Blockchain and Consensus from Proofs of Work without Random Oracles

Juan A. Garay
Texas A&M University
garay@cse.tamu.edu

Aggelos Kiayias[*]
University of Edinburgh
akiayias@inf.ed.ac.uk

Giorgos Panagiotakos
University of Edinburgh
giorgos.pan@ed.ac.uk

Abstract

One of the most impactful applications of "proofs of work" (POW) currently is in the design of blockchain protocols such as Bitcoin. Yet, despite the wide recognition of POWs as the fundamental cryptographic tool in this context, there is no known cryptographic formulation that implies the security of the Bitcoin blockchain protocol. Indeed, all previous works formally arguing the security of the Bitcoin protocol relied on direct proofs in the random oracle model, thus circumventing the difficulty of isolating the required properties of the core POW primitive.

In this work we fill this gap by providing a formulation of the POW primitive that implies the security of the Bitcoin blockchain protocol in the standard model. Our primitive entails a number of properties that parallel an efficient non-interactive proof system: completeness and fast verification, security against malicious provers (termed "hardness against tampering and chosen message attacks") and efficiency and security for honest provers (the latter captured as almost k-wise independence of the proving algorithm running time). Interestingly, our formulation is incomparable with previous formulations of POWs that applied the primitive to contexts other than the blockchain and highlights the importance of *run-time independence* as a property for POWs suitable for blockchain protocols.

Using our primitive and standard properties of the underlying hash function, we establish the security of the Bitcoin backbone protocol [Eurocrypt 2015] without relying on random oracles. We then tackle the question of constructing a consensus protocol based on POW. We illustrate that previously known solutions essentially relied on the random oracle and propose a new blockchain-based consensus protocol provably secure under the same assumptions as above. This yields the first consensus protocol for honest majority reducible to a POW primitive without random oracles.

[*]Research partly supported by Horizon 2020 project PANORAMIX, No. 653497.

Contents

1 Introduction **3**

2 Preliminaries **6**

3 Proofs of Work **7**

4 Bitcoin from Proofs of Work without Random Oracles **10**

5 Consensus from Proofs of Work without Random Oracles **12**

A Bitcoin from Proofs of Work without Random Oracles (cont'd) **17**

A.1 Bitcoin backbone model and definitions 17
A.2 Security properties of the blockchain 18
A.3 The modified Bitcoin backbone protocol 19
A.4 Robust public transaction ledgers 21
A.5 Security proof 22
A.6 Proof of the Modified Bitcoin Backbone Protocol 31
A.7 Comparison with the random-oracle analysis 33

B Consensus from Proofs of Work without Random Oracles (cont'd) **37**

1 Introduction

For over 20 years the term "proof of work" (POW) has been used liberally in the cryptography and security literature to describe a number of cryptographic primitives that were applied in a variety of settings, including spam mitigation [21], sybil attacks [20], and denial of service protection [30, 7]. A POW scheme can be typified as a primitive that entails two main algorithms: A proving algorithm and a verification algorithm, an abstraction which, despite minor differences, can be used to describe most schemes in the literature. The proving algorithm will receive as input a parameter specifying the intended "difficulty" of the POW as well as additional information such as a context or message with respect to which the POW is to be computed. (Intuitively, the higher level of difficulty selected, the more effort [in terms of computational steps] would be required.)

The main properties that almost all previous works share are: (i) *amortization resistance*, which guarantees that the adversary cannot compute POWs without doing work proportional to the computational resources available to it; (ii) the *existence of trapdoors*, or an alternative *POW sampling* computation which when appropriately used, bypasses the previous restriction, and may even allow sampling a POW with effort independent of the given difficulty level; and (iii) *fast verification*, which requires that the time it takes to verify the correctness of a POW is a lot less than the time it takes to compute it.

However, the most impactful application of POWs to date is in the design of "permissionless" blockchain protocols [34], where the use of POWs enables the construction of consensus protocols that are not based on a pre-existing public-key infrastructure or authenticated channels. At a high level, the way POWs help in this context is by slowing down message generation for all parties indiscriminately, thus generating opportunities for honest parties to converge to a unique view under the assumption that the aggregate computational power of honest parties sufficiently exceeds that of the adversary. Now, while this intuition matches the more rigorous analyses of the Bitcoin protocol that have been carried out so far [24, 35, 25, 8], these works have refrained from formally defining the underlying POW algorithms as a stand-alone cryptographic primitive and relied instead on the random oracle (RO) model [14] to prove *directly* the properties of the blockchain protocol, as opposed to following a reduction approach to the underlying cryptographic primitives. The same is true for other provably secure POW-based distributed protocols [5, 31, 26]. In fact, to our knowledge, no prior work has identified the set of properties for the POW primitive that are sufficient to imply the security of a blockchain protocol (see related work below).

Our contributions. In this work we fill the above gap by putting forth a formalization of a notion of POW that we prove to be sufficient, and effectively necessary. for reducing in a black-box manner the security of the Bitcoin backbone protocol [24] to it. This paves the way for establishing the security of Bitcoin in the standard model, as opposed to the random oracle model, assuming our POW primitive can be realized under a simple computational assumption. Interestingly, the set of POW properties we identify is novel, including properties that have not been considered in previous works that attempted to formalize the POW primitive. Specifically, our contributions are three-fold:

1) Formalization of a POW scheme. Our syntax of a POW scheme entails three algorithms: parameter generation, proving and verification—Gen, Prove and Verify, respectively. Gen is invoked on input 1^λ, where λ is the security parameter, and outputs public security parameters pp. Prove is invoked on an input (pp, r, msg, h), where r is a random key that guarantees freshness, msg is a message that will be associated with the POW, and h is the hardness level. Expectedly, Verify is invoked on input (pp, r, msg, h, w) where w is (possibly) an output of Prove. We require a POW scheme to be:

- **Complete:** As in the case of an interactive proof, we require that proofs produced by Prove should be acceptable by the Verify algorithm.

- t_{ver}**-Verifiable:** This property mandates that the POW can be verified in t_{ver} steps by the honest verifier.
- α**-Successful:** This property bounds the probability that an honest prover will successfully produce a POW. Specifically, it requires the probability that Prove terminates in a certain number of steps t to be lower bounded by α, a function of t and the hardness level h.
- **Hard against Tampering and Chosen-Message Attacks ((β, ϵ)-H-TCMA):** This property deals with security against malicious provers, and is akin to the property of existential unforgeability under chosen-message attacks of digital signatures. It captures the fact that producing a sequence of POWs, for chosen messages, does not provide an advantage to an adversary in terms of running time. Specifically, the chances to produce more than $\beta \cdot t$ POWs in t steps (for any t) are less than ϵ. Furthermore, this should hold against an adversary able to tamper with the keys (that's the 'T' in the acronym), and even in the presence of a Prove oracle (excluding of course any proofs produced by the oracle from the adversary's output).
- **Run-time m-wise independent:** This final property deals with security for honest provers. It captures the setting where honest provers are potentially invoked on adversarial inputs and ensures that their running time enjoys some degree of independence. Specifically, we call a POW *run-time m-wise independent* if in the course of an adversarial program execution that makes function calls to Prove, the random variables defined as the running time of each Prove invocation, is a set of almost m-wise independent random variables (cf. [2]).

2) Security of the Bitcoin backbone protocol assuming POW. We then turn to the analysis of the Bitcoin backbone protocol [24] given our POW primitive. We consider a fixed number of parties (following the approach of [24]), but in the standard model assuming our POW primitive. We recall the three basic properties of the blockchain data structure, (strong) common prefix, chain quality and chain growth, and show how the Bitcoin backbone protocol should be modified in order to incorporate our POW primitive. Besides the POW primitive, proving the security of the backbone protocol requires standard collision resistance from the underlying hash function that is used to "glue" the blocks together.

