PoW Micronomics: A Fine-Grained Model for the Economic Analysis of Proof-of-Work Blockchains

Juan Garay[1] Yun Lu[2] Julien Prat[3] Brady Testa[4] Vassilis Zikas[5]

[1] Texas A&M University, garay@tamu.edu
[2] University of Victoria, yunlu@uvic.ca
[3] Ecole Polytechnique, Julien.Prat@ensae.fr
[4] Texas A&M University, btesta@tamu.edu
[5] Georgia Tech, vzikas@gatech.edu

Abstract. Following the cryptographic security analyses of proof-of-work (PoW) blockchain protocols, a line of research has focused on their economic robustness. The two core questions asked are: How resilient is the system to rational attacks, and how profitable it is for miners to execute it. However, to our knowledge, no work to date has attempted to address them considering the full complexity of the blockchain protocol, including difficulty readjustment, which is needed to handle dynamic participation, price fluctuations, and the impact of risk.

In this work, we provide a fine-grained game-theoretic analysis for Nakamoto-style PoW blockchains, which takes into account both incentives of parties to deviate and complications introduced by difficulty readjustment. Our results employ the *Rational Protocol Design* framework of Garay *et al.* [FOCS'13] and extend recent works on the economic robustness of the Bitcoin backbone protocol to the variable difficulty setting.

Notably, our fine-grained specification of miners' utility incorporates variable difficulty adjustment alongside factors like the average cost of mining and the depreciation factor, which, despite being common in economics, are typically either abstracted as exogenous parameters or ignored in the blockchain literature. We showcase the expressivity and usefulness of our formulation of utilities by using it to provide estimates and trends for such factors across several real-world cryptocurrencies.

Table of Contents

1 Introduction 2
1.1 Our Contributions 5
1.2 Organization of the Paper 6
2 Preliminaries 6
2.1 Blockchain Basics 6
2.2 Network Model 7
2.3 Bitcoin as a UC protocol 8
2.4 Rational Protocol Design 9
2.5 RPD on PoW-based Blockchains, Static Difficulty 12
3 Utility Functions for the Dynamic Participation Setting 13
3.1 The Relaxed Ledger Functionality 13
3.2 The Utility Functions 14
4 Incentive Compatibility Analysis 16
4.1 Discounting Captures Bounded Incentives 16
4.2 When is Mining Profitable? 17
4.3 Incentive Compatibility of Bitcoin 20
5 Empirical Validation 25
5.1 Overview of the Experiments 25
5.2 Interpretation of Results 29
A Glossary of Terms in the Utility Definitions 33
B Useful Theorems and Definitions 33
C Ideal Ledger Functionalities and Algorithms 34

1 Introduction

Modern cryptocurrencies fuel a market with a multi-trillion dollar market capitalization. In response, a number of works have focused on the economic analysis of these systems. However, such works would typically treat the underlying blockchain ledger as an ideal ledger or bulletin board.

This abstraction is supported by a large body of research in cryptography, security, and distributed computing which proves that several of the underlying blockchain protocols are secure under reasonable assumptions about the system's resource allocation. For example, so-called Nakamoto-style proof-of-work (PoW) blockchains such as Bitcoin, can be proven secure assuming the majority of the hashing power invested on the system is controlled by so-called *honest* parties, i.e., who faithfully execute the Bitcoin protocol; this is often referred to as the honest majority of hashing power assumption. Similarly, proof-of-stake systems—e.g., Cardano, Ethereum v2+, Algorand, etc. [18,43,34]—assume that the majority of the stake in the system is held by such honest parties.

Yet, basing the soundness of a game-theoretic analysis on such an axiomatic security assumption, however useful it may be for making clean and simple statements, goes against the principle that parties are rational. Indeed, in all such protocols, parties receive compensation for maintaining the blockchain (i.e., mining in Bitcoin) in the form of rewards or transaction fees. As such, deviating

from the actual protocol (which would render them dishonest and therefore affect the assumption) might in some cases be the strategy that would be preferred by an honest miner.

The above discussion motivates the game-theoretic analysis of the blockchain protocol itself. Indeed, several works have performed such analyses [33,16,42].

The first mover's advantage of Bitcoin—making it by far the leading cryptocurrency in terms of market cap—and the intriguing tension between rewards and associated costs (due to energy-intensive mining) of PoW, makes it one of the most studied blockchains from the above game-theoretic angle. Notwithstanding, to our knowledge, all existing game-theoretic treatments of Naka moto-style PoW protocols have ignored important aspects of the protocol—such as taking into account the effect of *difficulty readjustment* [3,6]—which play a crucial role in their stability.

To explain difficulty readjustment and its importance in PoW, let us briefly focus on Bitcoin. In a nutshell, every 2016 blocks, the Bitcoin protocol readjusts how difficult it is to solve a hash puzzle using data from the previous epoch. To maintain an approximate constant block production rate, the protocol proceeds in epochs of 2016 blocks, where each epoch has a fixed difficulty. At the epoch boundary, the protocol takes into account the number of blocks that were generated in the epoch and adjusts the difficulty of the puzzle—if the block production rate is too high (resp. too low), it makes solving the puzzle harder (resp. easier) to reduce (resp. increase) this rate.

It is worth noting that analyzing Bitcoin under (permanently) fixed difficulty is not uncommon, even in the classical cryptography and security literature. Indeed, several of the early works that performed a cryptographic analysis [26,36] explicitly make this assumption. To our knowledge, the first complete cryptographic analysis of the actual Bitcoin (-backbone[6]) protocol was done in [27], with the first composable analysis appearing very recently [24].

When transitioning to a game-theoretic analysis of (PoW) block-chains, several additional simplifications have been made. An early analysis by Kiayias *et al.* [33] assumes fixed difficulty, does not take into account transaction fees, the explicit tension between mining cost and rewards (which is closely connected to the cost of electricity and the price of Bitcoin), the sequential (game) spirit of Bitcoin mining and the fact that to use cryptography within such a game, one needs to worry about the effect of asymptotics. (Indeed, a strand of literature has demonstrated the need for cryptography-friendly stability notions [20,30,29,1,23].) The above simplifications were also made in earlier works on the stability of Bitcoin [21,40] which demonstrated that it does not achieve an intuitive notion of fairness, which would require that the profit of the miners is proportional to the hashing power they invest in the system.

A more recent line of work analyzed PoW in the Rational Protocol Design (RPD) model of Garay *et al.* [25]. This model was introduced to incorporate rational attacks to a cryptographic protocol while taking into account the sequential-game character of mining and embracing asymptotics. The first such analysis [3] proved that the Bitcoin backbone is in fact, *attack-payoff secure*. (which is the term coined in [25] to refer to protocols secure against a rational attacker, who might deviate in an adversarial, coordinated manner, but only in ways that maximize their expected utility). They also proved that assuming a high-enough price of Bitcoin (in comparison to the unit cost of hashing/electricity) it is also (RPD-)*incentive compatible*, which intuitively says that if there is no attack, then parties prefer mining to not mining.

The results from [3] consider a natural but simplistic utility, taking into account the gains by including a block on the blockchain—incurred by transaction fees or block rewards—versus the cost

[6] The terminology *Bitcoin backbone* was minted by Garay *et al.* [26] to refer to the consensus layer of Bitcoin, which focuses on the mining process and ignores application-related issues such as transaction fees, block rewards, etc

of mining (where the latter was brought to the same unit as the former by means of a conversion rate parameter). They proved that as long as expected gains (in a polynomially long execution) exceed expected rewards, then Bitcoin is attack-payoff secure, and if this excess gap is large enough (because of the high price of Bitcoin for example) then it is even incentive compatible.

The above result is an ideal example which demonstrates that abstracting away details of the implementation or utility-relevant aspects can yield counter-intuitive results and/or apparent mismatches in the relevant literature. In particular, follow-up work by Badertscher *et al.* [6] observed that the results from [3] were rooted in a folklore-theorem of game theory and economics: If players are sufficiently patient, then any feasible and individually rational payoff can be sustained as a Nash equilibrium of the repeated game. For instance, one of the attack-payoff security statements from [6] relied on the following abstract reasoning: If the reward of the adversary in a fork is polynomially bounded, then forking is weakly dominated (i.e., yields no better utility than) by the strategy of running the protocol for a (longer) polynomial number of rounds and collecting the associated rewards.

The solution was to refine the utilities' definitions, akin to the common way in which economists resolve the above folkloric inconsistency: **discounting with time**. Specifically, the approach in [6] augmented the utility from [3], introducing the notion of a utility function's *limited horizons*. Simply put, limited-horizon functions essentially prevent an artifact that appears when a utility function is defined over infinite time. They show that if the utility function does not abide by limited horizons, then any adversary can simply mine honestly for a bit longer, and thus in any setting it **always** is attack-payoff secure, as an adversary will never be incentivized to deviate from the protocol. Naturally, adding such a discounting factor on the utility makes the attack-payoff security depend on the protocol parameters.

Notwithstanding, the results in both [6] and [3] appear to be in tension with classical (in)stability statements about Bitcoin, the immediate question being: How do these results not contradict the selfish mining attack of Eyal and Sirer [21], which has become the established reference for the statement that Bitcoin is not incentive compatible? The answer to this is subtler than one might think and further demonstrates the need for a more expressive utility function: In a nutshell, the selfish mining attack [21] does not (necessarily) break the security (i.e., the backbone properties) of Bitcoin, but rather its "fairness"—i.e., an attacker who invests almost the same amount of work as the honest parties, can get disproportionally more rewards than they do. To argue that this renders attacking the protocol a best response, one needs to assume that the attacker's utility does not only depend on rewards vs costs (as in [6,3]), but also on whether or not the attacker gets disproportionally more utility than the other parties that are not part of the attack (coalition).

One might be tempted to think that the solution is to adjust the utility of the attacker by taking fairness into account. However, there is a clear argument against this (which as we will see, will lead to the necessity of adding variable difficulty to the model): When the attacker has favorable network conditions (e.g., assuming he finds a PoW around the same time as an honest party, he gets to propagate it faster) [8] and the difficulty parameter is fixed as is common in game-theoretic analyses, then the net profit (rewards minus costs) of the attacker does not change by performing selfish mining. All the attacker can do is simply make honest parties "waste" their hashing power. It is unclear if such a "spite" addition to the utility corresponds to what Bitcoin miners really care about, and most importantly, if we could quantify the "spitefulness"-satisfaction in relation to the real rewards in a utility definition. In the following, we will refer to an attacker without such a spite-related incentive as *financially driven*.

As we already hinted above, the key to resolving the above mismatch is to actually analyze the PoW protocol (with difficulty readjustment). The reason is that although wasting honest parties' effort cannot increase the maximum profit of a financially driven attacker, things change drastically when taking difficulty readjustment into account: By making honest parties waste their hashing power, the adversary reduces the difficulty which (at least temporarily) increases the rate at which he produces blocks (in the following epoch), and therefore the expected rewards, without necessarily decreasing the hashing cost. As such, selfish mining becomes potentially profitable even for a financially driven adversary.

1.1 Our Contributions

Motivated by the above, we propose the first analysis of the attack-payoff security of Nakamoto-style PoW protocols (like Bitcoin, Bitcoin Cash, and Ethereum Classic) which take into account, to our knowledge for the first time, both variable difficulty adjustment and the basic factors dominating the miners' financial incentives, including the folklore discounting-with-time aspect (which captures, among others, interest-based asset appreciation and mining equipment depreciation), the price of the crypto asset, and the average cost of electricity and mining rigs.

Putting this utility formulation to work, we provide sufficient conditions on the mining protocol for it to be attack-payoff secure (i.e., secure against a financially driven adversary) without the need to assume honest majorities. We view this result as an important fallback security statement, which does not rely on the axiomatic assumption that the attacker cannot control a majority of the hashing power—which has been proven to fail in cases in smaller blockchains like Bitcoin Cash and Ethereum Classic. We also study sufficient conditions for the protocols to be (RPD-) incentive compatible, which was not even known for the fixed difficulty setting (assuming a utility with discounting with time).

In more detail, our contributions are as follows:

(1) We propose the first (RPD-) attack-payoff utility function for PoW blockchains which accounts for dynamic participation and variable difficulty adjustment, and includes basic parameters of the miner's financial incentives including the discounting function, the price of the associated PoW-based crypto asset, the costs associated with mining, and the potential gain of the attacker for provoking a deep fork. We specify utilities for both the attacker and the parties running the protocol and discuss how the various parameters capture real-world sources of revenue and cost. Our model captures fluctuations in price, fee structure, and mining costs, allowing us to model market dynamics in ways that the previous analyses do not permit. We refer to our utility as the *micronomic* utility function.

(2) Putting our utility function into use, we characterize the parameter regions which allow for proving attack-payoff security and incentive compatibility of the underlying PoW asset. We demonstrate that if the cut-off parameter and the price of the cryptocurrency are sufficiently high and the costs are sufficiently low, one does not need the honest majority assumption to prove security against a financially driven attacker. This provides a game-theoretic fallback to standard cryptographic analysis (which assumes honest majority of hashing power). In particular, it (a) formalizes the intuition that the reason an attacker does not attack the system (e.g., by corrupting a majority of the miners) is that this would be a dominated strategy, and (b) provides a characterization for what parameters (e.g., price range, block production rate, etc.) are "safe." In addition to attack-payoff security, we provide the parameter range for which the PoW blockchain is (RPD-) *incentive*

compatible. To our knowledge, no existing work has performed such an analysis for utilities that are discounted over time, even in the fixed difficulty setting.

(3) Importantly, we corroborate our theoretical modeling with several real-world case studies. These studies incorporate periods of higher and lower pricing volatility, and also encompass a 51% attack. We demonstrate that even in the periods of high volatility and during forking attacks that we examined, miners are still incentivized to continue mining. Most interestingly, we can validate the zero-profit condition under free entry, and we can detect anomalous periods in which miners' risk profile spikes (indicating movement across PoW chains).

1.2 Organization of the Paper

The rest of the paper is organized as follows. In Section 2, we provide preliminary material on blockchains, Universal Composability (needed for RPD) and the RPD framework, as well as detail our network model. In Section 3, we introduce the utility functions for the dynamic-participation, variable-difficulty setting, while Section 4 establishes the bounds of honest-miners' utilities, which leads to a characterization of under what conditions is an honest miner incentivized to mine. Finally, Section 5 explores several real-world examples that corroborate our modeling, assumptions, and results. A glossary of terms, the detailed description of ideal functionalities and algorithms, as well as other complementary material, are presented in the appendix.

2 Preliminaries

Here we discuss the fundamentals of PoW-based blockchains. We refer the reader to [26,37] for a more detailed discussion of the topic in the fixed difficulty setting, and to [27,17] in the variable difficulty setting. We further discuss their description in the UC framework [12], and then discuss Rational Protocol Design (RPD) [25] and how it has been applied to PoW-based blockchains [3,6].

We say that a function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every $c \in \mathbb{N}$, there exists n_0 such that for $n > n_0$, we have that $|f(n)| < 1/n^c$. We let $\mathsf{negl}(\kappa)$ denote a negligible function with respect to κ, the security parameter.

2.1 Blockchain Basics

A *blockchain* $\mathcal{C} = \mathbf{B}_1, \ldots, \mathbf{B}_n$ is a (finite) sequence of blocks where each *block* $\mathbf{B}_i = \langle \mathsf{pt}_i, \mathtt{st}_i, \mathsf{n}_i, T_i, t_i \rangle$ is a quintuple consisting of the (hash) *pointer* to the previous block pt_i, the *state block* $\mathtt{st}_i$ containing transactions, the target for the hash value T_i, the timestamp t_i, and the *nonce* n_i. $\mathbf{B}_1$ is referred to as the *genesis block*, sometimes also referred to as $\mathbf{G}$. Formally, we define $\mathbf{G} = \langle \perp, \mathtt{gen}, \perp, T_0, 0 \rangle$, where T_0 is an initial target for the blockchain. The genesis block serves as the first block in a chain, and as such has no pointer pt_i or nonce n_i. The *head* of a chain $\mathcal{C}$ is denoted $\mathsf{head}(\mathcal{C}) = \mathbf{B}_n$, and *length* $\mathsf{length}(\mathcal{C}) = n$. The sequence of the first $\mathsf{length}(\mathcal{C}) - k$ blocks of $\mathcal{C}$, i.e $(\mathcal{C}^{\lceil k} = \mathbf{B}_1, \ldots, \mathbf{B}_{length(\mathcal{C})-k})$ is denoted $\mathcal{C}^{\lceil k}$. Note that $\mathcal{C}^{\lceil k}$ itself is also a valid chain. If we have two chains C_1, C_2, and write $C_1 \preceq C_2$, we mean that C_1 is a prefix of C_2.

We can further delineate the chain and its state. We say that a chain $\mathcal{C}$ has the state $\vec{\mathtt{st}} = \mathtt{st}_1 \parallel \cdots \parallel \mathtt{st}_n$. $\vec{\mathtt{st}}$ contains the data portion of the blockchain, which in the case of Bitcoin is the transaction ledger. We map the state blocks $\vec{\mathtt{st}} = \mathsf{blockify}_{\mathbb{B}}(\vec{N})$ following the notation introduced in [35] and adopted in [7,24]. $\vec{N}$ refers to a vector of transactions to be included into the network.

In (most) blockchains, it is assumed that participants may freely come and join and leave the network, leading to a fluctuation in the number of parties in the network. To try and stabilize block generation rates to be roughly constant, Bitcoin implements a *target recalculation* mechanism, which activates after every m blocks (known as an epoch). This mechanism examines the timestamps of the last m blocks, and calculates a new difficulty so that the network will on average produce a block in expected time $\mathsf{b_{tgt}}$. This calculation is sent through a dampening filter, preventing the new value from fluctuating outside of the interval $[\frac{1}{\tau}, \tau]$. In Bitcoin, the parameters are $m = 2016$, $\mathsf{b_{tgt}} = 10$ minutes, and $\tau = 4$.

