})$ from $\mathcal{A}$, if T is a valid delay, mid is a message-ID for a tuple $(\mathtt{st}, \text{mid}, D_{\text{mid}}, p)$ in a message buffer and $D_{\text{mid}}^{MAX} + T \leq \Delta$, set $D_{\text{mid}} := D_{\text{mid}} + T$ and set $D_{\text{mid}}^{MAX} := D_{\text{mid}}^{MAX} + T$.

C.2 The Weak Bitcoin Ledger for the RPD Framework

For sake of completeness we include the Bitcoin ledger functionality $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ that allows forks from [BGM+18] and does not enforce many of the other security relevant features (although consistency is the one we focus on). The ledger has the following defining features:

State Tree: Instead of storing a single ledger state `state`, $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ stores a tree `state-tree` of state blocks where for each node the direct path from the root defines a ledger state. 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 the current-state view of p_i which the adversary can set. The pointer of a honest party can only be set to a node which has a at least the distance to the root of the current pointer node.

Adding Transactions: Submitted transactions are simply collected in `buffer` without any additional check. Transactions in `buffer` which are added to `state-tree` are not removed as they could be reused at an other branch of `state-tree`.

Adding Blocks and Forking: The command NEXT-BLOCK which allows the adversary to propose the next block takes additionally a leaf of `state-tree` as input which defines where the next block will be added. By default the next block is added to the longest branch of `state-tree`. To add the next block to an intermediate node of `state-tree` the adversary may use the command FORK which otherwise provides the same functionality as NEXT-BLOCK.

Extend-Policy: The weak extend-policy weakExtendPolicy that is invoked when the ledger extends one of its branches checks a few simple validity validity conditions. It takes a state-tree `state-tree` and pointer `pt` as input. It first computes a default block $\boldsymbol{N}_{\mathsf{df}}$ which can be appended at the longest branch of `state-tree` without rendering the state invalid. Then it checks if the proposed blocks (by the adversary) $\boldsymbol{N}$ can be safely appended at the node `pt` without violating the validity of the chain. If this is the case it returns $(\boldsymbol{N}, \mathtt{pt})$. Otherwise it returns the $\boldsymbol{N}_{\mathsf{df}}$ and a pointer to the leaf of the longest branch in `state-tree`. In contrast to strong ledger in [BMTZ17a], the weak extend-policy does not check if the adversary inserts too many or too few blocks, does not give guarantees whether old transactions will be included, and does not enforce a fraction of honest blocks.

The ledger further obtains time-stamps from a clock functionality. We refer to [BMTZ17a] for its specification as it is a simple functionality.

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

General: The functionality 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: $\texttt{state-tree} = \texttt{gen}$, $\texttt{NxtBC} := \varepsilon$, $\texttt{buffer} := \emptyset$, $\tau_L = 0$. 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 $\boldsymbol{\mathcal{I}}_H^T$ (initially $\boldsymbol{\mathcal{I}}_H^T := \varepsilon$).

Party Management: 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 [BMTZ17a]). 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 $\mathcal{H} \neq \emptyset$. A party is considered fully registered if it is registered with the ledger and the clock.

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 $(\text{CLOCK-READ}, \text{sid}_C, t)$ set $\tau_L := t$ 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 $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \hat{\mathcal{P}}$. On the other hand, for any synchronized party $p \in \mathcal{H} \setminus \mathcal{P}_{DS}$, if p is not registered to the clock, then $\mathcal{P}_{DS} \cup \{p\}$.
2. If I was received from an honest party $p_i \in \mathcal{P}$:
   (a) Set $\boldsymbol{\mathcal{I}}_H^T := \boldsymbol{\mathcal{I}}_H^T || (I, p_i, \tau_L)$;
   (b) Evaluate $\boldsymbol{R} = ((\boldsymbol{N}_{\texttt{pt}_1}, \texttt{pt}_1, \ldots, (\boldsymbol{N}_{\texttt{pt}_k}, \texttt{pt}_k)) := \mathsf{weakExtendPolicy}(\boldsymbol{\mathcal{I}}_H^T, \texttt{state-tree}, \texttt{NxtBC}, \texttt{buffer})$
   (c) For each pointer $\texttt{pt}_i$ such that $\boldsymbol{N}_{\texttt{pt}_i} \neq \varepsilon$ add path $\mathsf{Blockify}(\boldsymbol{N}_{\texttt{pt}_i,1}), \ldots, \mathsf{Blockify}(\boldsymbol{N}_{\texttt{pt}_i,\ell})$ to `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, $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ 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 fully registered 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 $(\texttt{state-tree}, \texttt{buffer}, \mathcal{I}_H^T)$ 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 $\mathcal{I}_H^T$ as above) $\mathsf{predict\text{-}time}(\mathcal{I}_H^T) = \hat{\tau} > \tau_L$ 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 $I = (\text{NEXT-BLOCK}, \texttt{pt}, \text{hFlag}, (\text{txid}_1, \ldots, \text{txid}_\ell))$ 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 $i = 1, \ldots, \ell$ 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$||\text{txid}_i$.

  (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 $I = (\text{FORK}, \texttt{pt}, (\text{txid}_1, \ldots, \text{txid}_\ell))$ is sent from the adversary, update $\texttt{NxtBC}$ as follows:

  (a) Set listOfTxid $\leftarrow \varepsilon$

  (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 txid $= \text{txid}_i$ then set listOfTxid := listOfTxid$||\text{txid}_i$.