We first prove that using the H-TCMA property and assuming the adversarial hashing power is suitably bounded, it is unlikely in any sufficiently long time window for the adversary to exceed the number of POWs of the honest parties. Then, using the α-Successful and (β, ϵ)-H-TCMA properties in conjunction with run-time m-wise independence, where $m = \Theta(n)$, the number of parties, we establish that summations of running times of successive Prove invocations have the *variance* needed to ensure that "uniquely successful rounds" (i.e., rounds where exactly one of the honest parties produces a POW) happen with high density in any sufficiently long time window. (Recall that a high rate of uniquely successful rounds was fundamental in the security proof of [24].) Using these last two core results, and under suitable constraints for the basic POW parameters $\alpha, \beta, \epsilon, h$ and number of parties n, we prove the security of the Bitcoin backbone protocol in the standard model just assuming our POW primitive and the security of the underlying hash function.

The above analysis also highlights the distinctive nature of our notion of POWs that is geared to the setting of blockchain protocols as well as to the necessity of the properties we identify. First, using a POW that is merely efficiently verifiable, and even H-TCMA secure but without the run-time m-wise independence property, it is easy to construct an attack against the blockchain protocol. Specifically, without m-wise independence, it is conceivable to have a POW that always takes the same number of steps to be solved. Using such a POW, the honest parties will find a solution in the same round and thus an adversary, even without hashing power, but merely exploiting network scheduling, can maintain an indefinite fork. In constrast, as we prove, given m-wise independence, the symmetry between honest parties can be broken for a suitable choice of m, and honest parties are capable of converging to a single blockchain following the longest chain rule (note that we assume a static number of parties). Second, regarding the necessity of H-TCMA, it is easy to see that if the property breaks, the adversary can induce an execution where the freshness keys used to generate POWs afford it an unfair computational advantage. In this way, the number of adversarial POWs will exceed those of the honest parties (despite an adversarial hashing power minority) and the adversary will be able to create a fork. It is worth noting that both H-TCMA and run-time m-wise independence are quite trivial in the random oracle model and as such they were never highlighted in any previous work on the analysis of POW-based blockchain protocols. (Indeed, as a "sanity check," we show that the Bitcoin POW scheme we outline is secure in the random oracle model according to our definitions; moreover, according to the security parameters we obtain for the scheme, the security guarantees we get from our analysis of the Bitcoin backbone are similar to those proved in [24, 35]).

3) Consensus from POW. Finally, we tackle the problem of designing a consensus protocol [36, 33] for an honest majority of parties that can reduce to our POW primitive. The only known consensus protocol in this setting (where no *private* setup assumption is provided) is given in [24] where it is shown how using a technique called "2-for-1 POW" two POW-based protocols can be run concurrently and create a blockchain where the number of honest party contributions is proportional to their actual number; this property is then shown to imply consensus. We stress that this is not the case for the Bitcoin blockchain protocol, where attacks such as block withholding (as used in the "selfish mining" attack [23]) enable the adversary to create blockchains that misrepresent the number of honest parties' contributions.

We observe that this protocol cannot work in our setting as it cannot be reduced to our POW primitive in a black-box way. Indeed, 2-for-1 POW non-trivially relies on the random oracle model and the fact that each witness for a POW in the RO setting can be rearranged in a certain way so as to obtain a test for a witness for *another* POW in a way that is independent from the first solution. Such property is achievable in the RO model but not black-box-reducible to our POW primitive. We propose a new blockchain protocol that circumvents this problem and preserves the required property, namely, that honest parties' contributions to the blockchain reflect their actual number.

The core idea of our protocol is as follows: When mining a block the parties include their input to the consensus protocol as well as the headers of "orphan" blocks that exist in forks stemming off their main chain and have not been included so far. The header of a block contains the hash of the previous block in the chain, the witness to the POW, application specific data (such as the input to the consensus protocol), and a hash that may contain headers of other blocks. Using this mechanism, we show that it is possible to reconstruct the whole tree of block headers from the blockchain contents and thus in this way preserve all block headers produced by the honest parties. This ensures that the resulting ledger will reflect the number of parties and hence a consensus protocol may now be easily reduced to this blockchain protocol.

Prior and related work. Dwork and Naor [22] first considered POWs under the term "pricing functions," as a means of protection against spam e-mail. The main properties discussed in their work are amortization resistance, "moderate hardness" and the existence of trapdoors ("shortcuts" in their terms). Three different constructions were presented there. The first one is based on the difficulty of extracting square roots modulo a prime p and does not have a trapdoor; the second construction has a trapdoor and is based on the hardness of forging signatures (i.e., hardness of factoring in the specific instantiation), while the last construction is based on "broken" signature schemes, where moderately hard algorithms that can forge signatures exist, but the signing key is protected from these attacks.

In a different direction, Juels and Jacobsson [29] and Back [6, 7] use POWs to construct electronic payment systems. In [29] the authors consider the following properties: amortization resistance, fast verification, and some special composability property which states that generating a POW for some scheme may help in generating a POW for another scheme. As acknowledged by the authors themselves, the definitions they provide are only sketches. In [6, 7] another set of closely related properties is considered, but again the approach is not rigorous. Notably, the author mentions the concept of "trapdoor-freeness," i.e., that the party which generates the initial parameters of the scheme should not be able to also generate a trapdoor regarding these parameters. This property comes in sharp contrast with the POW sampling property, signifying a conflict on what actually are the *essential* properties of a POW.

More recently, Bitansky *et al.* [17] construct time-lock puzzles as well as POW schemes from randomized encodings. Since the focus of their work is time-lock puzzles, the properties of POW schemes—amortization resistance, fast verification, and POW sampling—are only briefly investigated, although they do instantiate a POW scheme based on randomized encodings and the existence of non-amortizing languages in the worst case.

Another interesting approach is that of Ball *et al.* [10], who construct a POW scheme that can be used to do "useful" work. That is, an instance of a problem is selected and a proof of work on this instance along with a solution can be constructed, while at the same time preserving amortization resistance. While the authors mention that their POW scheme can be used for Bitcoin, no formal proof is provided that Bitcoin security reduces to the primitive they construct.

Taking a further step in this same direction, i.e., proving that Bitcoin's security can be reduced to an assumption on the underlying POW scheme, Poelstra [37] proposes that one could base the security of Bitcoin on so-called "dynamic membership multi-party signatures." Unfortunately, this work does not offer a formal treatment of the primitive nor a reduction of the security of the Bitcoin protocol to it.

In [3], Alwen and Tackmann study moderately hard functions (MoHF), providing simulation based definitions for what they call "non-interactive proofs of effort" (niPoE), which—as explicitly acknowledged by the authors—cannot be used to analyze Bitcoin. The main impediment is that the adversary can only invoke the same MoHF only once per protocol session, while for the Bitcoin protocol multiple invocations of the same MoHF should be allowed.

Finally, we note that we perform our analysis in the static setting, i.e., where the POW difficulty remains the same throughout the execution as in [24]. We refer to [25] for an analysis of Bitcoin in the dynamic setting and leave as an open question the extension of our results to this setting.