As the blockchain extends by miners adding new blocks, each block is linked to previous blocks by means of a hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$. In a chain $\mathcal{C}$, we require that $H[\mathbf{B}_{i-1}] = p_i$. We also require that if $i < j$ then two timestamps t_i, t_j must satisfy $t_i < t_j$. It is also necessary for the hashes to be valid PoWs. Let $\mathsf{gettarget}(\vec{\mathtt{st}})$ refer to the target calculation function; we provide the algorithm in Appendix C. Then we have for $\mathcal{C}^{\lceil i}$ and the corresponding state $\vec{\mathtt{st}} = \vec{\mathtt{st}}_1 \parallel, \ldots, \parallel, \vec{\mathtt{st}}_i$, it holds that $H[\mathbf{B}_i] < \mathsf{gettarget}(\vec{\mathtt{st}})$ is true for all $i > 1$. We refer to the case where $\mathsf{gettarget}(\vec{\mathtt{st}})$ returns a constant value as the *static* difficulty model, and the case where it may return different values as the *dynamic* (or variable) difficulty model. It is also possible for multiple pointers to point to the same block, formally: $H[\mathbf{B}_{i-1}] = \mathsf{pt}_i = \mathsf{pt}_{i'}$ where $i \neq i'$. This event is referred to as a fork, as it can be visualized as a branch extending from a chain. Thus, the blockchain is represented as a tree $\mathcal{T}$. It is worth noting that while forks usually are negative events, some fork events are planned to branch the network into two, typically to add/change key components of the blockchain. For example, Ethereum switching over to a proof of stake (PoS) consensus mechanism.

2.2 Network Model

The network resources assumed to be accessible to parties are described below as ideal UC functionalities.

- **Random Oracle.** The random oracle $\mathcal{F}_{\text{RO}}$ functionality ideally models a hash function as a random function. We use the formulation provided in [7], subsequently adopted in [24]. The oracle models a function $H : \{0,1\}^* \to \{0,1\}^\kappa$, where κ is the security parameter. $\mathcal{F}_{\text{RO}}$ internally stores a table of all previous input/output pairs. If a new input is provided, then $\mathcal{F}_{\text{RO}}$ samples a uniform $y \in \{0,1\}^\kappa$ to map its output to; otherwise it refers to its internal table for the result. In Bitcoin, the random oracle is used to implement the PoW mechanism. Restricting the number of queries to the oracle is accomplished by means of a wrapper $\mathcal{W}^q(\mathcal{F}_{\text{RO}}^\kappa)$, which ignores queries beyond the allotted amount.
- **The Clock.** In this work, we assume access of protocol participants to a global clock functionality $\bar{\mathcal{G}}_{\text{CLOCK}}$ [7,32]. $\bar{\mathcal{G}}_{\text{CLOCK}}$ allows for the protocol to proceed synchronously (recall that by default, the UC formulation assumes asynchronous execution). It is known that global functionalities require a regularity condition, and thus $\bar{\mathcal{G}}_{\text{CLOCK}}$ enforces that participants register with a REGISTER command. The session id associated with the clock is *cid*. $\bar{\mathcal{G}}_{\text{CLOCK}}$ keeps track of a counter τ_{cid}, which tracks the progression of time in each session. After inputs are received from all parties in the session *cid*, the clock will increment τ_{cid}. Parties can send (CLOCK-READ, *cid*) to $\bar{\mathcal{G}}_{\text{CLOCK}}$, and will receive (CLOCK-READ, *cid*, τ_{cid}) in response.
- **Network Communication.** The network diffusion functionality $\mathcal{F}_{\text{DIFF}}$ (cf. [26]) provides an interface for parties to "broadcast" messages to the network. For Bitcoin, this is used to broadcast transactions for inclusion, as well as successful proofs of work (PoWs). $\mathcal{F}_{\text{DIFF}}$ maintains a

party set, and a list of messages for which it keeps track of the current time remaining before the message can be received (see below). $\mathcal{F}_{\text{DIFF}}$ receives a value Δ from the adversary that indicates the maximal delay permissible in the execution.

2.3 Bitcoin as a UC protocol

The Universal Composability (UC) framework developed by Ran Canetti [12] has become the gold standard for analyzing cryptographic protocols in settings where they may be arbitrarily composed with other protocols (even themselves). Such a framework was motivated by the realization that there are protocols which may be secure in the stand-alone setting, but are not under arbitrary composition. In a nutshell, the UC framework introduces the environment, and treats all machines as Interactive Turing Machines. The UC framework crucially provides a composition theorem, which describes how protocols that realize their functionalities can be used in conjunction with other protocols. A full, detailed description of the UC framework is out of scope for this work; readers are encouraged to read [12,15,14], for a more complete understanding of the topic.

After property-based treatments of the Bitcoin protocol [26,36] in the static difficulty case, the Bitcoin Protocol in the static difficulty model was subsequently analyzed in the UC framework [7]. This approach, which we adopt to implement the weakened ledger, assumes access to a diffusion functionality $\mathcal{F}_{\text{DIFF}}$, a random oracle $\mathcal{F}_{\text{RO}}$, and a global clock $\bar{\mathcal{G}}_{\text{CLOCK}}$.[7] Informally, the clock receives updates from individual parties that signal that they are ready to proceed to the next round. Once all honest parties have signaled, the round counter stored in the clock is incremented. There is a delay parameter Δ, in which messages must be delivered before.

The UC ledger $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$ [7] modularizes its analysis by abstracting the lottery process of the proof of work mechanism into a functionality $\mathcal{F}_{\mathbf{StX}}$. $\mathcal{F}_{\mathbf{StX}}$, upon receiving $(\text{SUBMIT-NEW}, sid, \vec{\mathtt{st}}, \mathtt{st})$ from a registered party, samples a Bernoulli distribution to determine whether $\mathtt{st}$ is to extend $\vec{\mathtt{st}}$. If it is, then the state is stored internally, and is available for other parties to retrieve by sending $(\text{FETCH-NEW}, sid)$. The adversary can send messages on behalf of a corrupted party, and enforce a delay on messages, so long as the total delay for the message is less than Δ. $\mathcal{F}_{\mathbf{StX}}$ is shown to be realizable in the $(\mathcal{F}_{\text{RO}}, \mathcal{F}_{\text{DIFF}})$-hybrid world.

It is then shown that the ledger functionality $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$ is realizable in the $(\mathcal{F}_{\mathbf{StX}}, \mathcal{F}_{\text{DIFF}})$-hybrid world, provided restrictions on the computational power available to miners. $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$ enforces bounds on the common prefix, chain quality, and chain growth [26]. For example, reading from the ledger is done by a party that sends (READ, sid), and receives $(\text{READ}, sid, \mathsf{state}_i)$ as response, where state_i is a state that is truncated no more than the common prefix parameter. This truncation is determined by the adversary, who is allowed to set the state slackness, for which honest parties may not yet agree on. Note that the ledger functionality doesn't enforce that all honest parties must see the same state, as such a guarantee is too strong to realize.

$\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$ only works in the static difficulty case, where no difficulty adjustments occur. Later work [24] extends the UC-treatment into the dynamic difficulty setting. To provide security guarantees in the cryptographic setting, Garay *et al.* [27] proposed the following restriction on the rate at which parties can come and go from the network. We note that we dispense of this notion for our

[7] Note that $\bar{\mathcal{G}}_{\text{CLOCK}}$, and later on $\mathcal{G}_{\text{LEDGER}}^{\mathbb{B}}$, are modeled as *global* functionalities. Intuitively, these are functionalities that we desire to be accessible to separate protocols, but this can violate security guarantees. See, e.g., [13,2,5] for details.

rational analysis. This restriction is done by enforcing that the sequence of the number of parties (n_r) is a (γ, s)-respecting sequence:

Definition 1 (Party Fluctuation [27]). *For $\gamma \in \mathbb{R}^+$, a sequence $(n_r)_{r\in\mathbb{N}}$ is called (γ, s)-respecting if for any set S of at most s consecutive rounds, $\max_{r\in S} n_r \leq \gamma \cdot \min_{r\in S} n_r$.*

In [24], $\mathcal{F}_{\mathtt{StX}}$ is separated into a static $\mathcal{F}_{\mathtt{StX}}(\mathcal{P}, \Delta, p_H, p_A)$, which is treated as a static difficulty instance of the state exchange functionality, and $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathtt{StX}})$, which is a "wrapper" over $\mathcal{F}_{\mathtt{StX}}(\mathcal{P}, \Delta, p_H, p_A)$. Here, $\mathsf{stx\text{-}params} = \{T_0, m, \mathsf{b_{tgt}}, \tau, \gamma, s\}$, where T_0 is the initial target, m is the epoch length, $\mathsf{b_{tgt}}$ is the ideal time per block, τ is the dampening factor, and γ, s are bounds on the fluctuation of parties. The wrapper tracks the number of participants registered per round to facilitate the party bounds. We use $\mathcal{F}_{\mathtt{StX}}(\mathcal{P}, \Delta, p_H, p_A)$ and $\mathcal{F}_{\mathtt{StX}}$ interchangeably when the parameters are clear from context.

The authors of [24] show that the Bitcoin protocol UC-realizes $\mathcal{W}^{\mathsf{ledger\text{-}params}}(\bar{\mathcal{G}}_{\text{LEDGER}})$ in the $(\bar{\mathcal{G}}_{\text{CLOCK}}, \mathcal{W}(\mathcal{F}_{\text{RO}}), \mathcal{F}_{\text{DIFF}})$-hybrid world, under certain choices of parameters for $\mathsf{ledger\text{-}params}$. [24] additionally assumes that the sequence of parties is (γ, s)-respecting, but this work makes no such assumption.

2.4 Rational Protocol Design

The Rational Protocol Design (RPD) framework was introduced in [25], based on the ideal/real paradigm for cryptographic protocol security (cf. [12]), and it was originally applied to the problem of secure function evaluation [28]. At its core, RPD bridges the gap between the composable cryptographic analysis enabled by the UC framework, and the purely game-theoretic approach to modeling parties' incentives.

In a nutshell, RPD is formulated as a *meta-game* between two (potentially computationally unbounded) parties: a protocol designer $\mathtt{D}$, and an attacker $\mathtt{A}$. Security is defined via a two-party sequential zero-sum game with perfect information, called the *attack game*. The designer's move is to submit a protocol specification $\Pi \in \{0,1\}^*$ that implements an ideal functionality $\mathcal{F}$ capturing the task at hand. The attacker $\mathtt{A}$ responds by defining a (polynomial-time) adversary $\mathcal{A}$ to interact with the protocol Π. The game is scored by a *payoff* function v in which events are mapped from the joint view of the participants to a real value. While the formulation in [25] originally envisioned the utilities to be symmetric, follow-up work [3,6] extended the model to an asymmetrical utility function specification for the designer and the attacker, denoted by $v_{\mathtt{D}}, v_{\mathtt{A}}$, respectively.

In RPD, we are interested in the utility of the attacker or designer, based on a set of relevant events (e.g., adversary breaks security or correctness of the computation, adversary or designer buys a mining rig, etc.) with known costs/profits. Then, applying the RPD methodology can be boiled down to three steps as follows. First, the ideal functionality $\mathcal{F}$ is "weakened" to model events that do not happen in the usual ideal functionality—e.g. security breaches from potential attacks that an adversary might launch (as in the rational model they do not necessarily need to respect assumptions such as honest majority). Second, we define functions $v_{\mathtt{A}}$ and $v_{\mathtt{D}}$ that describe payoffs (sum of costs and profits) based on these events in the weakened ideal world. Finally, given a particular adversarial strategy $\mathcal{A}$ (chosen by attacker) and protocol Π (chosen by designer): (1) For the attacker's utility: For any $\mathcal{A}$, there may be many simulators that all simulate $\mathcal{A}$ in the weakened ideal world, each incurring different events and payoffs. We thus consider the utility of the 'best' simulator in the sense that it incurs the least attacker payoff but still simulates $\mathcal{A}$. (2)

For the designer's utility: Similarly, there are many possible environments for each protocol Π. We consider the designer utility under the 'worst-case' environment.

The utilities are tied directly to events that occur on the transcript of the ideal execution. To help facilitate this, the framework defines functionality $\langle\mathcal{F}\rangle$, understood to be a 'weakening' or 'defiled' version of $\mathcal{F}$, which incorporates the potential attacks that could be carried out on a protocol that implements $\mathcal{F}$ under less favorable assumptions. For example, in [25] it is allowed for the secure inputs of a secure function evaluation to be leaked, while in [3,6] the adversary is allowed to invoke a fork 'event' in the ideal world. Collectively, the model is referred to as a tuple $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ in the zero-sum formulation, and $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{D}}, v_{\mathtt{A}})$ in the non-zero-sum formulation. The attack game itself is referred to as $\mathsf{G}_{\mathcal{M}}$.

We first describe the attacker's utility. $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ is understood to be the ensemble that occurs after applying $v_{\mathtt{A}}$ to the views of $\langle\mathcal{F}\rangle, \mathcal{Z}$ in an experiment in the presence of simulator $\mathcal{S}$. We let $\mathcal{U}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})_{\mathtt{A}}$ denote the expected value of $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$, and let $u_{\mathtt{A}}(\Pi,\mathcal{A})$ denote the expected utility value of the attacker when protocol Π is played, and the attacker responds with $\mathcal{A}$.

We further define the class of all valid simulators for the adversary (with respect to a weakened functionality $\langle\mathcal{F}\rangle$ and a protocol Π) as $\mathcal{C}_{\mathcal{A}} = \{\mathcal{S} \in \mathsf{ITM} | \forall \mathcal{Z} : \text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}\}$. To ensure reducibility to the zero-sum game form, we additionally define $\mathcal{S}_{\mathtt{A}}, \mathcal{Z}_{\mathtt{A}}$ to be the class of simulators that can be used in the definition of $u_{\mathtt{A}}(\Pi,\mathcal{A})$ and the class of environments that will maximize that utility for the simulators in $\mathcal{S}_{\mathtt{A}}$ respectively. That is,

$$\mathcal{S}_{\mathtt{A}} = \left\{ \mathcal{S} \in \mathcal{C}_{\mathcal{A}} | \sup_{\mathcal{Z}\in\mathtt{ITM}} \{\mathcal{U}_{v_{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})\} = u_{\mathtt{A}}(\Pi,\mathcal{A}) \right\}$$

and

$$\mathcal{Z}_{\mathtt{A}} = \left\{ \mathcal{Z} \in \mathtt{ITM} \, | \exists \mathcal{S} \in \mathcal{S}_{\mathtt{A}} : \mathcal{U}_{v_{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z}) = u_{\mathtt{A}}(\Pi,\mathcal{A}) \right\}.$$

Finally, the attacker and designer's utilities are expressed as follows:

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

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

If $\mathcal{C}_{\mathcal{A}} = \emptyset$, then the utility for the attacker is ∞ (and $-\infty$ for the designer), as it implies that no simulator adequately simulates a protocol that emulates the relaxed functionality. At first glance, the inclusion of the supremum and infimum may seem strange, but is done to accurately model the roles that the environment and simulators play. For the attacker, we take the infimum of $\mathcal{C}_{\mathcal{A}}$ because the simulator should not reward the attacker for provoking events that have no effect on the execution of the protocol. Subsequently, the adversary should only gain utility when such events do affect the execution by forcing the simulator to provoke the event. For the environment, we take the supremum of the environments to capture the adversaries' ideal choice of environment: that which would maximize its utility.

Given a model $\mathcal{M}$, (weakened) functionality $\langle\mathcal{F}\rangle$, and protocol Π, the resulting utilities can be tied to specific security guarantees of interest. First, a *$\mathcal{M}$-maximizing adversary* is defined,

informally, as one that behaves so as to maximize the utility gained when attacking a protocol Π defined on $\langle\mathcal{F}\rangle$. Thus, any protocol sent by the designer must attempt to counter the $\mathcal{M}$-maximizing adversary. Such a protocol Π is called *attack-payoff optimal.* In more detail:

Definition 2 ($\mathcal{M}$-maximizing adversary [25]). *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ be an attack model and Π a protocol that realizes $\langle\mathcal{F}\rangle$. We say that an ITM $\mathcal{A}$ is an* $\mathcal{M}$-maximizing adversary *for Π if*

$$\mathcal{U}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) \overset{\mathsf{negl}}{\approx} \sup_{\mathcal{A}' \in \mathtt{ITM}} \mathcal{U}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}') =: \mathcal{U}^{\Pi,\langle\mathcal{F}\rangle}.$$

Definition 3 (Attack-payoff optimality [25]). *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ be an attack model and Π a protocol that realizes $\langle\mathcal{F}\rangle$. We say that protocol Π is* attack-payoff optimal *in $\mathcal{M}$ if for any other protocol Π', we have that*

$$\mathcal{U}^{\Pi,\langle\mathcal{F}\rangle} \overset{\mathsf{negl}}{\leq} \mathcal{U}^{\Pi',\langle\mathcal{F}\rangle}.$$

Ideally, we want to be able to give even stronger guarantees about the protocol Π in the rational setting. Attack-payoff optimality does not completely prevent the adversary from provoking the 'bad' events from the simulator. To that end, circumstances of interest are those in which Π is defined so that the simulator never invokes the 'bad' events at all. This notion is referred to as *attack-payoff security.*

Definition 4 (Attack-payoff security [25]). *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ be an attack model and Π be a protocol that realizes functionality $\langle\mathcal{F}\rangle$. Protocol Π is said to be* attack-payoff secure *in $\mathcal{M}$ if $\mathcal{U}^{\Pi,\langle\mathcal{F}\rangle} \overset{\mathsf{negl}}{\leq} \mathcal{U}^{\Phi^{\mathcal{F}},\langle\mathcal{F}\rangle}$, where $\Phi^{\mathcal{F}}$ is the dummy $\mathcal{F}$-hybrid protocol (i.e., the protocol that forwards all inputs to and outputs from the functionality $\mathcal{F}$).*

Additionally, we refer to an adversary $\mathcal{A}$ that maximizes message delays so that it can propagate its messages faster than other parties as being *front-running.* One effect of this is that it allows the adversary to ensure that if it finds a block at the same round that an honest party does, then the adversary's block will be propagated first. For honest miners, this is impossible; they cannot guarantee which block will make it into the blockchain (and indeed, the adversary can influence this).