  (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 $I = (\text{SET-POINTER}, (p_{i_1}, \hat{\texttt{pt}}_{i_1}), \ldots, (p_{i_\ell}, \hat{\texttt{pt}}_{i_\ell}))$, 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 $j \in [\ell] : \hat{\texttt{pt}}_{i_j}$ 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 $I = (\text{DESYNC-STATE}, (p_{i_1}, \texttt{state}'_{i_1}), \ldots, (p_{i_\ell}, \texttt{state}'_{i_\ell}))$, 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 $(\text{DESYNC-STATE}, ok)$ to $\mathcal{A}$.

The extend policy for $\mathcal{G}^{\ddot{\mathbb{B}}}_{\text{WEAK-LEDGER}}$ from [BGM+18] is the following:

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

function WEAKEXTENDPOLICY($\mathcal{I}^T_H$, `state-tree`, `NxtBC`, `buffer`)
  Let τ_L be current ledger time (computed from $\mathcal{I}^T_H$)
  // The function must not have side-effects: Only modify copies of relevant values.
  Create local copies of the values `buffer`, `state-tree`, `pt` and $\tau_{\texttt{state}}$.
  // First, create a default honest client block as alternative:
  Set $\texttt{pt}_D$ to the leaf of the longest branch of `state-tree` and denote by $\texttt{state}_D$ the corresponding state.
  Set $\boldsymbol{N}_{\mathsf{df}} \leftarrow \texttt{tx}^{\text{coin-base}}_{\text{minerID}}$ of an honest miner
  Sort `buffer` according to time stamps.
  Let $\texttt{tx} = (\texttt{tx}_1, \ldots, \texttt{tx}_\ell)$ be the transactions in `buffer`
  Set $\texttt{st} \leftarrow \mathsf{blockify}_{\ddot{\mathbb{B}}}(\boldsymbol{N}_{\mathsf{df}})$
  repeat
    Let $\texttt{tx} = (\texttt{tx}_1, \ldots, \texttt{tx}_\ell)$ be the current list of (remaining) transactions
    for $i = 1$ to ℓ **do**
      if $\mathsf{ValidTx}_{\ddot{\mathbb{B}}}(\texttt{tx}_i, \texttt{state}_D || \texttt{st}) = 1$ **then**
        $\boldsymbol{N}_{\mathsf{df}} \leftarrow \boldsymbol{N}_{\mathsf{df}} || \texttt{tx}_i$
        Remove $\texttt{tx}_i$ from `tx`
        Set $\texttt{st} \leftarrow \mathsf{blockify}_{\ddot{\mathbb{B}}}(\boldsymbol{N}_{\mathsf{df}})$
      end if
    end for
  until $\boldsymbol{N}_{\mathsf{df}}$ does not increase anymore
  // Now, parse the proposed block by the adversary
  // Possibly more than one block should be added, possibly at several places
  $\boldsymbol{R} \leftarrow \varepsilon$ // Result variable
  for each `pt` such that $\texttt{NxtBC}[\texttt{pt}] \neq \varepsilon$ **do**
    Set `state` to the state corresponding to pointer `pt` (i.e., a path in `state`-tree)
    Parse $\texttt{NxtBC}[\texttt{pt}]$ as a vector $((\text{hFlag}_1, \texttt{NxtBC}_1), \cdots, (\text{hFlag}_n, \texttt{NxtBC}_n))$
    $\boldsymbol{N}_{\texttt{pt}} \leftarrow \varepsilon$ // Initialize Result
    for each list $\texttt{NxtBC}_i$ of transaction IDs **do**
      // Compute the next state block
      $\boldsymbol{N}_i \leftarrow \varepsilon$
      // Verify validity of $\texttt{NxtBC}_i$ and compute content
      Use the txid contained in $\texttt{NxtBC}_i$ to determine the list of transactions
      Let $\texttt{tx} = (\texttt{tx}_1, \ldots, \texttt{tx}_{|\texttt{NxtBC}_i|})$ denote the transactions of $\texttt{NxtBC}_i$
      if $\texttt{tx}_1$ is not a coin-base transaction **then**
        return $(\boldsymbol{N}_{\mathsf{df}}, \texttt{pt}_D)$
      else
        $\boldsymbol{N}_i \leftarrow \texttt{tx}_1$
        for $j = 2$ to $|\texttt{NxtBC}_i|$ **do**
          Set $\texttt{st}_i \leftarrow \mathsf{blockify}_{\ddot{\mathbb{B}}}(\boldsymbol{N}_i)$
          if $\mathsf{ValidTx}_{\ddot{\mathbb{B}}}(\texttt{tx}_j, \texttt{state} || \texttt{st}_i) = 0$ **then**
            return $(\boldsymbol{N}_{\mathsf{df}}, \texttt{pt}_D)$

end if
$\boldsymbol{N}_i \leftarrow \boldsymbol{N}_i || \mathtt{tx}_j$
end for
Set $\mathtt{st}_i \leftarrow \mathsf{blockify}_{\mathbb{B}}(\boldsymbol{N}_i)$
end if
$\boldsymbol{N}_{\mathtt{pt}} \leftarrow \boldsymbol{N} || \boldsymbol{N}_i$
$\mathtt{state} \leftarrow \mathtt{state} || \mathtt{st}_i$
$\boldsymbol{\tau}_{\mathtt{state}} \leftarrow \boldsymbol{\tau}_{\mathtt{state}} || \tau_L$
end for
$\boldsymbol{R} \leftarrow \boldsymbol{R} || (\boldsymbol{N}_{\mathtt{pt}}, \mathtt{pt})$
Update (the local copy of) `state-tree` to include the extended path `state`
end for
return $\boldsymbol{R}$
end function