Organization of the paper. The basic computational model, definitions and cryptographic building blocks used by our constructions are presented in Section 2. Formal definition of a POW and its security properties are presented in Section 3. Section 4 and Appendix A are dedicated to Bitcoin, showing how the Bitcoin backbone protocol should be modified to incorporate our POW primitive and reducing its security to it, with the former presenting a summary and the latter details and proofs. Further, and as a "sanity check," in Appendix A.7 we also analyze Bitcoin's POW implementation, showing that it is secure in the random oracle model according to our definition. Finally, the new consensus protocol for an honest majority based on POW and without random oracles is presented in Sections 5 and Appendix B.

2 Preliminaries

We let λ denote the security parameter, and use $x \overset{\$}{\leftarrow} X$ to denote choosing a value uniformly at random from set X. In this paper we will follow the concrete approach [12, 15, 27, 16] to security evaluation rather than the asymptotic one. We will use functions t, ϵ, whose range is $\mathbb{N}, \mathbb{R}$ respectively and have possibly many different arguments, to denote concrete bounds on the running time (number of steps) and probability of adversarial success of an algorithm in some given computational model, respectively. When we speak about running time this will include the execution time plus the length of the code (cf. [16]; note also that we will be considering uniform machines). We will always assume that t is a polynomial on the security parameter λ, although we will sometimes omit this dependency for brevity.

Instead of using interactive Turing machines (ITMs) as the underlying model of distributed computation, we will use (interactive) RAMs. The reason is that we need a model where subroutine access and simulation do not incur a significant overhead. ITMs are not suitable for this purpose, since one needs to account for the additional steps to go back-and-forth all the way to the place where the subroutine is stored. A similar choice was made by Garay *et al.* [27]; refer to [27] for details on using interactive RAMs in a UC-like framework, as well as to Section A.1. Given a RAM M, we will denote by $\mathsf{Steps}_M(x)$ the random variable that corresponds to the number of steps of M given input x. We will say that M is t-bounded if it holds that $\Pr[\mathsf{Steps}_M(x) \leq t(|x|)] = 1$.

Finally, we remark that in our analyses there will be asymptotic terms of the form $\mathsf{negl}(\lambda)$ and concrete terms; throughout the paper, we will assume that λ is large enough to render the asymptotic terms insignificant compared to the concrete terms.

Cryptographic primitives and building blocks. We will make use of the following notion of security for cryptographic hash functions (see, e.g., [38] for a thorough review):

Definition 1. Let $\mathcal{H} = \{\{H_k : M(\lambda) \rightarrow Y(\lambda)\}_{k\in K(\lambda)}\}_{\lambda\in\mathbb{N}}$ be a hash-function family, and $\mathcal{A}$ be a PPT adversary. Then $\mathcal{H}$ is *collision resistant* if and only if for any $\lambda \in \mathbb{N}$ and corresponding $\{H_k\}_{k\in K}$ in $\mathcal{H}$

$$\Pr[k \overset{\$}{\leftarrow} K; (m, m') \leftarrow \mathcal{A}(1^\lambda, k); (m \neq m') \wedge (H_k(m) = H_k(m'))] \leq \mathsf{negl}(\lambda)$$

We will also need some classical notions related to random variables:

Definition 2. The *statistical distance* of two random variables X, Y with range U, denoted $\Delta(X, Y)$, is defined as

$$\Delta(X, Y) = \frac{1}{2}\sum_{u\in U} |\Pr[X = u] - \Pr[Y = u]|.$$

For $\epsilon > 0$, we say that X, Y are *ϵ-close* when $\Delta(X, Y) \leq \epsilon$.

Definition 3. A set of random variables $X_1, \ldots, X_n$ is *k-wise independent* if, for any subset $I \subseteq [n]$, where $|I| \leq k$, and for any values x_i, $i \in I$:

$$\Pr[\bigwedge_{i\in I} X_i = x_i] = \prod_{i\in I} \Pr[X_i = x_i].$$

3 Proofs of Work

At a high level, a proof-of-work (POW) scheme is a protocol that enables one party to convince others that she has invested some computational power during some specific time interval and with respect to a specific piece of information, which we will generically refer to as "message." In this section we formalize this notion and present its desired security properties.

POW syntax. We formalize the POW notion as follows. Given a specific security parameter λ, let PP be the public parameter space, R be the key space, M the message space and W the witness (or POW) space. With foresight, the role of the key is to provide freshness for the POW computation and the role of the message is to allow information to be embedded into the POW; see Remark 1 below regarding the significance of these input elements.

Definition 4. A POW scheme consists of three algorithms $\mathsf{POW} = (\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ where:

- $\mathsf{Gen}(1^\lambda)$ is a randomized algorithm that takes as input the security parameter λ, and returns a set of public parameters $pp \in PP$.

- $\mathsf{Prove}(pp, r, msg, h)$ is a randomized algorithm that takes as input public parameters $pp \in PP$, a key $r \in R$, a message $msg \in M$ and hardness parameter $h \in \mathbb{N}$, and returns a witness $w \in W$. Sometimes, informally, we may refer to w itself as the proof of work.
- $\mathsf{Verify}(pp, r, msg, h, w)$ is a deterministic algorithm that takes as input public parameters $pp \in PP$, a key $r \in R$, message $msg \in M$, hardness parameter $h \in \mathbb{N}$ and a witness $w \in W$, and returns true or false to indicate the validity of the proof.

Remark 1. In contrast with related literature on the subject [9, 17], we use in our syntax both a key and a message argument. By explicitly separating the key from the message, we aim at decoupling two requirements of any POW scheme: (i) The key is a way to guarantee that the computational work is "fresh," i.e., executed during a specific time interval (say, from the time the key became known to the prover), and (ii) the message is a way to allow arbitrary information to be encoded in the computed POW. Regarding the relevance of these inputs parameters, observe the following. If there is no way to tell when a POW was created (i.e., freshness is not guaranteed), then the adversary may precompute an arbitrary number of POWs before the protocol starts and try to trick other parties by pretending they all started at the same time. On the other hand, if there is no way to encode information in the POW, the usefulness of the primitive would be rather limited. Even though POW's may be guaranteed to be fresh, if there is no context related to their computation, it will be easy for an adversary to copy POWs across sessions or to different and possibly adversarial contexts and thus "steal" the computational effort of honest participants.

Security properties. Next, we present a number of security properties that we will require POW schemes to satisfy. We start with the completeness property.

Definition 5. We say that a POW scheme is *complete* if for every $pp \in PP, r \in R, h \in \mathbb{N}$, and $msg \in M$:

$$\Pr\left[\mathsf{Verify}(pp, r, msg, h, \mathsf{Prove}(pp, r, msg, h)) = \mathsf{true}\quad\right] \geq 1 - \mathsf{negl}(\lambda).$$

Next, we require that the verification time is bounded by a parameter t_{ver}.

Definition 6. We say that a POW scheme is t_{ver}-*verifiable*, if the Verify algorithm takes time at most t_{ver} (on all inputs).

Next, we capture how the adversary can subvert an honest verifier (resp., prover) in the context of POWs. In the first case, the adversary's objective is to compute a number of POWs a lot faster than an honest prover, while in the second case it is to make the honest prover take too much time to generate a POW. It is useful to notice the parallel to the properties of soundness and zero-knowledge for zero-knowledge proofs. As in that case, the two properties we are after here will be complementary.