Definition 5 (Front-running adversaries [3]). *The front-running adversarial strategy $\mathcal{A}_{\text{fr}}$ is specified as follows: Upon activation in round $r > 0$, $\mathcal{A}_{\text{fr}}$ activates in a round-robin fashion all its (passively) corrupted parties, say $p_1, \ldots, p_t$. When p_i generates a new message to be sent through the network, $\mathcal{A}_{\text{fr}}$ immediately delivers m to all its recipients. In addition, upon any activation, any message submitted to the network $\mathcal{F}_{\text{DIFF}}$ is maximally delayed. We denote $\mathbb{A}_{fr}$ the set of front-running adversaries.*

We are now ready to state the definition of incentive compatibility. Informally, a protocol Π is *incentive compatible* if both the designer and the attacker are incentivized not to deviate from the protocol. We note that such definition precludes the designer from obtaining a negative utility; any designer that obtains a negative utility would not be incentivized to mine, and thus would be enticed to deviate from the protocol specification. As a result, the pair $(\Pi, \mathcal{A})$ would no longer be a subgame-perfect equilibrium in the attack game under model $\mathcal{M}$, implying that the protocol is not incentive compatible.

Definition 6 (Subgame Perfect Equilibrium [3]). *Let $\vec{\Pi}$ and $\mathbb{A}$ be sets of possible strategies for the designer and the attacker, respectively. We say that a pair $(\Pi, \mathcal{A}) \in (\vec{\Pi}, \mathbb{A})$ is a* $(\vec{\Pi}, \mathbb{A})$-subgame perfect equilibrium *in the attack game defined by model $\mathcal{M}$, if it is a $(\mathsf{negl}(\kappa))$-subgame-perfect equilibrium in the restricted attack game where the set of all possible strategies of the designer (resp., the attacker) is $\vec{\Pi}$ (resp., $\mathbb{A}$).*

Definition 7 (Incentive Compatiblity in RPD [3]). *Let Π be a protocol and $\vec{\Pi}$ be a set of polynomial-time protocols that have access to the same hybrids as Π. We say that Π is* $\vec{\Pi}$-incentive compatible *($\vec{\Pi}$-IC for short) in the attack model $\mathcal{M}$ iff for some $\mathcal{A} \in \mathbb{A}_{fr}$, $(\Pi, \mathcal{A})$ is a (Π, ITM)-subgame-perfect equilibrium in the attack game defined by $\mathcal{M}$.*

2.5 RPD on PoW-based Blockchains, Static Difficulty

Baderstcher *et al.* [6] define the attacker's utility in the fixed difficulty setting as follows:

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) \triangleq \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathrm{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \mathtt{breward}(r) \cdot c \cdot \Pr[I^{\mathtt{A}}_{b,r}] \right.\right.$$
$$- \sum_{q \in \mathbb{N}} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}]$$
$$+ \mathtt{fpayoff} \cdot \Pr[K]$$
$$\left.\left. - \sum_{g \in \mathbb{N}} \mathtt{ccost}(g) \cdot \Pr[C^{\mathtt{A}}_{g}] \right\} \right\}. \tag{1}$$

While the utility function is somewhat similar to the variable difficulty setting we examine in this work, readers are invited to read [6] for a more comprehensive overview. In a nutshell, $\mathtt{breward}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is the block reward for successfully mining. $\mathtt{mcost}, \mathtt{fpayoff} \geq 0$ represent the cost of mining (i.e, renting or expending energy to mine), and the reward for successfully causing a fork. $\mathtt{ccost}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is defined as the cost of corrupting a party, and is restricted to be a non increasing function.

As observed in [6], without a carefully chosen precondition, the utility function cannot reasonably reflect real-world circumstances. Specifically, the authors show that it is possible to prove attack-payoff security (cf. Section 2.4) when it is profitable to passively mine, even when an adversary can profit from forking. This is a consequence of the fact that the adversary can accrue more profit from continually mining, so long as the run-time of the protocol is still polynomially bounded. Any profit that the adversary may accrue by causing a fork during the execution can be exceeded by simply running a little longer — and collecting the profit achieved from passively mining, which implies that an adversary might never be incentivized to cause a fork. Badertscher *et al.* [6] refer to a utility function that exhibits this behavior as having *unbounded incentives*.

Definition 8 (Bounded Incentives [6]). *We say that a utility function has* bounded incentives *if there exists a polynomial $h(\cdot)$ such that no $\mathcal{A}$ gains a higher utility than $h(\kappa)$, where κ is the security parameter.*

It is also shown in [6] that this artifact can be remedied by restricting the utility function to prevent profit from being gained indefinitely. More specifically, their solution assumes that $\texttt{breward}(r)$ is a non-increasing function such that, starting from a certain round onwards, its value decreases to the point where further passive mining is no longer profitable. In that case, we say that such a utility function satisfies *limited horizons*.

Definition 9 (Limited Horizons [6]). *Let* $\texttt{breward}(\cdot)$ *be a non-increasing function such that there exists a round* $r_{\text{flip}} \in \mathbb{N} : \forall r \geq r_{\text{flip}},\ p \cdot \texttt{CR} \cdot \texttt{breward}(r) < \texttt{mcost}$. *Then utility function* $u_{\texttt{D}}(\Pi, \mathcal{A})$ *(resp.,* $u_{\texttt{A}}(\Pi, \mathcal{A})$*) satisfies* limited horizons.

3 Utility Functions for the Dynamic Participation Setting

In this section, we show how to instantiate the RPD methodology with a utility that accounts for all major factors which dictate a miner's behavior (except for spite towards other miners). We start off in Section 3.1 by specifying the relaxed variable difficulty ledger without the restrictions on fluctuation imposed in [24]. Then, in Section 3.2, we give the detailed utility formulations for the designer and attacker, and relate our modeling choices to what happens in a real-world execution of a PoW-based cryptocurrency.

3.1 The Relaxed Ledger Functionality

The 'bad' events mentioned above are defined on a weakened version of the $\bar{\mathcal{G}}_{\text{LEDGER}}$ functionality described in [24], specifically defined for the dynamic-participation setting we are considering. The detailed specification of the functionality can be found in Appendix C. In a nutshell, the weakened ledger removes the restrictions on party growth; permits the simulator to create forks (see the 'fork' command from the adversary in red text); violates the Common Prefix guarantees by more freely setting the state slackness; and removes the guarantees about Chain Growth and Chain Quality[8]; again, the simulator used to define utility is the one incurring the *least* violations (i.e., it minimizes utility for the adversary).

$\bar{\mathcal{G}}_{\text{LEDGER}}$ is used in conjunction with other functionalities—the ledger 'wrapper' functionality, weakened state exchange, and weakened extend policies—which can be found in Appendix C. The ledger wrapper handles the adversary setting the communication delay, as well as party set management. The weakened state exchange handles the dynamic difficulty calculation, though it drops the restrictions on party growth. Finally, the weakened extend policies ensure validity of the next block state, while dropping the guarantees on Chain Quality and Chain Growth.

Variable Difficulty and Epochs. The ledger functionality and state exchange functionality in the variable difficulty setting dynamically recalculates the difficulty at the beginning of each epoch—the interval in rounds between target recalculations. Formally, a round r is in epoch i if $r_{e_i} \leq r < r_{e_{i+1}}$, where r_{e_i} is the first round in which the largest path state in the $\texttt{state-tree}$ of $\mathcal{W}^{\mathsf{weak\text{-}ledger\text{-}params}}(\mathcal{G}_{\text{WEAK-LEDGER}})$ satisfies the relation $i = \lfloor \frac{|\mathsf{state}|}{m} \rfloor$.

[8] Common Prefix, Chain Growth and Chain Quality are the basic PoW-based blockchain properties formulated in [26,27].

3.2 The Utility Functions

Following the careful redesign of the weakened ledger to remove (non-utility) assumptions/restrictions, we tackle the core of the RPD treatment: Specifying the utility functions.

We start with the utility of the designer D whose profits and costs are incurred collectively by non-adversarial (uncorrupted) miners. For brevity, we refer to such miners as D-miners (or honest parties) and to adversarial miners as A-miners. (As we discussed, an attacker's chosen adversary might be adaptive, i.e., "flip" D-miners to A-miners as the protocol evolves and depending on its view.[9])

The utility of designer D in the protocol execution is the sum of all utility accumulated in each round of the execution. In each such round r, the designer might:

- earn block rewards (e.g., via the coinbase transaction) for each block that a D-miner inserts into the blockchain ledger (as seen by other D-miners).[10] So if in round r the D-miners collectively insert b blocks, where the reward of each such block is defined by the function $\texttt{breward}(r)$[11], then his utility increases by $b \cdot \texttt{breward}(r) \cdot c$, where c corresponds to the price (in USD)[12] of the corresponding crypto asset;
- collect $\mu \cdot c$ utility where μ is the total fees for all blocks added to the ledger by D-miners in that round; and
- pay $\texttt{mcost}(r)$ for each hash-query that a D-miner makes (i.e., miners collectively making q queries incurs a cost $q \cdot \texttt{mcost}(r)$.

Symmetrically, in each round r, the attacker might:

- earn block rewards (e.g., via the coinbase transaction) for each block that a D-miner inserts into the blockchain ledger (as seen by other A-miners). So if in round r the A-miners collectively insert b blocks, then his utility increases by $b \cdot \texttt{breward}(r) \cdot c$;
- collect $\mu \cdot c$ utility where μ is the total fees for all blocks added to the ledger by A-miners in that round; and
- pay $\texttt{mcost}(r)$ for each hash-query that an A-miner makes (i.e., miners collectively making q queries incurs a cost $q \cdot \texttt{mcost}(r)$

By parameterizing the above quantities with r and inserting c into the equation (which will be implicitly associated with each round in the utility), we can already capture many of the moving parts, including some of the aspects of variable difficulty. However, dynamic participation requires further subtle modifications, which contributes to the unique universal character of our utility model. How can we capture the addition of new hashing rigs into the system, and how do we differentiate between ones operated by D-miners vs. A-miners? To answer this, one needs to draw intuition from reality: When a new miner is created, there is a cost associated to it; but generating it is not an adversarial act. To make it work for the adversary, the adversary needs to pay for it (by buying or renting it). As such, we can capture this dynamic via the following paradigm: New hashing devices that are created (i.e., new miners that are spawned) are charged (as a cost) to the designer; the cost of spawning each (unit of) hashing rig in round r is specified by a function

[9] Allowing to capture this "flipping" in such a seamless manner is one of the benefits of using RPD.

[10] This tricky event becomes clean when considering the translation to the ideal ledger setting, further highlighting the suitability of the RPD methodology.

[11] Incorporating $\texttt{breward}(r)$ as a function allows for capturing the halving effect or other deflationary choices of miner incentives.

[12] In this work, we use monetary utility functions and denominate all rewards in USD.

$\texttt{scost}(r)$, which might change in different rounds. As such, spawning s such units in round r incurs a cost of $s \cdot \texttt{scost}(r)$ to the designer. But what if the attacker is the one who commissioned this unit? To capture that, we use *adaptive corruption* of the corresponding miner as soon as it is inserted into the protocol. Such a corruption incurs a cost $\texttt{ccost}(t, r)$ to the attacker and an associated profit $\texttt{ccost}(t, r)$ to the designer, where t is the number of corrupted miners so far.

The above modeling allows for outmost flexibility: By separating $\texttt{scost}(r)$ from $\texttt{ccost}(t, r)$ and allowing for corruption (acquisition) of miners to happen at or after creation, it allows us to model all relevant scenarios: If the attacker is the one who effectively commissioned the hashing unit(s) in round r, then $\texttt{scost}(r) = \texttt{scost}(\cdot, r)$. On the other hand, if we want to capture that the unit was used by miners before being acquired by the adversary, we can model that as corrupting the unit strictly later than the round it was created; further, we can even capture the fact that acquiring it much later means acquiring it cheaper (e.g., depreciation by $\$\ell$) by setting $\texttt{scost}(r) - \texttt{scost}(\cdot, r') = \ell$. We believe that this flexibility can be instrumental in using our utility definition to capture complicated mining phenomena.

Motivated by the findings in [6], we also include a function $\texttt{D}(r) : \mathbb{N} \to [0, 1]$, which captures the notion of the time value of money. We refer to $\texttt{D}(r)$ as the *discounting function*. Intuitively, a person/entity tends to prefer some reward immediately, as opposed to receiving the same reward at a later point in time. Thus, $\texttt{D}(r)$ is modeled as a non-increasing function, starting with $\texttt{D}(0) = 1$. Recall that [6] confirmed a version of the folklore theorem in the blockchain RPD setting by proving that it is necessary to ensure that miners cannot simply mine indefinitely in order to prove attack-payoff security in a non-trivial setting. In their work, this notion is captured by requiring the block reward function $\texttt{breward}(r)$ to be a non-increasing function that eventually forces mining to be unprofitable. We enforce this restriction through a discounting function that allows for more explicit modeling of the incentive mechanisms. We define the *utility boundary round* $r_{end}(\kappa)$ for our discounting function, where for any round $r > r_{end}(\kappa)$, we have $\texttt{D}(r) = 0$. This captures the idea that an adversary's interest in running the protocol is time limited.

Finally, the attacker might receive payoff for forking the blockchain, which is modeled by $\texttt{fpayoff}(c, r)$, where c, r represent the price of the cryptocurrency in fiat currency and round, respectively. It is worth noting that we make no assumption that the fork is necessarily catastrophic to the blockchain. While in [3] a super-polynomial penalty was charged to the designer, we do not include this assumption here. The reason is that the entities that mostly absorb the impact of 51% attacks (other than what is encoded in the price drop which is already part of our utility) are typically the exchanges, rather the miners themselves (cf. [11]). This does not preclude a change in price as a result of a fork though; however, and perhaps surprisingly, some event studies (e.g., [39,41]) suggest that forking attacks typically **do not** lead to substantial price decreases.

We now list all the events that appear in the utility definitions.

- For $(s, r \in \mathbb{N}^2)$, the event, $S^{\mathcal{Z}}_{s,r}$ is the event where the environment $\mathcal{Z}$ spawns s new parties in round r.
- For $(b, r \in \mathbb{N}^2, c \in \mathbb{R}^+)$, the event $I^{\mathtt{A}}_{b,r,c}$ is the event of (the simulator) inserting b blocks into the ledger at round r and at price c, such that for each block b, there exists a D-miner p who's associated pointer $\texttt{pt}_p$ in $\mathcal{G}_{\text{WEAK-LEDGER}}$ is set such that the block is present in their state. Additionally, all of those blocks were previously queries to the random oracle by the adversary as a result of the extend policy being invoked. For each of the blocks, the block should appear in the transcript between the adversary and the simulator, and the contents of the block have

appeared on this transcript prior to the block's first appearance. $I^{\mathtt{D}}_{b,r,c}$ is similar, except that one of these two conditions does not hold.

- For $(r \in \mathbb{N}, c \in \mathbb{R}^+)$, the event $K_{r,c}$ is the event of (the simulator) invoking the FORK command at round r.
- For $(q, r \in \mathbb{N}^2)$, $W^{\mathtt{A}}_{q,r}$ (resp. $W^{\mathtt{D}}_{q,r}$) is the event of (the simulator simulating) q mining queries by the adversary (resp., D-miners) in round r and receives q query responses in the same round.
- For $(t, r \in \mathbb{N}^2)$, $C^{\mathtt{A}}_{t,r}$ is the event of (the simulator) corrupting g parties, at round r.
- For $(t, r \in \mathbb{N}^2)$, $C^{\mathtt{D}}_{t,r}$ is the event of (the simulator) corrupting/buying g parties at round r from the designer.
- Let $\mathcal{T}_{\mathcal{Z}}$ be the set that contains all transactions originally input from the environment. For $(r \in \mathbb{N}, c \in \mathbb{R}^+, \mu \in \mathbb{R}^+)$ the event $F^{\mathtt{A}}_{r,c,\mu}$ (resp., $F^{\mathtt{D}}_{r,c,\mu}$) is the event where the blocks that are inserted at round r provides μ total fees to the miner at price c, where μ is the sum of all transaction fees $f(\mathtt{tx})$ where $\mathtt{tx} \in \mathcal{T}_{\mathcal{Z}}$. The blocks are considered inserted when for block b, there exists a D-miner p who's associated pointer $\mathtt{pt}_p$ in $\mathcal{G}_{\text{WEAK-LEDGER}}$ is set such that the block is present in their state.