We deal with malicious provers first. We put forth an attack that we will use to express a class of adversaries that attempt to create POWs faster than expected. Intuitively, this constitutes an attack against an honest verifier that may be trying to gauge a certain measure using the number of POWs. The game defining the attack is shown in Figure 1; we call the corresponding security property *Hardness against Tampering and Chosen Message Attack* (H-TCMA). In order to capture the interactions that the adversary may have with different parties, we allow the adversary to have access to a proving oracle $\mathcal{P}$, essentially allowing him to simulate any scenario where he interacts with other parties that use the same POW scheme. Every time the oracle is queried, we assume that it runs the Prove procedure with uniformly sampled randomness. A subtle point in the modeling of security in the presence of such oracle is that $\mathcal{P}$ should also "leak" the time it took for a query to be processed. In an actual execution while interacting with honest parties that are producing POWs, time is a side channel that

```
$\mathsf{Exp}^{\text{H-TCMA}}_{\mathcal{A},\mathcal{F}}(1^\lambda, h, \ell)$
$\Sigma \leftarrow U_\lambda$; $pp \leftarrow \mathsf{Gen}(1^\lambda)$;                         (Public parameters)
$st \leftarrow \mathcal{A}_1(1^\lambda, \Sigma, pp)$;                                    (Precomputation)
$r \leftarrow \mathcal{U}_m$;                                                    (Fresh randomness)
$(f_i, msg_i, w_i)_{i\in[\ell]} \leftarrow \mathcal{A}_2^{\mathcal{P}(\cdot,\cdot)}(1^\lambda, r, st)$;                   (POW computation)
return $\bigwedge_{i=1}^{\ell} \begin{pmatrix} \mathsf{Verify}(pp, f_i(\Sigma,r), msg_i, w_i) \wedge \neg\mathsf{Asked}(f_i(\Sigma,r), msg_i, w_i) \\ \wedge\, (f_i \in \mathcal{F}) \wedge (\forall j \in [\ell] : f_i(\Sigma,r) = f_j(\Sigma,r) \rightarrow f_i = f_j) \end{pmatrix}$
```

Figure 1: *The Hardness against Tampering and Chosen-Message Attack experiment for a POW scheme.*

may influence the adversarial strategy; in order to preserve the dependency on this side channel we will require from $\mathcal{P}$ to leak this information.

In addition, we require that the keys used by the adversary to construct proofs be fresh, i.e., we want to avoid situations where the adversary outputs POWs that he has precomputed a long time ago. We model this by requiring that the keys are related to a uniformly random string r generated *externally* to the adversary. Hence, the adversary will only know r after it was given to him, thus providing us with a point of reference before which this value cannot be predicted.

Further, we allow the adversary to tamper with the POW key by manipulating it via tampering functions belonging to a family of functions $\mathcal{F}$; each POW key can be manipulated with a different tampering function. Looking forward, the tampering function in our application will be related to a keyed hash function, where the key of the hash is part of a *common random string* (CRS). Hence, we choose to model functions in $\mathcal{F}$ to have two inputs: Σ (the CRS) and r. Moreover, the output of the adversary is deemed invalid if he tampers r with functions f_1, f_2 in such a way that $f_1(\Sigma, r) = f_2(\Sigma, r)$. Otherwise, the adversary could launch a generic attack that is unrelated to the POW scheme, and produce POWs at twice the rate of an honest prover, as follows. The adversary first finds f_1, f_2 that have this property, and then computes POWs using the tampered key $f_1(\Sigma, r)$. The trick is that each of them will also correspond to a POW with key $f_2(\Sigma, r)$. Hence, he effectively can double the rate at which he produces POWs. Tampering function classes with this property have been considered in the *related key attacks* literature under the term *claw-free*; see, e.g., [11].

More formally, the adversary will have access to $\mathcal{P}(\cdot,\cdot)$, a proof-of-work oracle that on input (f, msg), where $f \in \mathcal{F}$ and $msg \in M$, returns the pair (w, t) where w is the output of $\mathsf{Prove}(pp, f(\Sigma, r), msg, h)$ and t is the number of steps taken by the Prove algorithm on these parameters. Function $\mathsf{Asked}(s, msg, w)$ is true if w was a response of $\mathcal{P}$ to some query (f, msg), where $s = f(\Sigma, r)$.

We are now ready to formulate the security property of *Hardness against Tampering and Chosen Message Attacks* (H-TCMA). It has two parameters, β and ϵ, and, informally, it states that no adversary $\mathcal{A}$ exists in the experiment of Figure 1 that takes at most t steps after receiving value r and it produces $\ell \geq \beta \cdot t$ POWs with probability better than ϵ. Note that in total we allow $\mathcal{A}$ to take any polynomial number of steps, i.e., the adversary is allowed to execute a precomputation stage that permits it to obtain an arbitrary number of POWs. In the definition below, we allow β to depend on the hardness level h, and ϵ on β, t and $q_{\mathcal{P}}$, the number of queries the adversary makes to the proving oracle.

Definition 7. Let $\mathcal{F}$ be a family of functions $f : \{0,1\}^\lambda \times \{0,1\}^m \rightarrow R$, for some $m \in O(\mathsf{poly}(\lambda))$. A POW scheme $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ is (β, ϵ)-*Hard against Tampering and Chosen-Message Attacks* (H-TCMA) with respect to tampering function class $\mathcal{F}$, if for any $h \in \mathbb{N}$, for every polynomials $t_{\mathsf{pre}}(\cdot), t(\cdot)$, and every adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ is t_{pre}-bounded and $\mathcal{A}_2$ is t-bounded and makes at most $q_{\mathcal{P}}$ queries to oracle $\mathcal{P}$, the probability of $\mathcal{A}$ winning in $\mathsf{Exp}^{\text{H-TCMA}}_{\mathcal{A},F}(1^\lambda, h, \lceil \beta(h) \cdot t \rceil)$ (Figure 1) is less than $\epsilon(\beta(h), t, q_{\mathcal{P}})$.

In the H-TCMA definition we are going to consider tampering functions classes that at the very least preserve the unpredictability of r. Otherwise, the adversary can generically attack any POW scheme by predicting the tampered POW key and precomputing POWs. Formally, we will say that $\mathcal{F}$ is *computationally unpredictable* if the adversary, given the CRS Σ, cannot predict a value y that he will be able to "hit" when he gains access to r through some $f \in \mathcal{F}$. We note that information-theoretic analogues of this type of tampering classes have being considered in the non-malleable codes literature; see, e.g., [28].

Definition 8. Let $\mathcal{F}$ be a family of functions $f : \{0,1\}^\lambda \times \{0,1\}^m \to \{0,1\}^d$. We will say that $\mathcal{F}$ is *computationally unpredictable* if for any PPT RAM $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ it holds that

$$\Pr_{\Sigma \leftarrow \mathcal{U}_\lambda; r \leftarrow \mathcal{U}_m} \begin{bmatrix} (st, y) \leftarrow \mathcal{A}_1(\Sigma); \\ f \leftarrow \mathcal{A}_2(st, r) : \\ f \in \mathcal{F} \wedge f(\Sigma, r) = y \end{bmatrix} \leq \mathsf{negl}(\lambda).$$

Next, we consider the case of subverting an honest prover. Given that our primitive is non-interactive, subverting an honest prover amounts to finding a certain set of keys over which the honest prover algorithm fails to produce proofs of work sufficiently fast and regularly. We say a POW scheme is *α-successful* when the probability that the prover computes a POW in t steps is at least α.

Definition 9. We say that a POW scheme is *$\alpha(h,t)$-successful* if for any $t \in \mathbb{N}$, $pp \in PP$, $r \in R, msg \in M$ and $h \in \mathbb{N}$ it holds that:

$$\Pr\left[\mathsf{Steps}_{\mathsf{Prove}}(pp, r, msg, h) < t\right] \geq \alpha(h,t).$$

Finally, in the same corrupt-prover setting, we require the proving time of honest provers to have some (limited) independence. This property, in combination with the efficiency and H-TCMA properties, will prove crucial in ensuring that when multiple provers work together the distribution of the number of them who succeed in producing a POW has some "good" variance and concentration properties.

Definition 10. Let M^{Prove} be a polynomially bounded RAM machine, $\mathcal{E}$ denote the ensemble of random variables corresponding to the executions of M^{Prove}, and $\mathcal{I}$ be an ordered set of calls made to the Prove function by M. Let random variable X_i be equal to the number of steps taken by the i-th function call in $\mathcal{I}$. We say that a POW scheme is *run-time m-wise independent*, if for any $I \subseteq \mathcal{I}$, there exists random variable $(Y_i)_{i \in I}$ that is $\mathsf{negl}(\lambda)$-close to $(X_i)_{i \in I}$ and $\{Y_i\}_{i \in I}$ is m-wise independent (cf. Definition 3).

Next, using our POW primitive and standard properties of the underlying hash function, we establish the security of the Bitcoin backbone protocol [24].

4 Bitcoin from Proofs of Work without Random Oracles

As mentioned in Section 1, existing analyses of Bitcoin have relied on the random oracle model to prove the properties of the blockchain protocol directly. In this section we take a reduction approach to the underlying cryptographic primitive—POW, as defined in Section 3—to prove the security of the Bitcoin backbone protocol [24] in the standard model. Due to space limitations, here we only present a summary of these results and refer the reader to Appendix A for the details.

Model. We analyze the Bitcoin backbone protocol in the model of [24] with some modifications. In this model the execution is driven by an "environment" program $\mathcal{Z}$ that may spawn multiple instances running protocol Π. The programs in question can be thought of as "interactive RAMs" communicating through registers in a well-defined manner, with instances and their spawning at the discretion of a control program which is also an IRAM and is denoted by C. In particular, the control program C forces the environment to perform a "round-robin" participant execution sequence for a fixed set of parties.

Specifically, the execution driven by $\mathcal{Z}$ is defined with respect to a protocol Π, an adversary $\mathcal{A}$ (also an IRAM) and a set of parties $P_1, ..., P_n$. The protocol Π is defined in a "hybrid" setting and has access to an "ideal functionality," called the *diffusion channel*. When the functionality receives an instruction to diffuse a message m from party P_i it marks the party as complete for the current round; note that m is allowed to be null. The adversary $\mathcal{A}$ is allowed to receive the contents of all messages for the round and specify the contents of the Receive() register for each party P_i. Moreover, it has to specify when it is complete for the current round. When all parties are complete for the current round, the functionality inspects the contents of all Receive() strings and includes any messages that were diffused by the parties in the current round but not contributed by the adversary to the Receive() registers. The variable round is then incremented. The adversary can also corrupt up to t of these parties adaptively.

Finally, we define a predicate on executions and prove our properties in disjunction with this predicate, i.e., either the property holds or the execution is not good.

Definition 11. Let $(t_\mathcal{A}, \theta)$-good be a predicate defined on executions in the hybrid setting described above. Then E *is* $(t_\mathcal{A}, \theta)$*-good*, where E is one such execution, if
- the total number of steps taken by $\mathcal{A}$ and $\mathcal{Z}$ per round is no more than $t_\mathcal{A}$;
- the adversary sends at most θ messages per round.

The Bitcoin backbone protocol. The Bitcoin backbone protocol [24], parameterized by functions $V(\cdot), R(\cdot), I(\cdot)$, is an abstraction of the Bitcoin protocol. In the protocol, each block is a tuple of the form $\langle s, x, w \rangle$, s is called the seed, x is called the data, and w is called the witness; a chain is a sequence of such blocks. At the start of the execution all parties have access to a common *genesis* block that contains the public parameter pp of the POW scheme and the key k of the hash functions H, G used. Differently stated, we assume the existence of a *common reference string* (CRS), that becomes available to all parties at the start of the execution. A chain $\mathcal{C} = B_1 \dots B_m$ is *valid* with respect to the CRS if and only if (i) B_1 is the genesis block, (ii) for any two consecutive blocks $\langle s_i, x_i, w_i \rangle, \langle s_{i+1}, x_{i+1}, w_{i+1} \rangle$ it holds that $H_k(s_i, G_k(x_i), w_i) = s_{i+1}$, (iii) each block is a valid POW, i.e., $\mathsf{Verify}(pp, s_i, x_i, w_i) = \mathsf{true}$, and (iv) the content validation predicate $V(\langle x_1, \dots, x_m \rangle)$ outputs true.

At each round, each party chooses the longest valid chain amongst the ones it has received and tries to extend it by computing a POW. If it succeeds, it diffuses the new block to the network. In more detail, each party will run the Prove procedure for $t_\mathcal{H}$ steps, with the message parameter being determined by the input contribution function $I(\cdot)$, and the key parameter being the hash of the last block. We assume that the hardness parameter h is fixed for all executions. Finally, if the party is queried by the environment, it outputs $R(\mathcal{C})$ where $\mathcal{C}$ it the chain selected by the party; the chain reading function $R(\cdot)$ interprets $\mathcal{C}$ differently depending on the higher-level application running on top of the backbone protocol.

Robust public transaction ledgers. Informally, a *public transaction ledger* can be thought of as a book accessible to all parties where transactions are recorded. Ledgers correspond to (transactions stored in the) chains maintained by the backbone protocol. In the protocol execution there also exists an oracle Txgen that generates valid transactions. Note that it is possible for the adversary to create two transactions that are conflicting; valid ledgers must not contain such transactions. We will assume that the oracle is unambiguous, i.e., that the adversary cannot create transactions that come in 'conflict' with the transactions generated by the oracle.

In [24], suitable definitions were given for functions $V(\cdot), R(\cdot), I(\cdot)$ in order to turn the backbone protocol into a protocol realizing a public transaction ledger. Namely, $V(\langle x_1, \ldots, x_m \rangle)$ is true if its input is a valid ledger. Function $R(\mathcal{C})$ returns the contents of the chain if they constitute a valid ledger, otherwise it is undefined. Finally, $I(st, \mathcal{C}, round, \texttt{INPUT}(), \texttt{RECEIVE}())$ returns the largest subsequence of transactions in the input and receive registers that constitute a valid ledger, with respect to the contents of the chain the party already has, together with a randomly generated neutral transaction. We denote the instantiation of our protocol with these functions by $\Pi_{\mathsf{PL}}^{\mathtt{POW}}$ ('PL' for "public ledger"). For more details we refer to [24].

Definition 12. A protocol Π implements a *robust public transaction ledger* if it satisfies the following two properties:

– **Persistence:** Parameterized by $k \in \mathbb{N}$ (the "depth" parameter), if in a certain round an honest player reports a ledger that contains a transaction tx in a block more than k blocks away from the end of the ledger, then tx will always be reported in the same position in the ledger by any honest player from this round on.

– **Liveness:** Parameterized by $u, k \in \mathbb{N}$ (the "wait time" and "depth" parameters, resp.), provided that a transaction either (i) issued by Txgen, or (ii) is neutral, is given as input to all honest players continuously for u consecutive rounds, then there exists an honest party who will report this transaction at a block more than k blocks from the end of the ledger.