The RPD utility of the attacker $\mathtt{A}$ is thus defined as:

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) \triangleq \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathrm{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{r \in \mathbb{N}} \mathtt{D}(r) \left(\sum_{b \in \mathbb{N}, c \in \mathbb{Q}} b \cdot \mathtt{breward}(r) \cdot c \cdot \Pr[I^{\mathtt{A}}_{b,c,r}] \right. \right. \right.$$
$$+ \mathtt{fpayoff}(c, r) \cdot \Pr[K_{r,c}] - \sum_{q \in \mathbb{N}} q \cdot \mathtt{mcost}(r) \cdot \Pr[W^{\mathtt{A}}_{q,r}]$$
$$\left. \left. \left. - \sum_{t \in \mathbb{N}} \mathtt{ccost}(t, r) \cdot \Pr[C^{\mathtt{A}}_{t,r}] + \sum_{c \in \mathbb{Q}, \mu \in \mathbb{N}} \mu \cdot c \cdot \Pr[F^{\mathtt{A}}_{c,r,\mu}] \right) \right\} \right\}. \tag{2}$$

Regarding the nested summations, we note that the outer sum is taken over all rounds, and applies to both the costs and the rewards. This is because the time value of money applies to both sides: An actor prefers to obtain rewards as soon as possible, but also prefers to defer any costs to the future. The inner summations are taken over the various intervals. For example, the first inner term is taken over b, c, which are the number of blocks inserted, along with the price in fiat currency, respectively.

The utility for the designer $\mathtt{D}$ is defined analogously by swapping the sup, inf quantifiers, turning the corruption cost to profit, removing the forking reward, and adding the spawning cost. We refer to Appendix B for the detailed expression.

4 Incentive Compatibility Analysis

In this section, we establish the circumstances under which both the designer $\mathtt{D}$ and the attacker $\mathtt{A}$ (in fact, the miners following the respective strategies) are incentivized to participate in the protocol. We start off by discussing the nature of the discounting function, and why it is necessary to adequately model participants' incentives.

4.1 Discounting Captures Bounded Incentives

Recall the notion of 'limited horizons' (Def. 9). Unfortunately, limited horizons impose some restrictions that exclude certain utility functions that may be of interest. For example, except for

the $\texttt{breward}(r)$ function, the other parameters are all constant values. An obvious approach to fix this is to change the constant values into functions of time (i.e., round), and require that they are non-increasing functions. One problem is that this does not allow us to model changes in price, or the probability of successfully solving a PoW, as it would always be required to decrease. To address this, we abstract out the non-increasing function notion by introducing a *discounting function*. Intuitively, the discounting function captures the time value of assets (e.g., money) and prevents the adversary from mining indefinitely. Still, the remainder of the utility function has to be bounded by a polynomial, as a super-polynomial running time (in the security parameter) would reintroduce the same issue by essentially overpowering the depreciation. This would imply that future value provides more utility than current value, which contradicts the principle of time value of money.

Lemma 1 (Discounting captures bounded incentives). *Let* $\texttt{D}(r) = \texttt{negl}(r)$ *for* $r < r_{end}(\kappa)$, $\texttt{D}(r) = 0$ *for* $r \geq r_{end}(\kappa)$, *where* $r_{end}(\kappa) \in \mathbb{N}$ *is the utility boundary round, with functions*[13] $\texttt{breward}(r)$, $\mu(r)$, $\texttt{mcost}(r)$, $\texttt{scost}(g, r)$, $\texttt{ccost}(g, r) \in O\left(\frac{1}{\texttt{D}(r)}\right)$. *Then* $u_{\texttt{D}}$ *(resp.,* $u_{\texttt{A}}$*) satisfies bounded incentives.*[14]

Proof. Refer to Equations 1 and 2. The proof follows directly from the fact that if $\texttt{D}(r)$ is negligible, the remaining portion of the utility is multiplied by a negligible function. Because of our assumption on $\texttt{breward}(r)$, $\mu(r)$, $\texttt{mcost}(r)$, $\texttt{scost}(g, r)$, $\texttt{ccost}(r)$, then the total utility accrued before $r_{end}(\kappa)$ is also negligible. As no utility can be gained after round $r_{end}(\kappa)$, this directly implies that $\mathcal{A}$ cannot find a better payoff than $h(\kappa)$, since $\mathcal{A}$ cannot simply keep mining longer, which implies that the utility obeys bounded incentives (Definition 8). The same argument applies for the designer's utility.

We note that this is a more general case compared to the utility function definition that obeys bounded incentives in [6], with two inconsequential differences. There, $p, c(r), \texttt{breward}(r), \texttt{mcost}$ are all constants while no fees are considered, and thus satisfy the requirement that $\texttt{breward}(\cdot)$ is a non-increasing function such that $\exists r_{end}(\kappa) \in \mathbb{N} : \ \forall r \geq r_{end}(\kappa), \ p \cdot \texttt{CR} \cdot \texttt{breward}(r) < \texttt{mcost}$. As a result, their $\texttt{breward}(r)$ must be decreasing, while here we capture this by abstracting it to the discounting function $\texttt{D}(r)$. Additionally, we allow the rest of the utility after round $r_{end}(\kappa)$ to still be greater than 0, whereas in [6], the utility function must be less than 0. The end result is the same: The adversary is no longer incentivized to passively mine after round $r_{end}(\kappa)$.

□

4.2 When is Mining Profitable?

We note an important simplifying assumption, namely, that the designer (in fact, the miners faithfully executing the code) does not withdraw its computing power from the network during rounds in which negative utility would be accrued, unless it does so for the entire duration of the execution. This is to appropriately model the fact that miners typically do not switch all of their machines off as soon as they are not profitable, and will sometimes wait to 'weather' out negative conditions. In the static difficulty case [3,6], this is not an issue, as $p(r)$, $\texttt{CR}(r)$, $\texttt{breward}(r)$, $\texttt{mcost}(r)$ are defined in such a way that precludes the possibility of fluctuations that lead to a change in profitability

[13] We require the costs to be $O\left(\frac{1}{\texttt{D}(r)}\right)$ to allow for negative costs (i.e., subsidies).

[14] We remark that, in reality, this discounting function is most often modeled as an exponentially decaying function (and thus is negligible in r). While other discounting functions do exist (e.g., hyperbolic), in this work we use the former.

before round $r_{end}(\kappa)$. In the dynamic participation case, which we are considering, we can easily have fluctuations in the price or costs that do change whether a given round is profitable to mine on.

At a technical level, not including this restriction can lead to some undesirable conclusions. For example, consider a definition of the utility function with parameters $p(r)$, $\texttt{CR}(r)$, $\texttt{breward}(r)$, $\texttt{mcost}(r)$, etc, in which mining was not profitable for all rounds r where $\texttt{D}(r) > 0$, except for some round r_i. We would conclude that the designer would have positive utility to participate in the protocol, simply because they could mine for that one round, but abstain from mining throughout the execution. Intuitively, such a circumstance would not be palatable to the designer (or any miner), as they likely would find it too cumbersome to mine only for a single round. In reality, switching off all mining machinery for short durations incurs adjustment costs, which may prove difficult to model accurately. By contrast, our approach allows us to reason about utility over a broader period of time.

On the necessity of sufficiently small block generation rates. In the subsequent theorems for this section, we assume that for every round r, we have that $p_r n_r \ll 1$. Of course, this condition implies that in every round, the expected number of blocks generated is less than 1. (We note that in contrast to [26], here we are considering block generation rate per round, instead of per epoch.) The intuition is that if too many blocks are generated in a period of time, consensus among the protocol participants becomes intractable, because new information is being sent to parties so quickly that forks become inevitable. In this section, we formally substantiate that intuition: Assuming that the number of parties is polynomial in the security parameter and the initial success probability p_0 is sufficiently low, then the number of blocks created per-round is $o(1)$. First, a technical lemma:

Lemma 2. *Consider two executions of versions of the attack game V_1 and V_2, one where in V_1 each epoch consists of r rounds and in V_2 in which every epoch consists of $q(k)r$ rounds, where q is a polynomial in the security parameter k. Then for any strategy profile $(\Pi, \mathcal{A})$:*

- *$(\Pi, \mathcal{A}')$ (weakly) dominates $(\Pi, \mathcal{A})$ (for the attacker) for some $\mathcal{A}'$ in V_1 iff $(\Pi, \mathcal{A}')$ (weakly) dominates $(\Pi, \mathcal{A})$ (for the attacker) in V_2;*
- *$(\Pi', \mathcal{A})$ (weakly) dominates $(\Pi, \mathcal{A})$ (for the designer) for some Π' in V_1 iff $(\Pi, \mathcal{A}')$ (weakly) dominates $(\Pi, \mathcal{A})$ (for the attacker) in V_2.*

Next, we present this key insight.

Observation 1 *Let D be a discounting function for which there is a function $r_{end}(\cdot)$ such that $\forall r > r_{end}(\kappa), D(r) = 0$. Suppose there is a polynomial $\mathsf{poly}(\kappa)$ such that for all rounds r the total number of parties $n_r \leq \mathsf{poly}(\kappa)$, and the initial probability of successfully creating a block with one hash p_0 is such that $n_0 p_0 = 1/f(\kappa)$ where $f(\kappa) = \omega(r_{end}(\kappa))$. Then, for the variable-difficulty protocol Π, utility Eq. 1, any adversary $\mathcal{A}$, and any constant $c > 0$ there exists a $\kappa > 0$ such that*

$$u(\Pi, \mathcal{A}) = u^*(\Pi, \mathcal{A})$$

where u^* is the utility function u, but ignoring costs/rewards in all rounds where $n_r p_r \geq c$.

To see the that above observation holds: Initially $p_0 = 1/(\mathsf{poly}(\kappa) f(\kappa))$. We need $p_r = c/\mathsf{poly}(\kappa)$ in order for $n_r p_r \geq c$. To see how many rounds it takes for p_r to reach this number, we recall that in each epoch, the difficulty can only be adjusted up to a factor of τ, which requires the speed of block creation over the epoch to change by a factor of τ. In other words, to increase the probability

p_0 into τp_0, we need to slow down block creation so it takes τ times the original time for an epoch. Similarly, it takes τ^2 times the original time for an epoch to take the probability from τp_0 to $\tau^2 p_0$, and so on. Thus, it takes at least $c \cdot f(\kappa)$ rounds for $n_r p_r \geq c$. But we assume $f(\kappa) = \omega(r_{end}(\kappa))$, so we can always find a κ such that $c \cdot f(\kappa) > r_{end}(\kappa)$, i.e., after reaching a round r such that $n_r p_r \geq c$, for all the rounds after, $r' > r$, the discounting function $D(r') = 0$. Thus we can ignore these rounds in the utility function.

More formally, consider an environment $\mathcal{Z}$. Recall that in RPD, the attacker's utility $u_{\mathtt{A}}$ is determined from the supremum of the set of ITMs. Thus, the environment that maximizes the utility for the adversary is the one that is considered when scoring. For $p_r n_r > 1$ to be true, either the environment $\mathcal{Z}$ needs to spawn $1/p_r - n_r$ parties, or the designer needs to select the initial expected block generation rate to be greater than 1. The latter case is a bad strategy: The designer's utility would be lower, as they would lose substantial amounts of work due to forking. In the former case, consider an environment $\mathcal{Z}'$, which attempts to increase the block generation rate by spawning an excess of parties. The adversary $\mathcal{A}$ incurs a substantial cost to corrupt these parties. If $\mathcal{A}$ wants to increase the block generation rate, $\mathcal{A}$ must not mine with his computing power. This incurs a great opportunity cost on the adversary, which is exacerbated by the discounting function. Additionally, if p_r is too low to begin with, the number of parties necessary to corrupt can be so large that it is impossible to spawn them.

While RPD does not directly place a limit on the number of parties, this can be enforced by the utility function: To prevent too many parties from spawning, we can either set $\texttt{ccost}(t, r)$ higher than any achievable utility, or define $\Pr[C^{\mathtt{A}}_{t,r}] = 0$ for $t > t'$ for some $t' \in \mathcal{R}$.

We note in the cryptographic setting the story is similar: high block-generation rate hinders cryptographic security in the static difficulty setting even when no corruptions occur [26].

Observation 2 *When $p_r n_r > 1$ for all r, then typical executions (as defined in [26]) do not occur with overwhelming probability.*

Empirically, we find that during the periods examined in Section 5, $p_r n_r \ll 1$ (Recall that the target block generation rates for Ethereum Classic and Bitcoin are 13 and 600 seconds per block respectively.). As of the time of writing, we are unaware of any instance in either Ethereum Classic or Bitcoin where $p_r n_r > 1$.

Next, we show a case where the utility of the designer must be positive; it immediately follows that a negative utility precludes incentive compatibility. In a nutshell, we show that if the mining cost is too great compared to the rewards for mining, then the change in difficulty does not occur fast enough to allow the honest miner to accrue enough utility.

Lemma 3. *Let $r_{end}(\kappa)$ be an round in epoch j' such that for all rounds after $r_{end}(\kappa)$, the total utility gained is negligible in κ (this is enforced by the deprecation function). Let $p_r n_r < 1$ for all r. Then, if $\sum_{r=r_{e_0}}^{r_{end}(\kappa)} D(r) \cdot (p_r - p_r^2 n_r \Delta) \cdot \texttt{CR}(r) \cdot (\texttt{breward}(r) + \mu(r)) - \texttt{mcost}(r) \cdot (n_r - t_r) > 0$, the designer utility is positive.*

Proof. We begin by discussing the case where the utility must be positive. First, in round r, all of the n_r parties will run the protocol under the single-query-per-round assumption. With t_r parties controlled by the adversary, the probability of an honest miner inserting a block into the chain as described in event $B^{\mathtt{D}}_{b,r,c}$ is lower bounded $p_r \cdot (1 - p_r)^{(n_r)\Delta}$. This is because in the worst case, we assume that the production of any block by any other party within the network delay will always cause the mined block to not be inserted into the longest chain. Because $p_r n_r < 1$ for all r, the

expected value of blocks generated within a round is no more than one, and thus the longest chain is expected to be extended by no more than one block. Thus, the designer utility is as follows:

$$\sum_{r=r_{e_0}}^{r_{end}(\kappa)} \mathtt{D}(r) \cdot p_r \cdot (1-p_r)^{n_r \Delta} \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r) \cdot (n_r - t_r) \tag{3}$$

When this expression is positive, then the utility is also positive. We can take the above expression and get the following:

$$\sum_{r=r_{e_0}}^{r_{end}(\kappa)} \mathtt{D}(r) \cdot p_r \cdot (1-p_r)^{n_r \Delta} \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r) \cdot (n_r - t_r) \tag{4}$$

$$\geq \sum_{r=r_{e_0}}^{r_{end}(\kappa)} \mathtt{D}(r) \cdot p_r \cdot (1 - p_r n_r \Delta) \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r) \cdot (n_r - t_r) \tag{5}$$

$$= \sum_{r=r_{e_0}}^{r_{end}(\kappa)} \mathtt{D}(r) \cdot (p_r - p_r^2 n_r \Delta) \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r) \cdot (n_r - t_r) \tag{6}$$

Therefore, when this expression is positive, the utility of the designer must also be positive.

□

4.3 Incentive Compatibility of Bitcoin

We now turn our attention to incentive compatibility (Definitions 6 and 7). Intuitively, this condition implies not only that the adversary has no incentive to deviate from the protocol, but also that honest participants are incentivized to continue mining throughout the execution of the protocol. This, in turn, implies a stable system: Adversaries gain nothing from cheating, while everyone continues to participate, allowing new blocks to be added to the chain.

Intuitively, this condition implies that the adversary does not have an incentive to deviate from the protocol, and the honest participants have incentive to continue mining, given the adversary does not deviate. This in turn implies a stable system; adversaries don't gain from trying to cheat, while everyone still participates, which allows for new blocks to be entered into the chain.

Attack-payoff security. In order to prove incentive compatibility (Def. 7), it is necessary first to show that the protocol in question is 'attack-payoff secure' (Def. 4)—i.e., it is an equilibrium strategy for the attacker to be passive (but front-running). We do so by showing it is 'not worth it' for an adversary to attempt to create a deep fork, due to the cost of rigs and how long a fork must last in order to reverse transactions. Next, we describe the steps at a high level .

The idea roughly follows the attack-payoff security treatment from [6], adapting for variable-difficulty blockchains: Given a cut-off parameter cutOff (how many blocks deep a transaction must be to be confirmed), any adversary with a limited *budget* (number of hashes above majority hashing power, spent over some period of time) is only able to break consistency (i.e., maintain a fork of depth at least cutOff) for a limited number of rounds [4]. The larger cutOff is, the more adversary budget is required to break consistency. On the other hand, the larger B is, the more costly it can be for the adversary—spending a higher budget implies that more rigs need to be corrupted/bought, and potentially mining at a loss.

The above relationship between the cutOff parameter and budget required to break consistency at that parameter means one can set cutOff high enough so that the budget required is unprofitably

high. That is, make breaking consistency (forking) difficult enough so that it is unprofitable to attempt such an attack.

To make any meaningful guarantees regarding adversarial behavior, it is a necessary prerequisite to place lower and upper bounds ($\mathbf{T}_{\mathsf{lb}}, \mathbf{T}_{\mathsf{ub}}$ respectively) on the hashing power. In doing so, we can provide an upper bound on the probability of successfully solving a proof of work. To this end, we adopt the definition of an adversary bounded in this manner that was introduced in [4,6].