We prove that assuming that the underlying POW scheme is secure, protocol $\Pi_{\mathsf{PL}}^{\mathtt{POW}}$ implements a robust transaction ledger. In more detail, we first show that the (β, ϵ)-H-TCMA property of the POW scheme implies that the rate at which the adversary computes blocks is upper bounded by β. Next, we prove that the rate at which rounds where only one honest party (resp. any honest party) successfully computes a block occur, is lower bounded by quantity γ (resp. f). This quantity depends on parameters α and ϵ of the POW scheme. We take advantage of these two observations by further assuming that the execution parameters are such that $\gamma \geq (1+\delta)\beta \cdot t'_{\mathcal{A}}$, for some $\delta \in (0,1)$; roughly, $t'_{\mathcal{A}}$ corresponds to the time needed by a RAM machine to simulate one round in the execution of the Bitcoin protocol, without taking into account calls made to the Prove subroutine by the honest parties. We call this the *Honest Majority Assumption*, which, in combination with a careful adaptation of the techniques in [24], it is sufficient to prove the security of $\Pi_{\mathsf{PL}}^{\mathtt{POW}}$. For more details refer to Appendix A.

Theorem 13. *Assuming the existence of a common reference string, a collision-resistant hash function, a* POW *scheme that satisfies* $(\beta, \mathsf{negl}(\lambda))$*-H-TCMA and* $(n \log \lambda)$*-wise runtime independence with respect to any computationally unpredictable tampering function class, and model parameters* $\{n, t, h, t_{\mathcal{H}}, t_{\mathcal{A}}, \theta\}$ *that comply with the Honest Majority Assumption, protocol* $\Pi_{\mathsf{PL}}^{\mathtt{POW}}$ *implements a robust public transaction ledger with parameters* $u = 4\lambda$ *and* $k = \lambda(f + \beta n t_{\mathcal{H}})$ *except with negligible probability in* λ.

As a "sanity check," we show in Appendix A.7 that the Bitcoin POW scheme we outline there is secure in the random oracle model according to our definitions; moreover, according to the security parameters we obtain for the scheme, the security guarantees we get from our black-box analysis of the Bitcoin backbone are similar to those proved in [24, 35]

5 Consensus from Proofs of Work without Random Oracles

In this section we show how to achieve consensus (a.k.a. Byzantine agreement [36, 33]) under exactly the same assumptions used for proving the security of the Bitcoin backbone protocol in Section 4 and Appendix A. In [24], consensus is achieved under the Honest Majority Assumption by using the POW construction in a *non-black-box* way, through a mining technique called "2-for-1 POWs." In more detail, the technique shows how miners can compute POWs for two different POW schemes at the cost of one, while at the same time ensuring that their resources cannot be used in favor of one of the two schemes. This cannot be directly translated to the security properties we have defined for POW schemes, as they all apply to the execution of a *single* Prove subroutine. Hence, we would have to introduce "2-for-1 POWs" as an extra property. In this section we show how blockchain-based consensus can be achieved by only using the security properties we have defined, directly, and without the extra non-black-box machinery used in [24]. This yields the first consensus protocol for honest majority reducible to a POW primitive without random oracles. The protocol is based on the Bitcoin backbone protocol, and formally specified by providing adequate definitions for the V, R, I functions presented in Section 4.

We first give a definition of the consensus problem. There are n parties, $t < n$ of which might be corrupted, taking an initial input $x \in V$ (without loss of generality, we can assume $V = \{0, 1\}$).

Definition 14. A protocol Π solves the consensus problem provided it satisfies the following two properties:

- **Agreement.** There is a round after which all honest parties output the same value.
- **Validity.** If all the honest parties have the same input, then they all output this value.

Next, we define some notation and terminology that will be used in the remainder of the section. We will use the terms "input" and "vote" interchangeably, referring to the parties' input in the consensus problem. We will use $\mathit{header}(\langle s, x, w\rangle)$ to denote the "compressed" version of block $\langle s, x, w\rangle$, equal to $\langle s, G(x)||\mathit{vote}, w\rangle$. Note that, as defined, the header of any block is of a fixed size. We also extend the definition of our hash function H as applied to headers of blocks. The hash of the header of some block B will be equal to the hash of B, i.e., $H((\mathit{header}(B)) = H(B) = H_k(s, G_k(x), w)$ (note that the header of B provides all the information need to calculate the hash of B).

We now present a high-level description of the protocol. The basic idea is that during block mining, parties are going to include in their blocks not only their own votes, but also headers of other blocks that they have seen and that *are not* part of their chain. Then, after a predetermined number of rounds, the parties will count the votes "referenced" in a prefix of their chain, including the votes found in the headers of the blocks referenced. In this way, they can take advantage of the robust transaction ledger built in Section 4 (and A). The persistence property implies that the honest parties will all agree on *which* votes should be counted, while the liveness property guarantees that the majority of the counted votes come from honest parties. The reader may be wondering about the reason behind honest parties including in their blocks also headers of other blocks that they have seen but that are not part of their chain. It's because, as shown in [24], the adversary is able to add more blocks in the main chain than his ratio of mining power (e.g., using a selfish-mining attack). This does not hold if the honest parties are able to also count off-chain blocks as our protocol does.

A main technical challenge is to be able to add the block references without making the honest parties' chains grow too large, and at the same time to ensure that the number of honest votes exceeds the adversarial ones. To overcome this challenge, we modify the POW generation algorithm so that it is run on the header of the block, i.e., $\mathsf{Prove}(s, G(x)||\mathit{vote}, h)$ and $\mathsf{Verify}(s, G(x)||\mathit{vote}, h, w)$, respectively. This way we are able to verify the validity of a block as a POW and determine the block's vote by only knowing its header. This is exactly the properties we need for the consensus application.

Moreover, we should be able to tell whether the referenced blocks are "fresh"; that is, the adversary should not be able to reference blocks that it has precomputed and are not related to the genesis block. We achieve this by requiring blockchains to have a special structure in order to be considered valid by the content validation predicate $V(\cdot)$. A chain will be *valid* when the referenced blocks on every prefix of the chain form a *tree* that has the genesis block at its root. In order to check this efficiently, we require that the reference list of each block is ordered, so that each entry extends some block header found in previous entries of the same or parent blocks.

Refer to Section B (Algorithm 5) for details on the modification of the content validation predicate.

Content validation predicate $V(\cdot)$	The validation function $V(\cdot)$ checks that the block headers included in the chain create a a block tree of valid POW's that starts from the genesis block.
Chain reading function $R(\cdot)$ (parameterized by M)	$R(\cdot)$ outputs the majority of the votes found in the block headers of the first M blocks of the selected chain.
Input contribution function $I(\cdot)$	The input function $I(\cdot)$ maintains state of which blocks have been received and outputs the input value x that contains (i) the headers of all valid blocks that extend the genesis block and and are not mentioned in the chain that the party is currently extending, and (ii) the party's input (i.e., 0 or 1).

Figure 2: *Consensus from POWs: The* $\Pi_{\mathsf{BA}}^{\mathtt{POW}}$ *protocol.*

The algorithm runs for $L = O(\lambda)$ rounds, after which it outputs the majority of the votes found in a prefix of the selected chain, of a predetermined length M. We call the resulting protocol $\Pi_{\mathsf{BA}}^{\mathtt{POW}}$ ("BA" for Byzantine agreement). A description of the consensus protocol (specifically, the V, R, I functions) is presented in Figure 2, and an example in Figure 3. Note that all parties terminate the protocol simultaneously. For proof of the theorem refer to Appendix B.