Definition 10 ($(\theta_{\mathsf{pow}}, \epsilon, \mathbf{T}_{\mathsf{lb}}, \mathbf{T}_{\mathsf{ub}}, B)$-adversary [4]). *Let $\theta_{pow}, \epsilon \in (0,1), \mathbf{T}_{lb}, \mathbf{T}_{ub} \in \mathbb{N}$. A ($\theta_{pow}$, $\epsilon, \mathbf{T}_{lb}, \mathbf{T}_{ub}, B$)-adversary is an adversary satisfying the following: At every round i, let n_a^i and n_h^i be the mining queries made by the corrupt and honest parties in this round. Then:*

1. *For all i, $\mathbf{T}_{lb} \leq n_a^i + n_h^i \leq \mathbf{T}_{ub}$*
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 that the adversary attacks between rounds $a < b$ if $B_i = 0$ for any $i < a$ or $i > b$ (i.e. he spends all of 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 $r_1 < r_2 < \cdots < r_t$ where $\sum_i B_i = B$ and the probability of success when difficulty is computed based on total hashing power $\mathbf{T}_{ub}$.

Lemma 4. *Let $0 < \epsilon < 1$. Let $R_{high} = \{r \in \mathbb{R} | (p_r \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r))) - \mathtt{mcost}(r)) > 0\}$ and $R_{low} = \{r \in \mathbb{R} | (p_r \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r))) - \mathtt{mcost}(r)) \leq 0\}$, and $\bar{n}_a := \frac{(1-\epsilon)\cdot\theta_{pow}\cdot\mathbf{T}_{ub}}{1+(1-\epsilon)\cdot\theta_{pow}}$. Further, let $u^{\mathtt{ub}}(B,t)$ be the utility that an adversary $\mathcal{A}$ with a budget of at least $B > 0$, and who spends it over $t \geq \frac{B}{\mathbf{T}_{ub} - \bar{n}_a}$:*

$$u^{\mathtt{ub}}(B,t) := \tag{7}$$

$$\sum_{r \in R_{high}} \mathtt{D}(r)(\mathbf{T}_{ub} \cdot p_r \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r))$$

$$+ \sum_{r \in R_{low}} \mathtt{D}(r)((\bar{n}+1) \cdot p_r \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) \tag{8}$$

$$- \mathtt{mcost}(r)) - \mathtt{ccost}(\bar{n_a} + \frac{B}{t}) + \mathtt{fpayoff} \tag{9}$$

Then, $u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u^{\mathtt{ub}}(B,t)$.

Proof. The proof is almost identical to Lemma 8 in [6], so we sketch out the key differences. The biggest difference occurs in the summation of the rounds. In [6], the utility of the adversary is always decreasing, so the adversary will always want to mine as much as possible as early as possible. In this case, the constraints on $u_{\mathtt{A}}(\Pi, \mathcal{A})$ do not require this. On round $r \in R_{\mathsf{high}}$, the adversary gains value by definition, and thus will mine with all of it's computing power. On round $r \in R_{\mathsf{low}}$, the adversary does not gain value, so it tries to mine with as little computing power as it can to minimize losses. Similarly to [6], we assume that the fork payoff is always received when deriving an upper bound on utility. □

In the static difficulty setting of [6], the authors precisely define a value $R(B,t,\ell)$, which is understood to be the maximum number of consecutive rounds that ℓ-consistency can be violated

(i.e., a ℓ-block-deep fork can be created with non-negligible probability), given an adversarial budget of B, spent over t rounds, with consistency parameter ℓ. This value is precisely computed in [4].

This computation does not directly carry over in the variable difficulty setting. However, it is easy to see that $R(B, t, \ell)$ must exist even in the variable difficulty setting. An adversary that only has B budget can only cause a fork to occur for some finite amount of time, after which the honest parties will have complete control of new blocks. After this point, the honest parties will eventually agree on their common prefix, and thus consistency will be restored after some $R(B, t, \ell)$ rounds. We leave the precise computation of $R(B, t, \ell)$ in the variable difficulty setting as future work.

Next, we show that attack payoff security can be guaranteed after expanding the cutoff parameter cutOff. We use p_{max} to denote the maximum p_r during the $R(B, t, \ell)$ rounds.

Lemma 5. *Let $\varphi > 0$ and ℓ a cut-off parameter for which the $\Pi^{\mathcal{D}}$ protocol is secure when honest majority assumption holds. Let cutOff be the actual cutoff parameter for $\Pi^{\mathcal{D}}$ in attack model $\mathcal{M} = (\bar{\mathcal{G}}_{\text{LEDGER}}, \mathcal{G}_{\text{WEAK-LEDGER}}, v_{\mathtt{D}}, v_{\mathtt{A}})$. Let the $\mathtt{D}(r)$ associated with the adversarial utility be defined as in 1. Let $\mathbf{T}_{lb}, \mathbf{T}_{ub} > 0$. Let $R(B, t, \ell)$ be the maximum number of consecutive rounds where ℓ-consistency fails, and p_{max} be the maximum probability between $r = 0$ and r_{end}. Let $u'_{\mathtt{A}}(\Pi, \mathcal{A})$ refer to some lower bound on utility $u_{\mathtt{A}}(\Pi, \mathcal{A})$ for a front-running but honest adversary. For any $(\theta_{pow}, \epsilon, \mathbf{T}_{lb}, \mathbf{T}_{ub}, B)$-adversary, if it holds that:*

$$\mathsf{cutOff} \geq \ell + \max_{(B,t) \mid u^{\mathtt{ub}}(B,t) > u'_{\mathtt{A}}(\Pi,\mathcal{A})} (1+\varphi)\mathbf{T}_{\mathsf{ub}} p_{\mathsf{max}} \cdot R(B, t, \ell),$$

then the Bitcoin protocol is attack-payoff secure.

Proof. The ℓ component ensures the cutoff parameter must wait at least as long as the consistency parameter that guarantee that consistency does not fail in the honest-majority case.

Within a given $R(B, t, \ell)$ rounds, a maximal expected value of $\mathbf{T}_{\mathsf{ub}} \cdot p_{\mathsf{max}}$ blocks will be created, as $\mathbf{T}_{\mathsf{ub}}$ queries can be made, with a success chance of no more than p_{max}. By a standard Chernoff bound (applicable since p_{max} is a constant that does not depend on the actions of the adversary), we can bound the number of expected blocks that are generated during $R(B, t, \ell)$ rounds.

By setting B, t to the value that maximizes the number of rounds that an adversary can profit from, we ensure that no adversary can profit more from causing a fork than they can by mining honestly, since they cannot create a fork larger than ℓ blocks (which would violate consistency). There is a maximum (B,t) because of $\mathtt{D}(r)$; after the utility boundary round r', no further utility can be gained, and thus $u^{\mathtt{ub}}(B, t) > u'_{\mathtt{A}}(\Pi, \mathcal{A})$ cannot be true for rounds after r', because the utilities in both cases would 0.

Since all payoff events in the real world can be matched by an adversary that invokes similar events in the ideal world, we imply attack payoff security. □

Incentive Compatibility Next, we show three sufficient conditions for achieving incentive compatibility. The first condition states that it is profitable for the designer to mine. The second condition states that if it is profitable to mine with any number of parties, it is more profitable to mine with as many machines as possible. The third condition states that the protocol is attack-payoff secure (see discussion above).

Definition 11 (Alternative Protocol Set [3]). *Let $\vec{\Pi}_{\mathsf{isvalidchain}_H}$ be the set of alternative protocols compatible with the Bitcoin network model in the $(\mathcal{W}^q(\mathcal{F}_{\text{RO}}^{\kappa}), \bar{\mathcal{G}}_{\text{CLOCK}}, \mathcal{F}_{\text{DIFF}})$-hybrid model that obey two restrictions. First, there is no chain $\mathcal{C}$ within the real world transcript where* isvalidchain

$(\mathcal{C}) = 0$, *unless it was output to the network by a corrupted protocol instance. Second, upon input* (READ, sid), *the protocol returns* $(\text{READ}, sid, \vec{st}^{\lceil T})$, *with* $\vec{st}$ *referring to the state of the longest chain in* $\mathcal{C}$ *and* T *to be the consistency parameter.*

In a nutshell, $\vec{\Pi}_{\mathsf{isvalidchain}_H}$ describes the strategy space of the designer. Restricting the move set is necessary, as if we allowed the designer to play any arbitrary protocol, we would be comparing completely different protocols that might not be related.

Theorem 1 (Incentive Compatibility). *Let* $\varphi, \epsilon \in (0,1)$*; let* $r_{end}(\kappa)$ *be the utility boundary round;* cutOff *be the cutoff for* $\Pi^{\not\Bbbk}$*;* $\mathbf{T}_{lb}, \mathbf{T}_{ub} > 0$ *be the lower and upper bound on total hashing power between rounds* 0 *and* $r_{end}(\kappa)$*, and* p_{max} *be the maximum probability during those rounds;* $R(B,t,\ell)$ *be the maximum number of consecutive rounds where consistency does not hold, and* $u'_{\mathtt{A}}(\Pi^{\not\Bbbk}, \mathcal{A})$ *refer to some lower bound on utility* $u_{\mathtt{A}}(\Pi^{\not\Bbbk}, \mathcal{A})$ *for a front-running adversary that follows* $\Pi^{\not\Bbbk}$*.* $u^{\mathtt{ub}}(B,t)$ *refers to the upper-bound utility of an* $(\theta_{pow}, \epsilon, \mathbf{T}_{lb}, \mathbf{T}_{ub}, B)$*-adversary (Def. 10). Then, if the following conditions hold:*

1. $\sum_{r=r_0}^{r_{end}(\kappa)} D(r) \cdot (p_r - p_r^2 n_r \Delta) \cdot CR(r) \cdot (breward(r) + \mu(r)) - mcost(r) \cdot (n_r - t_r) > 0$*;*
2. $\sum_{r=0}^{r_{end}(\kappa)} D(r) \cdot p_r(1-p_r)^{n_r} \cdot CR(r) \cdot (breward(r) + \mu(r)) - mcost(r))) > 0$*; and*
3. $\mathsf{cutOff} \geq \ell + \max_{(B,t)|u^{\mathtt{ub}}(B,t) > u'_{\mathtt{A}}(\Pi,\mathcal{A})}(1+\varphi)\mathbf{T}_{\mathsf{ub}} p_{\mathsf{max}} \cdot R(B,t,\ell)$*,*

protocol $\Pi^{\not\Bbbk}$ *is* $\vec{\Pi}_{\mathsf{isvalidchain}_H}$*-incentive compatible in the* $(\mathcal{F}_{\text{RO}}, \bar{\mathcal{G}}_{\text{CLOCK}}, \mathcal{F}_{\text{DIFF}})$*-hybrid model.*

Proof. Our proof follows a similar approach to that of [3]. We begin by showing an important fact about the utility. In a nutshell, the honest parties are only incentivized to mine with all of their computing power $n_r - t_r$ if it provides the optimal payoff.

Lemma 6. *When Condition 2 holds, the honest miners obtain optimal utility over the runtime by mining with all their available computing power.*

Proof. Let $Q^h = \{Q_1^h, \ldots, Q_{r_{end}(\kappa)}^h\}$ be a set of random variables associated with the number of queries that the honest parties make toward the random oracle, where each entry Q_r^q refers specifically to the number of queries at round r. For any given round r, Q_r^h can have a distribution in $\{0, \ldots, n(r) - t(r)\}$. It follows that the expected utility for the designer is the following:

$$\sum_{r=0}^{r_{end}(\kappa)} \Pr[Q_r^h = q] \cdot \mathtt{D}(r) \cdot (p_r - p_r^2)^{q\Delta} \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r) \cdot q) \tag{10}$$

Now, it follows that if $q+1$ queries provide greater than or equal utility than q queries, then the honest parties will maximize the utility by committing all of their computing power. The base case is trivial; if $q = 0$, then the utility is maximally bounded by 0, as no mining is occurring. We now show the more general case:

$$\sum_{r=0}^{r_{end}(\kappa)} \Pr[Q_r^h = q] \cdot \mathtt{D}(r) \cdot (1 - (1-p_r)^{(q+1)\Delta}) \cdot \mathtt{CR}(r) \tag{11}$$
$$\cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r) \cdot (q+1))$$
$$- \sum_{r=0}^{r_{end}(\kappa)} \Pr[Q_r^h = q] \cdot \mathtt{D}(r) \cdot (1 - (1-p_r)^{(q)\Delta}) \cdot \mathtt{CR}(r) \tag{12}$$
$$\cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r) \cdot$$
$$= \sum_{r=0}^{r_{end}(\kappa)} \Pr[Q_r^h = q] \cdot \mathtt{D}(r) \cdot p(1-p)^q \cdot \mathtt{CR}(r) \cdot (\mathtt{breward}(r) + \mu(r)) - \mathtt{mcost}(r))).$$

We are interested in when $\Pr[Q_r^h = n(r) - t(r)]$ for all $r \in [1, \dots r_{end}(\kappa)]$ when the resultant payoff is positive. This occurs when Condition 2 is met. □

By Theorem 2, there exists a realization of the state exchange functionality in the $(\mathcal{W}^q(\mathcal{F}_{\text{RO}}^\kappa), \bar{\mathcal{G}}_{\text{CLOCK}}, \mathcal{F}_{\text{DIFF}})$-hybrid world. Thus, it suffices to analyze this setting. First, we observe that Π doesn't send any messages over functionalities other than $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathbf{STX}})$. The state exchange functionality is only realizable with $\mathcal{F}_{\text{DIFF}}$. An alternative protocol Π' that didn't use $\mathcal{F}_{\text{DIFF}}$ would send messages over an insecure network, as $\mathcal{F}_{\text{DIFF}}$ is a stronger resource. Thus, any functionality that can be realized by Π, can also be realized by Π', which uses $\mathcal{F}_{\text{DIFF}}$.

We can also show that for all valid chains stored in $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathbf{STX}})$, no block will point to more than one block except for with negligible probability. More formally, for a chain $\mathcal{C} = \mathbf{B}_1, \dots, \mathbf{B}_n$, let $\mathcal{C}^{\mathsf{fstx}} = \{\mathcal{C}_1, \dots, \mathcal{C}_i\}$ refer to the set of all chains in $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathbf{STX}})$, and for a new block $\mathbf{B} = \langle \mathsf{pt}, \mathtt{st}, \mathsf{n}, T, t \rangle$. Then we have that $\Pr[H[\mathbf{B}_{i-1}] = p_i = p] < \mathsf{negl}(\kappa)$ for any p_i in any block in $\mathcal{C}_i^{\mathsf{fstx}}$, unless $\mathbf{B}_i = \mathbf{B}$.

This occurs because it inherently requires that a collision must have happened. In $\mathcal{W}^q(\mathcal{F}_{\text{RO}}^\kappa)$, we assume that all new queries to the functionality are uniformly sampled over the domain, which means that a single two queries has a probability of colliding with each other at $\frac{1}{2^\kappa}$. Taking a union bound over all of the possible chains, the probability becomes $\frac{poly(\kappa)}{2^\kappa}$, which is a negligible function. This means that two honest parties that find a solution to the PoW cannot, except with negligible probability, extend the same chain.

Because the adversary is front running, we also have that the output to $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathbf{STX}})$ in a given round r of an uncorrupted protocol instance of Π is not dependent of any other uncorrupted protocol instances of the same round, because a front running adversary will always delay honest party messages by a round, so honest parties at r cannot have their message delivered before round $r+1$.

Finally, the honest parties will extend the longest chain that they have access to at round r. First, the honest parties will receive any new found blocks from the adversary that were mined in round r, since the adversary is front running. Additionally, the chains received by honest parties will be the longest chain that any honest party has access to. After the adversary mines, we denote ℓ_r as the chain which is the longest after the adversary mines (but before the honest parties mine). If the honest parties actually extend the chain ℓ_r by providing a valid proof of work, it must have submitted to $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathbf{STX}})$. We showed above that for all valid chains stored in $\mathcal{W}^{\mathsf{stx\text{-}params}}$

($\mathcal{F}_{\mathbf{STX}}$), no block will point to more than one block, which means that when the honest parties succeed in solving a PoW, they extend the chain (and obtain the benefits), except with negligible probability. By Lemma 6 and Observation 3, their utility is maximized when the utility is positive and when they mine with all of their available computing power. Formally, we have that:

- $\forall \Pi \in \vec{\Pi}_{\mathsf{isvalidchain}_H}$, $u_{\mathtt{D}}(\Pi, \mathcal{A}) \leq u_{\mathtt{D}}(\Pi^{\Pi}, \mathcal{A}_{\mathsf{fr}})$,
- $\forall \mathcal{A}$, $u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u_{\mathtt{D}}(\Pi^{\Pi}, \mathcal{A}_{\mathsf{fr}})$ for some $\mathcal{A}_{\mathsf{fr}} \in \mathbb{A}_{fr}$.

From Condition 3, we can directly apply Lemma 5, which allows us to conclude that the protocol is attack payoff secure. Since attack payoff security is a necessary prerequisite, this leads directly to the main theorem statement: the protocol is incentive compatible with respect to (reasonable) alternative protocols of the class $\vec{\Pi}_{\mathsf{isvalidchain}_H}$. This concludes the proof. □

5 Empirical Validation

So far, our analysis has been mostly theoretical following the pathway that has been common in the relevant literature: Propose a game and associate utility that appears as a natural abstraction of the financial incentives, and devise a security/stability analysis of the blockchain under this utility. As important as this methodology is, we believe it can be usefully complemented by a common validation from financial economics: Demonstrating that the proposed utility is not just theoretically intuitive but can also be validated through empirical observations of the system's behavior. We view this empirical analysis as an integral part of our treatment, which sets our model apart from existing, highly parameterized, theoretical analyses.

More concretely, we explore several real-world settings involving Ethereum Classic and Bitcoin. Formally, we use $\mathcal{W}^{\mathsf{weak\text{-}ledger\text{-}params}}(\mathcal{G}_{\text{WEAK-LEDGER}})$ as our weakened functionality to which we apply the RPD methodology. While this functionality formally models the Bitcoin protocol, it can be modified with relatively straightforward (e.g. epoch length, difficulty update mechanism) changes to reflect the other protocols mentioned. For all of our results, our analysis incorporates a changing value for p. We model the deprecation using the function $\mathtt{D}(r) = \left(\frac{1}{1+\frac{.03}{365}}\right)^{\lfloor r/b \rfloor}$, where b is understood to be the number of expected blocks per day. This assumes a 3% interest rate per year with daily compounding, which is close to the inflation target of most central banks.

Measuring the exact total hashrate in any cryptocurrency, is a difficult, if not intractable problem. Nevertheless, current techniques for estimating the total hashrate extrapolate the information from the difficulty, and the rate at which new blocks are introduced into the blockchain. We use approximations provided by publicly available estimations, such as [9].

5.1 Overview of the Experiments

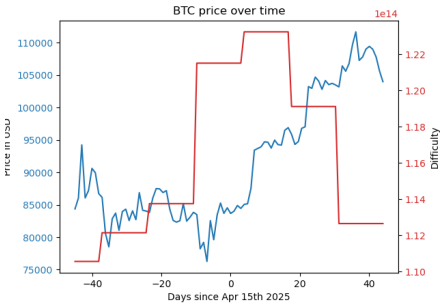
We have conducted an intensive data collection for three representative PoW cryptocurrencies—Bitcoin, Bitcoin Cash, and Ethereum Classic—and are visualizing our findings for several snapshots indicating different periods of volatility vs. stability, and for systems list Bitcoin Cash and Ethereum Classic, snapshots around periods when attack were known to have occurred. In this section, we describe our experiments and data visualization, and in Section 5.2 we discuss how these results can be interpreted using common principles of economic theory in combination with out utility definition.

Bitcoin We begin by analyzing Bitcoin in two separate points in time over a 90-day period, centered around April 15, 2025, and April 15, 2023. We source difficulty data from the blockchain itself, using the target values provided in the main chain itself which is used to calculate an estimated p_r. We collected daily pricing data, and mining costs are modeled by using the average cost (over each month) of electricity across all sectors of the economy, according to the US Energy Information Administration. When selecting mining machines for our model, we select machines that were relatively recently released, and thus were more efficient and profitable.

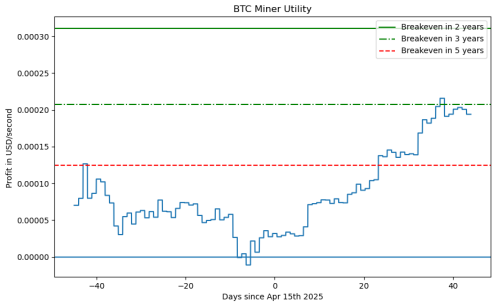
For 2025, results are visualized in Figures 1a and 1b, which depict the price (solid blue line) with the mining difficulty (solid red line), and the utility respectively of Bitcoin from around March 1st to June 1st. In Figure 1b, the bottommost horizontal line represents zero utility gained; the miner doesn't lose or gain any money. The green line, green dash line, and the red dash line represent the utility necessary to pay back the mining hardware within 2, 3, or 5 years respectively (see next paragraph for mining-hardware cost estimation choice). Similarly, we present the results for Bitcoin in 2023 in Figures 2a and 2b. In Figure 2b, the horizontal lines label the utility needed to pay back the mining hardware within 6 months, 12 months, 2 years, 3 years, and 5 years, from top to bottom.

Bitcoin as of April 2025 We used the Bitmain Antminer S21E XP Hyd 3U as the basis for our analysis. This ASIC miner is rated to output 860 TH/s, while using 11,180 watts of power, costing $\$19,599$ per unit. We estimate electricity at \$.1322/kWh in April, though we collected data for March and May as well. We can calculate the resulting $\texttt{mcost}(r)$ as $4.69 \cdot 10^{-19}$ per hash during April.

Currently, the block reward for successfully mining a block is 3.125 BTC, which halves every 210,000 blocks, roughly four years. Difficulty recalculation occurs every epoch=2016 blocks, or about two weeks. On April 1st, 2025, the difficulty was 113.7575T. This provides a probability $p(r) = 1.92 \cdot 10^{-24}$. The network hashrate is estimated to be around 878.70 EH/sec. An attacker who wanted to obtain majority control of the network by adding new machines would need to add $n = 1,021,745$ of the aforementioned machines to do so. This would imply a capital cost of more than 20 billion USD, and consume a minimum of 11.42 GW of power.



(a) Price of BTC around April 15th 2025

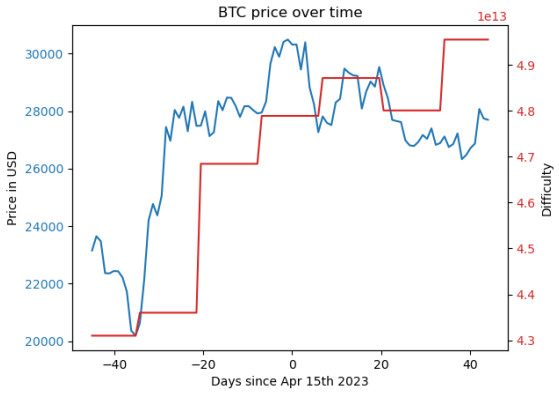


(b) Utility of BTC around April 15th 2025

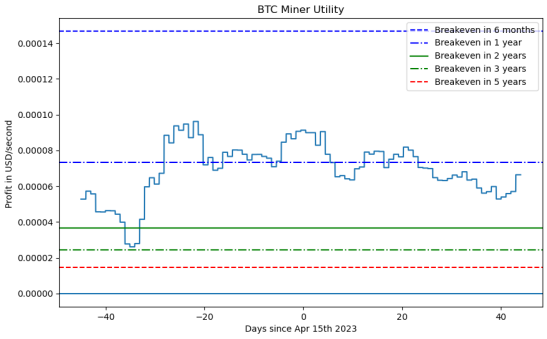
Fig. 1: Analysis of BTC around April 15th 2025

Bitcoin as of April 2023 As before, we additionally examine Bitcoin as of 2023. We use the Bitmain Antminer S19 XP hydro as our model device, offering 257 Thash/s at 5350 W, for a price of $\$2,313$

USD. Electricity is estimated to be \$.1280/kWh in April of 2023. In 2023, the block reward was 6.25 BTC.



(a) Price of BTC around April 15th 2023



(b) Utility of BTC around April 15th 2023

Fig. 2: Analysis of BTC around April 15th 2023

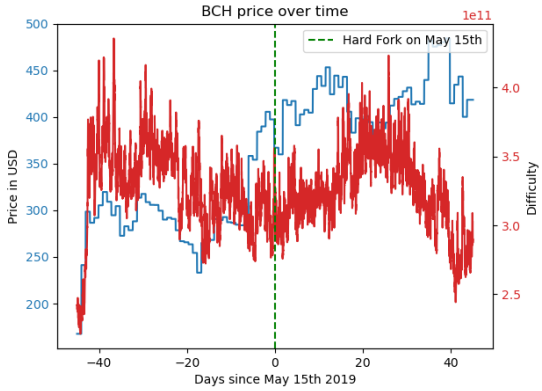
Bitcoin Cash We explore Bitcoin Cash in April through June of 2019. During this time period, a 51% attack was widely reported [31] to have occurred. Two mining pools, BTC.com and BTC.top, colluded in an attempt to undo fraudulent transactions by another miner. We use the Bitmain Antminer S15 as the mining hardware for our analysis. Boasting performance of 28 TH/s while using 1596 watts, the S15 was introduced in December of 2019 for a price of \$1,475 USD [22]. We estimate the hash rate of the network in May of 2019 at around 2.3 EH/s, which implies that $n = 82,142$.

The price of electricity is modeled as the average cost of electricity in the US across all sectors, as reported by the US Energy Information Administration, which was $\$.1007/kWh$ in April. Bitcoin Cash's difficulty recalculation function is different from Bitcoin; most notably in that it recalculates difficulty each block.

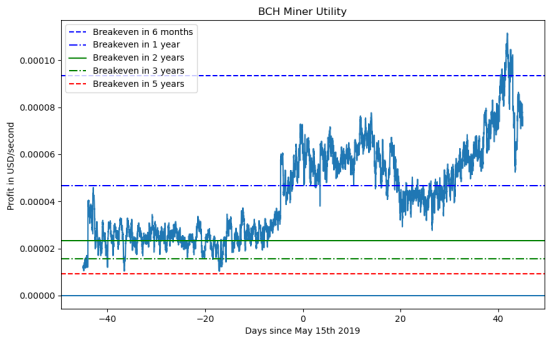
The price (solid blue line) of Bitcoin Cash and the difficulty (solid red line) is visualized in Figure 3a from April 1st to July 1st, 2019. The hard fork is marked in the middle by a large vertical line. We display the utility of a miner under the aforementioned assumptions in Figure 3b. There, we display the breakeven utility for a miner to pay back their machine within 6 months, 12 months, 2 years, 3 years, and 5 years, from top to bottom.

Ethereum Classic In Ethereum Classic, we explore three periods of time. The first two are 90 day periods centered around April 15, 2025 and April 15, 2023. We also examine a 24 month period from October 2019 to October 2021. We source difficulty data from the main chain. We note that the target recalculation period in Ethereum Classic during this time frame is after every block.

Ethereum Classic as of April 2025 We use the iPollo V2H as our model miner. This ASIC is rated to provide 3.4 GH/sec, using 475 watts of power, for a price of \$3,769 USD. We estimate



(a) Price of BCH around May 15th 2019

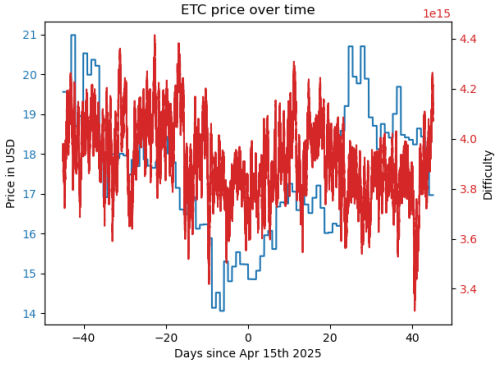


(b) Utility of BCH around May 15th 2019

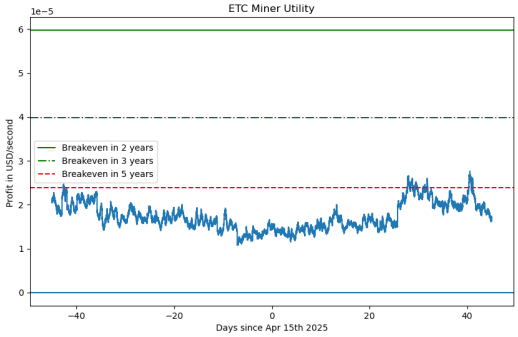
Fig. 3: Analysis of BCH around May 1st 2019

electricity prices in April 2025 at roughly \$.1313/kWh, which implies a per hash cost of $\texttt{mcost}(r) = 5.095 \cdot 10^{-15}$. Here, we use 290 TH/sec as our estimate for the hashrate. This would imply that 85,294 machines would be equivalent to the current hash rate, which collectively would cost over \$321 million USD and draw 40.51 MW of power.

For reference, as of April 1st, 2025, the price is roughly \$17.75. Finally, the block rewards for Ethereum classic are cut by 20% for every 5 million blocks added to the chain. Currently, this means that $\texttt{breward}(r) = 2.048$, when round r occurs on the aforementioned date. We visualize our results in Figures 4a and 4b. The former depicts the price and difficulty over the period, while the latter displays the utility of the miner. We add horizontal lines to denote the 2 year, 3 year and 5 year payoff rate, and the zero utility rate.



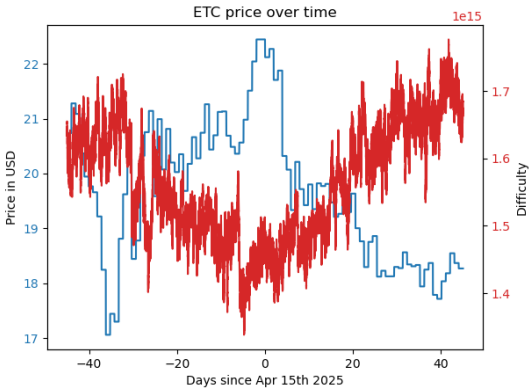
(a) Price of ETC around April 15th 2025



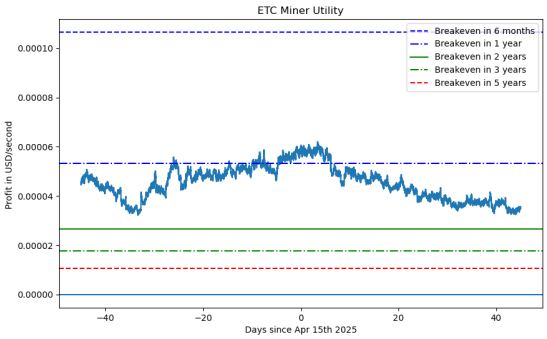
(b) Utility of ETC around April 15th 2025

Fig. 4: Analysis of ETC around April 15th 2025

Ethereum Classic as of April 2023 For 2023, we use the Jasminer X16-Q ASIC as our model device. The ASIC is rated for 1.95 Ghash/s while using 620 watts of power, and sells for $\$1,680$. We estimate electricity prices in April as $\$.1280$/kWh using the same source as before. In 2023, the block reward was still 2.56 ETC. The price and difficulty is depicted in Figure 5a; the utility of a miner is depicted in Figure 5b. For the miner utility, we denote the 6 month, 1 year, 2 year, 3 year, and 5 year payoff rate with horizontal lines, and the zero utility rate with a solid blue line.



(a) Price of ETC around April 15th 2023



(b) Utility of ETC around April 15th 2023

Fig. 5: Analysis of ETC around April 15th 2023

Ethereum Classic as of 2019-2021 Here, we apply our framework to Ethereum Classic over a longer period of time, ranging from October 2019 to October 2021. During this time, several events impacted the blockchain, including but not limited to: a forking attack in August 2020 [11,10,19], the COVID-19 pandemic, and a block reward decrease. We collect daily price data, and per block difficulty data. Electricity costs are calculated using the average electricity costs across all sectors of the US economy according to the US Energy Information Administration; data is collected over a monthly frequency. We incorporate the block reward decrease at block 10,000,000, which occurred in March 2020.

We use the Antminer E3 as the basis model for our analysis. Originally released in 2018, the Antminer boasts a hash rate of around 180 Mhash/sec for 800 watts. We estimate an adversary would have to purchase 54,556 new machines, for an estimated cost of $\$81,834,000$ USD, which uses a minimum of 43.645 MW of power to obtain a majority of the network.

Results of our analysis are depicted in Figure 6 and Figure 7. The former displays the difficulty and price of Ethereum Classic over a two year period centered around October 1st, 2020. The latter shows the utility of a miner operating during this time frame. We mark the 6 month, 1 year, 2 year, 3 year, and 5 year payback rate with a series of horizontal lines.

5.2 Interpretation of Results

Our analysis shows that typical price fluctuations do not substantially affect honest miner utility. In both BTC and ETC, pricing was significantly more volatile in 2025 compared to 2023. We refer

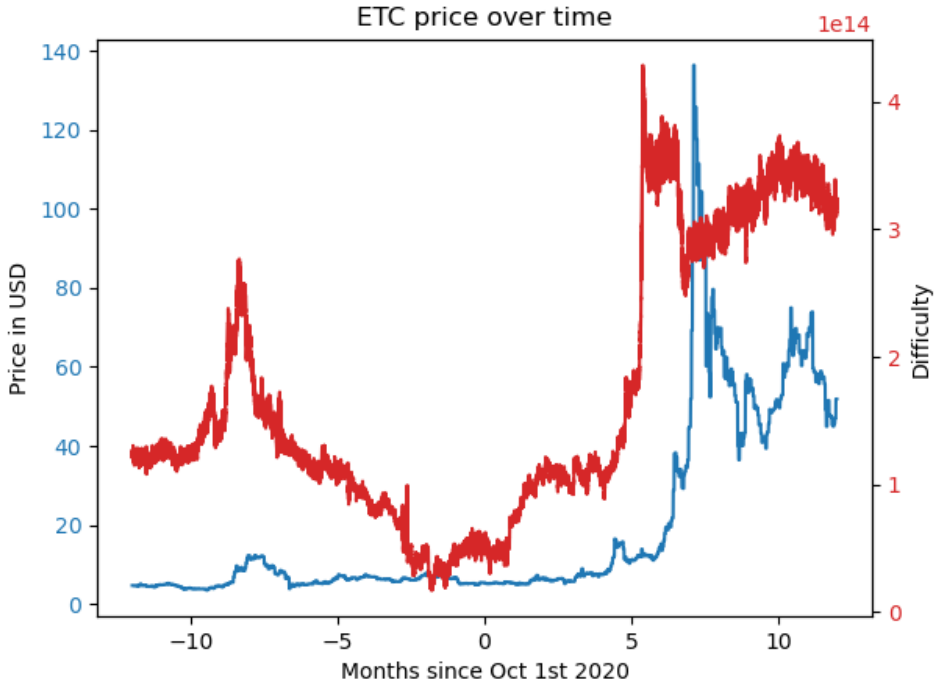


Fig. 6: Price of ETC from October 2019-2021

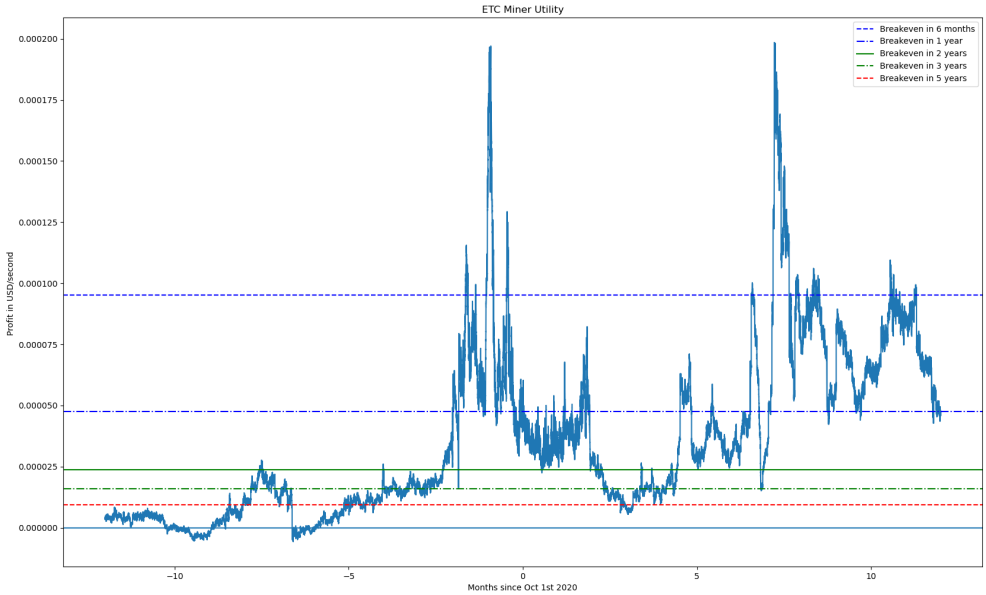


Fig. 7: Utility of ETC from October 2019-2021

the reader to Figure 4a and 5a for the price of Ethereum over the period, and to Figure 1 a and 2 a for the price of Bitcoin over the same period. Our modeling predicts that the miner will obtain a positive utility throughout the entire period.