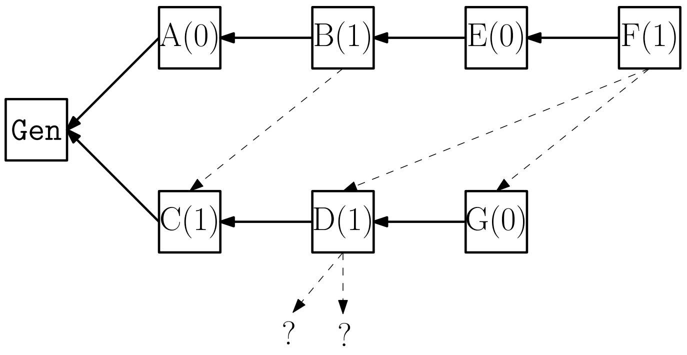

Figure 3: *The data structure maintained by* $\Pi_{\mathsf{BA}}^{\mathtt{POW}}$*. Block B references block C, block F references blocks D and G, and block D references some invalid blocks. This is not a problem, since (i) any chain that contains D will not be selected by any party, and (ii) D's vote is correctly counted since D is a descendant of* Gen.

Theorem 15. *Assuming the existence of a common reference string, a collision-resistant hash function, a* POW *scheme that satisfies $(n \log \lambda)$-wise runtime independence and $(\beta, \mathsf{negl}(\lambda))$-H-TCMA with respect to any computationally unpredictable tampering function class, and model parameters $\{n, t, h, t_{\mathcal{H}}, t_{\mathcal{A}}, \theta\}$ that comply with the Honest Majority Assumption, protocol $\Pi_{\mathsf{BA}}^{\mathtt{POW}}$ solves consensus in $O(\lambda)$ rounds with overwhelming probability.*

References

[1] G. M. Adelson-Velsky and E. M. Landis. An algorithm for the organization of information. *Doklady Mathematics*, 3:1259–1263, 1962.

[2] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 544–553. IEEE Computer Society, 1990.

[3] J. Alwen and B. Tackmann. Moderately hard functions: Definition, instantiations, and applications. In Y. Kalai and L. Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 493–526. Springer, 2017.

[4] G. Andresen. Bip: 34 - block v2, height in coinbase. Cryptology ePrint Archive, Report 2014/956, 2012. `https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki`.

[5] M. Andrychowicz and S. Dziembowski. Distributed cryptography based on the proofs of work. Cryptology ePrint Archive, Report 2014/796, 2014. `http://eprint.iacr.org/`.

[6] A. Back. Hashcash-amortizable publicly auditable cost functions. *Early draft of paper*, 2000.

[7] A. Back. Hashcash–a denial of service counter-measure, 2002.

[8] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas. Bitcoin as a transaction ledger: A composable treatment. In J. Katz and H. Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.

[9] F. Baldimtsi, A. Kiayias, T. Zacharias, and B. Zhang. Indistinguishable proofs of work or knowledge. Cryptology ePrint Archive, Report 2015/1230, 2015. `http://eprint.iacr.org/2015/1230`.

[10] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Proofs of useful work. *IACR Cryptology ePrint Archive*, 2017:203, 2017.

[11] M. Bellare and D. Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In T. Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, pages 666–684, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[12] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 394–403. IEEE, 1997.

[13] M. Bellare, J. Jaeger, and J. Len. Better than advertised: Improved collision-resistance guarantees for md-based hash functions. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 891–906, New York, NY, USA, 2017. ACM.

[14] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[15] M. Bellare and P. Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 399–416. Springer, 1996.

[16] D. J. Bernstein and T. Lange. Non-uniform cracks in the concrete: the power of free precomputation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 321–340. Springer, 2013.

[17] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-lock puzzles from randomized encodings. In M. Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 345–356. ACM, 2016.

[18] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[19] I. B. Damgård. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*, pages 416–427. Springer, 1989.

[20] J. R. Douceur. The sybil attack. In P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors, *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2002.

[21] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992.

[22] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

[23] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography*, 2014.

[24] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *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*, pages 281–310, 2015.

[25] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. *IACR Cryptology ePrint Archive*, 2016:1048, 2016.

[26] J. A. Garay, A. Kiayias, N. Leonardos, and G. Panagiotakos. Bootstrapping the blockchain, with applications to consensus and fast pki setup. In *IACR International Workshop on Public Key Cryptography*, pages 465–495. Springer, 2018.

[27] J. A. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. *Journal of cryptology*, 24(4):615–658, 2011.

[28] Z. Jafargholi and D. Wichs. Tamper detection and continuous non-malleable codes. In *Theory of Cryptography Conference*, pages 451–480. Springer, 2015.

[29] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security*, CMS '99, pages 258–272, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

[30] A. Juels and J. G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*. The Internet Society, 1999.

[31] J. Katz, A. Miller, and E. Shi. Pseudonymous secure computation from time-lock puzzles. *IACR Cryptology ePrint Archive*, 2014:857, 2014.

[32] A. Kiayias and G. Panagiotakos. Speed-security tradeoffs in blockchain protocols. Technical report, IACR: Cryptology ePrint Archive, 2015.

[33] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

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

[35] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.

[36] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[37] A. Poelstra. On stake and consensus (2015). *URL https://download. wpsoftware. net/bitcoin/pos. pdf.*

[38] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International Workshop on Fast Software Encryption*, pages 371–388. Springer, 2004.

[39] S. P. Vadhan et al. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.

A Bitcoin from Proofs of Work without Random Oracles (cont'd)

As mentioned in Section 1, existing analyses of Bitcoin have relied on the random oracle model to prove the properties of the blockchain protocol directly. In this section we take a reduction approach to the underlying cryptographic primitive—POW, as defined in Section 3—to prove the security of the Bitcoin backbone protocol [24] in the standard model. We start with some additional details about the model and pertinent preliminary definitions. We then continue with the security proof of the Bitcoin backbone protocol in the black-box POW setting. We conclude the section showing how Bitcoin's POW scheme is secure in the random oracle model according to our definitions, and that the security guarantees we get from our analysis are similar to those proved in [24, 35].

A.1 Bitcoin backbone model and definitions

In [24] a security model was proposed for the analysis of the Bitcoin backbone protocol. Here we overview the basics, substituting IRAMs for ITMs for the reasons explained in Section 2. The execution of a protocol Π is driven by an "environment" program $\mathcal{Z}$ that may spawn multiple instances running the protocol Π . The programs in question can be thought of as "interactive RAMs" communicating through registers in a well-defined manner, with instances and their spawning at the discretion of a control program which is also an IRAM and is denoted by C. In particular, the control program C forces the environment to perform a "round-robin" participant execution sequence for a fixed set of parties.

Specifically, the execution driven by $\mathcal{Z}$ is defined with respect to a protocol Π, an adversary $\mathcal{A}$ (also an IRAM) and a set of parties $P_1, ..., P_n$; these are hardcoded in the control program C. The protocol Π is defined in a "hybrid" setting and has access to one "ideal functionality," called the *diffusion channel* (see below). It is used as subroutine by the programs involved in the execution (the IRAMs of Π and $\mathcal{A}$) and is accessible by all parties once they are spawned.

Initially, the environment $\mathcal{Z}$ is restricted by C to spawn the adversary $\mathcal{A}$. Each time the adversary is activated, it may communicate with C via messages of the form (Corrupt, P_i). The control program C will register party P_i as corrupted, only provided that the environment has previously given an input of the form (Corrupt, P_i) to $\mathcal{A}$ and that the number of corrupted parties is less or equal t, a bound that is also hardcoded in $\mathcal{C}$. The first party to be spawned running protocol Π is restricted by C to be party P_1. After a party P_i is activated, the environment is restricted to activate party P_{i+1} , except when P_n is activated in which case the next party to be activated is always the adversary $\mathcal{A}$. Note that when a corrupted party P_i is activated the adversary $\mathcal{A}$ is activated instead.