We note that if the price drops (e.g., due to a fork or external event), or if electricity costs are greater than our assumed parameters, then honest parties' utilities may no longer be positive. We observe that around a ten percent decrease in price, coupled with a ten percent increase in mining costs for any given difficulty is enough to render mining unprofitable in the Bitcoin protocol, both in the 2023 and 2025 data sets. We emphasize that in the case of a price decrease, every participant is affected, impacting the difficulty substantially. This is opposed to the case where a change in mining cost occurs, which may affect fewer miners.

Over a two year period, we observe that an Ethereum Classic miner is expected to make a profit over the long run (see Figure 7). We generally observe that the utility of an honest miner roughly follows the price of the asset, with a notable deviation around the fall of 2020. This meteoric rise in utility is mediated by a precipitous drop in difficulty, which results in a higher probability of successfully solving a proof of work. We conjecture that this was largely caused by the uncertainty caused by the COVID-19 pandemic and the ensuing chaos in the markets.

We note that for a few days, our miner receives a negative utility toward the beginning. In both instances, the difficulty decreases as mining power leaves the network. Since the miners participating in the protocol do not have a homogeneous utility function, the ones who had the most significant losses, e.g., due to higher electricity costs or older hardware, were likely the first to drop out of the protocol. Some miners may choose to remain in the network despite unprofitability, e.g., speculating that the price of the asset may increase later. This behavior would be particularly expected in less risk-adverse miners. During this timeframe, Ethereum Classic underwent it's fourth 'fifthening', which reduced the block reward from 4 ETC to 3.2 ETC. This validates our model; we would expect that in the case where miners suddenly are not profiting, then they would stop mining, which would decrease the difficulty, leading to greater expected reward for the remaining miners.

Over the two year period in question, Ethereum Classic suffered from a series of 51% percent attacks in August of 2020 (cf. [11,10,19]). In line with observations from [39,41], the price of Ethereum Classic did not catastrophically decline as a result, but instead appeared relatively stable, as indicated by Figure 6. Perhaps surprisingly, the utility of the miner appears to increase as miners flee the network, causing a drop in difficulty which increases the probability of remaining miners successfully solving a proof of work.

Our results suggest that a miner is typically able to recoup their costs after one to four years, depending on the assumptions and the protocol examined. This assumption is consistent with prior studies that suggest similar time frames for miners [38]. We find that our experiments are in line with the free-entry assumption that miners can enter the market with little to no barriers, and that they generally obtain a positive utility for participating in the protocol.

References

1. T. Antonopoulos and T. Terauchi. Games for security under adaptive adversaries. In S. Delaune and L. Jia, editors, *CSF 2019 Computer Security Foundations Symposium*, pages 216–229. IEEE Computer Society Press, 2019.
2. C. Badertscher, R. Canetti, J. Hesse, B. Tackmann, and V. Zikas. Universal composition with global subroutines: Capturing global setup within plain uc. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part III*, page 1–30, Berlin, Heidelberg, 2020. Springer-Verlag.

3. C. Badertscher, J. A. Garay, U. Maurer, D. Tschudi, and V. Zikas. But why does it work? A rational protocol design treatment of bitcoin. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 34–65. Springer, 2018.
4. C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Consensus Redux: Distributed Ledgers in the Face of Adversarial Supremacy . In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, pages 143–158, Los Alamitos, CA, USA, July 2024. IEEE Computer Society.
5. C. Badertscher, J. Hesse, and V. Zikas. On the (ir)replaceability of global setups, or how (not) to use a global ledger. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II*, page 626–657, Berlin, Heidelberg, 2021. Springer-Verlag.
6. C. Badertscher, Y. Lu, and V. Zikas. A rational protocol treatment of 51% attacks. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 3–32, Virtual Event, Aug. 2021. Springer, Heidelberg.
7. C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas. Bitcoin as a transaction ledger: A composable treatment. *CRYPTO*, 2017.
8. D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In R. A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, Lecture Notes in Computer Science, pages 307–323. Springer, 1992.
9. Bitinfocharts. Bitcoin hashrate historical chart.
10. B. Blog. Attacker stole 807k etc in ethereum classic 51% attack.
11. B. Blog. Ethereum classic 51% chain attack july 31, 2020.
12. R. Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.
13. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In S. P. Vadhan, editor, *Theory of Cryptography*, pages 61–85, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
14. R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a global random oracle. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, Nov. 2014.
15. R. Canetti and T. Rabin. Universal composition with joint state. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Heidelberg, Aug. 2003.
16. M. Carlsten, H. A. Kalodner, S. M. Weinberg, and A. Narayanan. On the instability of bitcoin without the block reward. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 154–167. ACM Press, Oct. 2016.
17. T. H. Chan, N. Ephraim, A. Marcedone, A. Morgan, R. Pass, and E. Shi. Blockchain with varying number of players. *Cryptology ePrint Archive*, 2020.
18. J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.
19. Coinbase. Coinbase's perspective on the recent ethereum classic (etc) double spend incidents.
20. Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 112–130. Springer, Heidelberg, Aug. 2000.
21. I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.
22. Y. Finance. Bitmain releases antminer s15: How it stacks up against competitors.
23. G. Fuchsbauer, J. Katz, and D. Naccache. Efficient rational secret sharing in standard communication networks. In *Proceedings of the 7th International Conference on Theory of Cryptography*, TCC'10, page 419–436, Berlin, Heidelberg, 2010. Springer-Verlag.
24. J. Garay, Y. Lu, J. Prat, B. Testa, and V. Zikas. A composability treatment of bitcoin's transaction ledger with variable difficulty. In C. Garman and P. Moreno-Sanchez, editors, *Financial Cryptography and Data Security*, pages 233–248, Cham, 2026. Springer Nature Switzerland.
25. J. A. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th Annual IEEE Symposium on Foundations of Computer Science – FOCS 2013*, pages 648–657. IEEE Computer Society, 2013.
26. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

27. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323. Springer, Heidelberg, Aug. 2017.
28. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.
29. S. D. Gordon and J. Katz. Rational secret sharing, revisited. In *Proceedings of the 5th International Conference on Security and Cryptography for Networks*, SCN'06, page 229–241, Berlin, Heidelberg, 2006. Springer-Verlag.
30. J. Halpern and V. Teague. Rational secret sharing and multiparty computation: extended abstract. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 623–632, New York, NY, USA, 2004. Association for Computing Machinery.
31. A. Hertig. Bitcoin cash miners undo attacker's transactions with 51% attack. https://www.coindesk.com/tech/2019/05/24/bitcoin-cash-miners-undo-attackers-transactions-with-51-attack.
32. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In *Theory of Cryptography Conference*, pages 477–498. Springer, 2013.
33. A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tselekounis. Blockchain mining games. In *EC*, 2016.
34. A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, Aug. 2017.
35. A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*, page 705–734, Berlin, Heidelberg, 2016. Springer-Verlag.
36. R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In J.-S. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 643–673, Cham, 2017. Springer International Publishing.
37. R. Pass and a. shelat. Micropayments for decentralized currencies. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 207–218. ACM Press, Oct. 2015.
38. J. Prat and B. Walter. An equilibrium model of the market for bitcoin mining. *Journal of Political Economy*, 129(8):2415–2452, 2021.
39. S. Ramos, F. Pianese, T. Leach, and E. Oliveras. A great disturbance in the crypto: Understanding cryptocurrency returns under attacks. *Blockchain: Research and Applications*, 2(3):100021, 2021.
40. M. Rosenfeld. Analysis of bitcoin pooled mining reward systems. *CoRR*, 2011.
41. S. Shanaev, A. Shuraeva, M. Vasenin, and M. Kuznetsov. Cryptocurrency value and 51% attacks: evidence from event studies. *Journal of Alternative Investments*, 22(3):65–77, Dec. 2019.
42. I. Tsabary and I. Eyal. The gap game. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 713–728. ACM Press, Oct. 2018.
43. G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2025):1–42, 2014.

A Glossary of Terms in the Utility Definitions

We present a glossary of terms frequently used throughout our analysis in Table 1.

B Useful Theorems and Definitions

Here we provide a list of useful theorems and relevant definitions.

Protocol $\mathsf{StateExchange}$.

Theorem 2. *[24] Protocol* $\mathsf{StateExchange}^{q,T_0,\tau,m}(P)$ *UC-realizes functionality* $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathbf{STX}})$ *in the* $(\mathcal{W}^q(\mathcal{F}_{\mathrm{RO}}^{\kappa}), \mathcal{F}_{\mathrm{DIFF}}^{\mathsf{bc}})$*-hybrid model.*

Parameter	Meaning
$n(r), t(r)$	The number of total parties in the network at round r, and the total number of corrupted parties, respectively
p_r	The probability that an individual PoW query will succeed at round r. Note that within a given epoch, p_r will remain constant
$f(r)$	The probability of a fork event occurring. This corresponds to $\Pr[K_{r,c}]$
b	The number of blocks inserted into the blockchain within a specific round, as specified by the event $I^{\mathtt{A}}_{b,c,r}$
$\mathtt{breward}(r)$	The reward for successfully inserting a block into the blockchain on round r. Does not include fees
$\mathtt{D}(r)$	The discounting function that describes how a good's future value is worth less than the current value. Parameterized by round r
c	The current conversion rate at round r of the cryptocurrency to a fiat currency (e.g, USD)
$\mathtt{fpayoff}(c, r)$	The payoff for successfully causing a fork
q	The number of queries that is made by one party to the random oracle per round
$\mathtt{mcost}(r)$	The cost to the adversary to make a single query to the random oracle at round r. This incorporates all variables costs, including the cost of electricity
$\mathtt{scost}(r)$	The cost (to designer) for the environment spawning a new party in round r
$\mathtt{ccost}(g, r)$	The cost to the adversary for corrupting g parties. If the honest parties are the counterparty to the trade, then they also receive the cost
μ	The total fees collectible by a miner upon successfully mining a single block, as specified by the events $F^{\mathtt{A}}_{r,c,\mu}$ (resp.,$F^{\mathtt{D}}_{r,c,\mu}$ for the designer)
m	Size of an epoch in blocks. For Bitcoin, this is 2016
τ	Target recalculation factor. $\tau = 4$ in Bitcoin

Table 1: Glossary of utility function terms.

The designer's utility. The full definition of the designer's utility is displayed below:

$$
\begin{aligned}
u_{\mathtt{D}}(\Pi, \mathcal{A}) \triangleq \inf_{\mathcal{Z} \in \mathcal{Z}_{\mathtt{A}}} \Bigg\{ \sup_{\mathrm{S}^{\mathcal{A}} \in \mathcal{S}_{\mathtt{A}}} \Bigg\{ \sum_{r \in \mathbb{N}} \mathtt{D}(r) \Bigg(& \sum_{b \in \mathbb{N}, c \in \mathbb{Q}} b \cdot \mathtt{breward}(r) \cdot c \cdot \Pr[I^{\mathtt{D}}_{b,c,r}] \\
& + \sum_{c \in \mathbb{Q}, \mu \in \mathbb{N}} \mu \cdot c \cdot \Pr[F^{\mathtt{D}}_{c,r,\mu}] \\
& - \sum_{q \in \mathbb{N}} q \cdot \mathtt{mcost}(r) \cdot \Pr[W^{\mathtt{D}}_{q,r}] \\
& - \sum_{s \in \mathbb{N}} s \cdot \mathtt{scost}(r) \cdot \Pr[S^{\mathcal{Z}}_{s,r}] \\
& + \sum_{t \in \mathbb{N}} \mathtt{ccost}(t, r) \cdot \Pr[C^{\mathtt{D}}_{t,r}] \Bigg) \Bigg\} \Bigg\}.
\end{aligned}
$$

C Ideal Ledger Functionalities and Algorithms

We present the weakened ledger functionality below. It follows a similar structure to the dynamic difficulty ledger introduced by [24], including using a wrapper around the ledger. Let $\mathsf{weak\text{-}ledger\text{-}params} = \{\mathtt{windowSize}, \mathsf{Validate}, \mathsf{ExtendPolicy}, \mathsf{Blockify}, \mathtt{predict\text{-}time}\}$, where

$\texttt{windowSize} \in \mathbb{N}$ We also provide a weakened version of $\mathcal{W}^{\mathsf{stx\text{-}params}}(\mathcal{F}_{\mathtt{STX}})$ (although the internal $\mathcal{F}_{\mathtt{STX}}(\mathcal{P}, \Delta, p_H, p_A)$ is omitted, as it is almost identical to that which is presented in [24].

Functionality $\mathcal{W}^{\mathsf{weak\text{-}ledger\text{-}params}}(\mathcal{G}_{\text{WEAK-LEDGER}})$

Parameters: `windowSize`; Algorithms Validate, weakExtendPolicy, Blockify, `predict-time`.
Clock-time: The functionality maintains a variable τ_L that is kept in-sync with clock-time: Upon any activation (and thus also initialization), the ledger first sends $(\text{CLOCK-READ}, cid)$ to $\bar{\mathcal{G}}_{\text{CLOCK}}$ to receive the answer $(\text{CLOCK-READ}, cid, \tau_L)$, then proceeds with the remaining actions.
Variables and initialization: The functionality initializes `state-tree`, s_{ep}, `NxtBC`, $\vec{\mathcal{I}}_H^T \leftarrow \epsilon$, buffer $\leftarrow \emptyset$, $\Delta = 0$ as well as party sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS} \leftarrow \emptyset$. It also keeps track of the cardinality per round of the party sets, indexed by i i.e $(\mathcal{P}_i, \mathcal{H}_i, \mathcal{P}_{DS,i})$.
Setting Delay :
Upon receiving $(\text{SET-DELAY}, sid, d)$ from the adversary $\mathcal{A}$, if $d \in \mathbb{N}$ and SET-DELAY has never been received by this functionality, then set $\Delta = d$.

- Upon receiving $(\text{REGISTER}, sid)$ from some party P (or from $\mathcal{A}$ on behalf of a corrupted P), forward the request to $\mathcal{G}_{\text{WEAK-LEDGER}}$
- Upon receiving $(\text{DE-REGISTER}, sid)$ from some party $P \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted $P \in \mathcal{P}$), forward the request to $\mathcal{G}_{\text{WEAK-LEDGER}}$

Party Set Management:
Upon any other input I received from a party $P_i \in \mathcal{P}$ or from the adversary $\mathcal{A}$ the following steps are taken:

1. If $P_i \in \mathcal{H}$ or if I is a corruption message from $\mathcal{A}$ targeting $P_i \in \mathcal{H}$, then update $\vec{\mathcal{I}}_H^T \leftarrow \vec{\mathcal{I}}_H^T \parallel (I, P_i, \tau_L)$. If a party P_i gets corrupted, additionally update $\mathcal{H} \leftarrow \mathcal{H} \setminus \{P_i\}$, $\mathcal{P}_{DS} \leftarrow \mathcal{P}_{DS} \setminus \{P_i\}$
2. Let $\hat{P} := \{P \in \mathcal{P}_{DS} | \tau_P^{reg} < \tau_L - 4\Delta\}$. Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \{\hat{P}\}$.
3. If the message was not received from $\mathcal{G}_{\text{WEAK-LEDGER}}$ forward the message, along with the appropriate parameters and variables that this functionality stores.

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

Party Management:
- Upon receiving $(\text{REGISTER}, sid)$ from some party P (or from $\mathcal{A}$ on behalf of a corrupted P), set $\mathcal{P} = \mathcal{P} \cup \{P\}$, initialize $\mathtt{pt}_P \leftarrow 1, \mathsf{state}_P \leftarrow \epsilon$, and $\tau_P^{reg} \leftarrow \tau_L$. If P is an honest party and if $\mathcal{H} = \emptyset$ send $(\text{REGISTER}, cid)$ to $\bar{\mathcal{G}}_{\text{CLOCK}}$. If P is honest then update $\vec{\mathcal{I}}_H^T$ and set $\mathcal{H} \leftarrow \mathcal{H} \cup \{P\}$ and if additionally $\tau_P^{reg} > 0$ holds, set $\mathcal{P}_{DS} \leftarrow \mathcal{P}_{DS} \cup \{P\}$. Return $(\text{REGISTER}, sid, P)$ to the caller.
- Upon receiving $(\text{DE-REGISTER}, sid)$ from some party $P \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted $P \in \mathcal{P}$), set $\mathcal{P} \leftarrow \mathcal{P} \setminus \{P\}$, $\mathcal{H} \leftarrow \mathcal{H} \setminus \{P\}$, $\mathcal{P}_{DS} \leftarrow \mathcal{P}_{DS} \setminus \{P\}$. If $\mathcal{H} = \emptyset$, send $(\text{DE-REGISTER}, cid)$ to $\bar{\mathcal{G}}_{\text{CLOCK}}$. If P is honest then update $\vec{\mathcal{I}}_H^T$. Return $(\text{DE-REGISTER}, sid, P)$ to the caller.

Upon receiving forwarded input from the wrapper, do the following:

Honest Party Operations: If $P_i \in \mathcal{H}$ then additionally take the following steps:

1. $\{(\vec{N}_{\mathtt{pt}_1}, \mathtt{pt}_1), \ldots, (\vec{N}_{\mathtt{pt}_k}, \mathtt{pt}_k)\} \leftarrow \mathsf{weakExtendPolicy}(\vec{\mathcal{I}}_H^T, \texttt{state-tree}, \texttt{NxtBC}, \mathsf{buffer}; s_{ep})$. Reset $\texttt{NxtBC} \leftarrow \epsilon$ and store $s_{ep} \leftarrow s'$.
2. For each $\mathtt{pt}_i$ such that $\vec{N}_{pt_i} \neq \epsilon$ then parse update `state-tree` starting at $\mathtt{pt}_i$ with path $\mathsf{Blockify}(\vec{N}_{\mathtt{pt}_i,1}), \ldots, \mathsf{Blockify}(\vec{N}_{\mathtt{pt}_i,\ell})$

General Party Operations: If the input I is a ledger instruction from a party $P_i \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted party $P_i \in \mathcal{P}$), execute the respective code:

- *Submitting a transaction:* If $I = (\text{SUBMIT}, sid, \mathtt{tx})$ do the following:
  1. Choose a unique transaction ID $\mathtt{txid}$ and set $\mathtt{BTX} \leftarrow (\mathtt{tx}, \mathtt{txid}, \tau_L, P_i)$
  2. Set $\mathsf{buffer} \leftarrow \mathsf{buffer} \cup \{\mathtt{BTX}\}$ and output $(\text{SUBMIT}, \mathtt{BTX})$ to $\mathcal{A}$
- *Reading the state:* If $I = (\text{READ}, sid)$ then do the following: if $P_i \in \mathcal{H} \setminus \mathcal{P}_{DS}$ then set $\mathsf{state}_i := \mathsf{state}|_{min\{pt_i, |\mathsf{state}\}}$. Return $(\text{READ}, sid, \mathsf{state}_i)$ to the caller.
- *Maintaining the ledger state:* If $I = (\text{MAINTAIN-LEDGER}, sid, minerID)$ and $P_i \in \mathcal{H}$ and $\mathtt{predict\text{-}time}(\vec{\mathcal{I}}_H^T) > \tau_P^{reg}$ then send $(\text{CLOCK-UPDATE}, cid)$ to $\bar{\mathcal{G}}_{\text{CLOCK}}$. Else send I to $\mathcal{A}$.

Adversarial Messages: If the input I is an additional adversarial capability (received on the backdoor tape from $\mathcal{A}$) execute the respective code:

- *The adversary reading the state:* If $I = (\text{READ}, sid)$, then return $(\mathsf{state}, \mathsf{buffer}, \vec{\mathcal{I}}_H^T)$ to $\mathcal{A}$.
- *The adversarial proposing the next block:* If $I = (\text{NEXT-BLOCK}, \mathtt{pt}, (\mathtt{txid}_1, \ldots, \mathtt{txid}_\ell))$, update $\mathtt{NxtBC}$ as follows:
  1. Set listOfTxid $\leftarrow \epsilon$, and check that $\mathtt{pt}$ points to a leaf of $\mathtt{state\text{-}tree}$. (Otherwise ignore the command)
  2. For $i = 1, \ldots, \ell$ do: if there exists a $\mathtt{BTX} = (\mathtt{tx}, \mathtt{txid}, minerID, \tau_L, p_i) \in \mathsf{buffer}$ with ID $\mathtt{txid} = \mathtt{txid}_i$ then set listOfTxid := listOfTxid$\|$ $\mathtt{txid}_i$.
  3. Finally set $\mathtt{NxtBC}[\mathtt{pt}] := \mathtt{NxtBC}[\mathtt{pt}]$ $\|$listOfTxid and output $(\text{NEXT-BLOCK}, ok)$ to $\mathcal{A}$
- *The adversary proposing a fork:* If $I = (\text{FORK}, (\mathtt{txid}_1, \ldots, \mathtt{txid}_\ell))$, update $\mathtt{NxtBC}$ as follows:
  1. Set listOfTxid $\leftarrow \epsilon$
  2. For $i = 1, \ldots, \ell$ do: if there exists a $\mathtt{BTX} = (\mathtt{tx}, \mathtt{txid}, minerID, \tau_L, P_i) \in \mathsf{buffer}$ with ID $\mathtt{txid} = \mathtt{txid}_i$ then set listOfTxid := listOfTxid$\|$ $\mathtt{txid}_i$.
  3. Finally set $\mathtt{NxtBC}[\mathtt{pt}] := \mathtt{NxtBC}[\mathtt{pt}]$ $\|$listOfTxid and output (FORK, ok) to $\mathcal{A}$
- *The adversary setting state-slackness:* If $I = (\text{SET-SLACK}, (P_{i_1}, \hat{pt}_{i_1}), \ldots, (P_{i_\ell}, \hat{pt}_{i_\ell}))$, with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ then do the following: If for all $P_{i_j} \in \mathcal{H} \setminus \mathcal{P}_{DS}, j \in [\ell] : \hat{pt}_{i_j}$: has greater distance than $\mathtt{pt}_{i_j}$ from the root $\mathtt{state\text{-}tree}$, then update $pt_{i_1} := \hat{pt}_{i_1}$ for every $j \in |\ell|$. Return $(\text{SET-SLACK}, ok)$ to $\mathcal{A}$ Otherwise, set $\hat{pt}_{i_j}$ to the leaf with greatest distance from the root of $\mathtt{state\text{-}tree}$
- *The adversary setting the state for desyncronized parties:* If $I = (\text{DESYNC-STATE}, (P_{i_1}, \mathsf{state}'_{i_1}), \ldots, (P_{i_\ell}, \mathsf{state}'_{i_1}))$, with $\{P_{i_1}, \ldots, P_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ then set $\mathsf{state}_{i_j} := \mathsf{state}'_{i_j}$ for each $j \in [\ell]$ and return $(\text{DESYNC-STATE}, ok)$ to $\mathcal{A}$.
- *The adversary obtaining the set of registered parties* If $I = (\text{GET-REGISTERED}, sid)$, then return $(\text{GET-REGISTERED}, sid, \mathcal{P})$ to $\mathcal{A}$.
- *The adversary corrupting a party*: If $I = (\text{CORRUPT}, sid, P_i)$ and $\mathtt{predict\text{-}time}(\vec{\mathcal{I}}_H^T) > \tau_L$ then send $(\text{CLOCK-UPDATE}, cid)$ to $\bar{\mathcal{G}}_{\text{CLOCK}}$. Else return I to $\mathcal{A}$

Algorithm weakDefaultExtendPolicy

Let τ_L be the current ledger time (computed from $\vec{\mathcal{I}}_H^T$)
Set $\mathtt{pt}_D$ to the longest branch of $\mathtt{state\text{-}tree}$ and denote by state_D the corresponding state.
$\vec{N}_{\mathsf{df}} \leftarrow \epsilon$
Set $\vec{N}_{\mathsf{df}} \leftarrow \mathtt{tx}^{\mathsf{coinbase}}_{\mathsf{minerID}}$ of an honest miner
Sort buffer according to the time stamps and let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_n)$ be the transactions in buffer
$\mathtt{st} \leftarrow \mathsf{Blockify}_{\mathcal{B}}(\vec{N}_{\mathsf{df}})$
repeat
  Let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_n)$ be the current list of (remaining) transactions
  for $i = 1$ to n **do**

if $\mathsf{ValidTx}_{\vec{\mathbb{B}}}(\mathtt{tx}_i, \mathsf{state} \parallel \mathtt{st}) = 1$ **then**
$\vec{N}_{\mathsf{df}} \leftarrow \vec{N}_{\mathsf{df}} \parallel \mathtt{tx}_i$
Remove $\mathtt{tx}_i$ from $\vec{\mathtt{tx}}$
Set $\mathtt{st} \leftarrow \mathsf{Blockify}(\vec{N}_{\mathsf{df}})$
end if
end for
until $\vec{N}_{df}$ does not increase anymore
return $\vec{N}_{\mathsf{df}}, \mathtt{pt}_D$

Algorithm weakExtendPolicy

$\vec{N}_{df}, \mathtt{pt}_D \leftarrow \mathsf{DefaultExtension}(\vec{\mathcal{I}}_H^T, \mathsf{state}, \mathsf{buffer}, \mathtt{NxtBC}, s_{ep})$
Let τ_L be the current ledger time (computed from $\vec{\mathcal{I}}_H^T$)
Read $\vec{\tau}_{\mathtt{state}}$ and $\vec{\mathtt{hf}}$ from the passed state s_{ep}. If the state is empty, initialize two empty vectors.
$\mathsf{target} \leftarrow \mathsf{gettarget}^{m,T_0,\tau,\mathsf{b_{tgt}}}(\mathsf{state})$
$\vec{N} \leftarrow \epsilon$
for each $\mathtt{pt}$ such that $\mathtt{NxtBC}[pt] \neq \epsilon$ **do**
Set state to the state corresponding to pointer $\mathtt{pt}$
Parse $\mathtt{NxtBC}$ as a vector $((hFlag_1, \mathtt{NxtBC}_1), \cdots, (hFlag_n, \mathtt{NxtBC}_n))$
$\vec{N}_{\mathtt{pt}} \leftarrow \epsilon$
for each list $\mathtt{NxtBC}_i$ of transaction IDs **do**
$\vec{N}_i \leftarrow \epsilon$
Use the txid contained in $\mathtt{NxtBC}_i$ to determine the list of transactions
Let $\vec{\mathtt{tx}} = (\mathtt{tx}_1, \ldots, \mathtt{tx}_{|\mathtt{NxtBC}_i|})$ denote the transactions of $\mathtt{NxtBC}_i$
if $\mathtt{tx}_1$ is not a coin-base transaction **then**
$\mathsf{inv} \leftarrow 1$, goto [terminate]
else
$\vec{N}_i \leftarrow \mathtt{tx}_i$
for $j = 2$ to $|\mathtt{NxtBC}_i|$ **do**
$\mathtt{st}_i \leftarrow \mathsf{Blockify}_{\vec{\mathbb{B}}}(\vec{N}_i)$
if $\mathsf{ValidTx}_{\vec{\mathbb{B}}}(\mathtt{tx}_j, \mathsf{state} \parallel \vec{\mathtt{st}}_i) = 0$ **then**
$\mathsf{inv} \leftarrow 1$, goto [terminate]
end if
$\vec{N}_i \leftarrow \vec{N}_i \parallel \mathtt{tx}_j$
end for
$\mathtt{st}_i \leftarrow \mathsf{Blockify}_{\vec{\mathbb{B}}}(\vec{N}_i)$
end if
$\vec{N}_{\mathtt{pt}} \leftarrow \vec{N} \parallel \vec{N}_i$
$\vec{T} \leftarrow \vec{T} \parallel \mathsf{target}$
$\mathsf{state} \leftarrow \mathsf{state} \parallel \mathtt{st}_i$
$\vec{\tau}_{\mathtt{state}} \leftarrow \vec{\tau}_{\mathtt{state}} \parallel \tau_L$
end for
$\vec{R} = \vec{R} \parallel (\vec{N}_{\mathtt{pt}}, \mathtt{pt})$
Update `state-tree` (the local copy) to include state
end for
$\mathsf{inv} \leftarrow 0$
[terminate]:
if $\mathsf{inv} = 0$ **then**
return $\vec{R}$
else

$\vec{T} \leftarrow \vec{T} \parallel \mathsf{target} \parallel \cdots \parallel \mathsf{target}$ (extended by $|\vec{N}_{\mathsf{df}}|$ elements)
$\vec{\tau}_{\mathtt{state}} \leftarrow \vec{\tau}_{\mathtt{state}} \parallel \tau_L \parallel \cdots \parallel \tau_L$
return $\vec{N}_{\mathsf{df}}, \mathtt{pt}_D$
end if

Functionality $\mathcal{W}^{\text{stx-params}}(\mathcal{F}_{\text{WEAK-STX}}(\mathcal{P}, \Delta, p_H, p_A))$

The functionality maintains a party set $\mathcal{P} \leftarrow \emptyset$. The functionality maintains a set of trees for each party $\mathcal{T}_P$ of 3-tuple $(\vec{\mathtt{st}}, \vec{T}, \vec{\tau_s})$, where each $\vec{\mathtt{st}}$ is unique. The functionality also maintains a value $\Delta \in \mathbb{N}$, initialized to 0. It also registers with the global clock functionality $\bar{\mathcal{G}}_{\text{CLOCK}}$

Setting the delay :
Upon receiving (SET-DELAY, sid, d) from the adversary $\mathcal{A}$, if $d \in \mathbb{N}$ and SET-DELAY has never been received by this functionality, then set $\Delta = d$. Return (SET-DELAY, sid, ok)

Registration :
Upon receiving any REGISTER or DE-REGISTER command, sends (CLOCK-READ, cid) to $\bar{\mathcal{G}}_{\text{CLOCK}}$ to receive the answer (CLOCK-READ, cid, τ_L)

- Upon receiving (REGISTER, sid) from some party P (or from $\mathcal{A}$ on behalf of a corrupted P), set $\mathcal{P} = \mathcal{P} \cup \{P\}$ and initialize the tree $\mathcal{T}_P \leftarrow \mathtt{gen}$ where each rooted path corresponds to a valid state the party has received. Return (REGISTER, sid, P) to the caller.
- Upon receiving (DE-REGISTER, sid) from some party $P \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted $P \in \mathcal{P}$), set $\mathcal{P} := \mathcal{P} \setminus \{P\}$, and return (DE-REGISTER, sid, P) to the caller.

Submit/receive new states :

- Upon receiving (SUBMIT-NEW, $sid, \vec{\mathtt{st}}, \mathtt{st}$) from some participant $P_s \in \mathcal{P}$, if $\mathsf{isvalidstate}_{\mathcal{B}}(\vec{\mathtt{st}} \parallel \mathtt{st}) = 1$ and $(\vec{\mathtt{st}}, \cdot, \cdot) \in \mathcal{T}_P$, then do the following:
  1. Set $T \leftarrow \mathsf{gettarget}_{m,T_0,\tau,\mathsf{b_{tgt}}}(\vec{\mathtt{st}})$
  2. Set $p_H \leftarrow 1 - (1 - \frac{T}{2^\kappa})^q$, $p_A \leftarrow \frac{T}{2^\kappa}$ and forward (SUBMIT-NEW, $sid, \vec{\mathtt{st}}, \mathtt{st}$) to $\mathcal{F}_{\text{STX}}(\mathcal{P}, \Delta, p_H, p_A)$
  3. Upon response (SUCCESS, sid, B), if $B = 1$, then add $(\vec{\mathtt{st}} \parallel \mathtt{st}, T, \tau_L)$ to $\mathcal{T}_{\mathcal{P}}$ and send (CONTINUE, sid) to $\mathcal{F}_{\text{STX}}(\mathcal{P}, \Delta, p_H, p_A)$.
  4. Send (SUCCESS, sid, B, T, τ_L) to P_s
- Upon receiving any other input, forward the request to $\mathcal{F}_{\text{STX}}(\mathcal{P}, \Delta, p_H, p_A)$, and return its response.

Algorithm $\mathsf{gettarget}^{m,T_0,\tau,\mathsf{b_{tgt}}}(\vec{\mathtt{st}})$ [24]

1. Compute $i = \lfloor \frac{|\vec{\mathtt{st}}|}{m} \rfloor$
2. If $i = 0$, then return $T = T_0$
3. Parse $\vec{\mathtt{st}} = (st_1, T_1, t_1)|| \ldots ||(st_n, T_n, t_n)$
4. Set $\mathsf{T_{diff}} = t_{i \cdot m - 1} - t_{(i-1) \cdot m}$
5. Set $T = \frac{\mathsf{T_{diff}}}{m \cdot \mathsf{b_{tgt}}} \cdot T_{(i-1) \cdot m}$.
6. If $T > \tau T_{(i-1) \cdot m}$, then set $T = \tau T_{(i-1) \cdot m}$. If $T < \tau T_{(i-1) \cdot m}$, then set $T = \frac{\tau}{T_{(i-1) \cdot m}}$.
7. Return T