Initially, the diffusion functionality sets a variable round to be 1. It also maintains a Receive() string (register) defined for each party P_i. A party is allowed at any moment to fetch the contents of its personal Receive() string. Moreover, when the functionality receives an instruction to diffuse a message m from party P_i it marks the party as complete for the current round; note that m is allowed to be empty. At any moment, the adversary $\mathcal{A}$ is allowed to receive the contents of all messages for the round and specify the contents of the Receive() string for each party P_i. The adversary has to specify when it is complete for the current round. When all parties are complete for the current round, the functionality inspects the contents of all Receive() strings and includes any messages that were diffused by the parties in the current round but not contributed by the adversary to the Receive() tapes. The variable round is then incremented.

Based on the above, we denote by $\text{VIEW}^{P,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$ the random variable that corresponds to the view of party P at the end of an execution in our model, and by $\text{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$ the concatenation of the views of all parties. The probability space that these variables are defined depends on the coins of all honest parties, $\mathcal{A}$ and $\mathcal{Z}$.

Next, we consider the complications in the modeling due to the analysis of Bitcoin in the concrete security setting. Both in [24] and [35] a modified version of the standard simulation-based paradigm of [18] is followed, where there exist both a malicious environment and a malicious adversary. In addition, the POW scheme is modeled in a non black-box way using a random oracle (RO), and the computational power of the adversary is then bounded by limiting the number of queries it can make to the RO per round. Since in this work the POW scheme is modeled in a black-box way, an alternative approach to bound the adversary's power is needed.

A naïve first approach is to only bound the computational power of $\mathcal{A}$. Unfortunately this will not work for several reasons. Firstly, nothing stops the environment from aiding the adversary, i.e., computing POWs, and then communicating with it through their communication channel or some other subliminal channel; secondly, even if we bound the *total* number of steps of $\mathcal{A}$, it is not clear how to bound the steps it is taking per round in the model of [18], which we build on. Furthermore, another issue arising is that if the adversary is able to send, say, θ messages in each round, it can force each honest party to take $\theta \cdot t_{\mathsf{ver}}$ extra steps per round.

In order to capture these considerations we are going to define a predicate on executions and prove our properties in disjunction with this predicate, i.e., either the property holds or the execution is not good.

Definition 16. Let $(t_{\mathcal{A}}, \theta)$-good be a predicate defined on executions in the hybrid setting described above. Then E *is* $(t_{\mathcal{A}}, \theta)$*-good*, where E is one such execution, if

- the total number of steps taken by $\mathcal{A}$ and $\mathcal{Z}$ per round is no more than $t_{\mathcal{A}}$;[1]
- the adversary sends at most θ messages per round.

For the rest of the paper, whenever we say that some property holds, we will mean that either the property holds or the execution is not $(t_{\mathcal{A}}, \theta)$-good. Moreover, we assume that there exists some polynomial $p(\cdot)$ such that the number of rounds of all executions is upper bounded by $p(\lambda)$.

A.2 Security properties of the blockchain

A number of desired basic properties for the blockchain were introduced in [24, 32]. At a high level, the first property, called *common prefix*, has to do with the existence, as well as persistence in time, of a common prefix of blocks among the chains of honest players. Here we will consider a stronger variant of the property, presented in [32, 35], which allows for the black-box proof of application-level properties (such as the *persistence* of transactions entered in a public transaction ledger built on top of the Bitcoin backbone—cf. Appendix A.4). We will use $\mathcal{C} \preceq \mathcal{C}'$ to denote that some chain $\mathcal{C}$ is a prefix of some other chain $\mathcal{C}'$, and $\mathcal{C}^{\lceil k}$ to denote the chain resulting from removing the last k blocks of $\mathcal{C}$.

Definition 17 ((Strong) Common Prefix)**.** The *strong common prefix property* Q_{cp} with parameter $k \in \mathbb{N}$ states that the chains $\mathcal{C}_1, \mathcal{C}_2$ reported by two, not necessarily distinct honest parties P_1, P_2, at rounds r_1, r_2, with $r_1 \leq r_2$, satisfy $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$.

The next property relates to the proportion of honest blocks in any portion of some honest player's chain.

Definition 18 (Chain Quality)**.** The *chain quality property* Q_{cq} with parameters $\mu \in \mathbb{R}$ and $k \in \mathbb{N}$ states that for any honest party P with chain $\mathcal{C}$ in $\text{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$, it holds that for any k consecutive blocks of $\mathcal{C}$ the ratio of adversarial blocks is at most μ.

[1]The adversary cannot use the running time of honest parties that it has corrupted; it is activated instead of them during their turn. Also, note that it is possible to compute this number by counting the number of configurations that $\mathcal{A}$ or $\mathcal{Z}$ are activated per round.

Further, in the derivations in [24] an important lemma was established relating to the rate at which the chains of honest players were increasing as the Bitcoin backbone protocol was run. This was explicitly considered in [32] as a property under the name *chain growth.* Similarly to the variant of the common prefix property above, this property along with chain quality were shown sufficient for the black-box proof of application-level properties (in this case, transaction ledger *liveness*; see Appendix A.4).

Definition 19 (Chain Growth)**.** The chain growth property Q_{cg} with parameters $\tau \in \mathcal{R}$ (the "chain speed" coefficient) and $s, r_0 \in \mathbb{N}$ states that for any round $r > r_0$, where honest party P has chain $\mathcal{C}_1$ at round r and chain $\mathcal{C}_2$ at round $r + s$ in $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$, it holds that $|\mathcal{C}_2| - |\mathcal{C}_1| \geq \tau \cdot s$.

A.3 The modified Bitcoin backbone protocol

The Bitcoin backbone protocol [24], parameterized by functions $V(\cdot), R(\cdot), I(\cdot)$, is an abstraction of the Bitcoin protocol. First, we introduce some notation needed to understand the description of the algorithms. We will use the terms *block* and *chain* to refer to tuples of the form $\langle s, x, w \rangle$ and sequences of such tuples, respectively. The rightmost block of chain $\mathcal{C}$ is denote by head($\mathcal{C}$). Each block contains a seed, data, and a witness denoted by s, x, w, respectively. At the start of the execution all parties have access to a common *genesis* block that contains the public parameter *pp* of the POW scheme and the key k of the hash functions H, G used. Differently stated, we assume the existence of a *common reference string* (CRS), that becomes available to all parties at the start of the execution. A chain $\mathcal{C} = B_1 \dots B_m$ is *valid* with respect to the CRS if and only if (i) B_1 is the genesis block, (ii) for any two consecutive blocks $\langle s_i, x_i, w_i \rangle, \langle s_{i+1}, x_{i+1}, w_{i+1} \rangle$ it holds that $H_k(s_i, G_k(x_i), w_i) = s_{i+1}$, (iii) each block is a valid POW, i.e., $\mathsf{Verify}(pp, s_i, x_i, w_i) = \mathsf{true}$, and (iv) the content validation predicate $V(\langle x_1, \dots, x_m \rangle)$ outputs true. We call $H_k(B_i)$ the *hash of block* B_i and extend the definition of H to blocks so that $H_k(B_i)$ is equal to $H_k(s_i, G_k(x_i), w_i)$. Moreover, we define $H(\mathcal{C})$ to be equal to the hash of the head of chain $\mathcal{C}$.

At each round, each party chooses the longest valid chain amongst the ones it has received and tries to extend it by computing a POW. If it succeeds, it diffuses the new block to the network. In more detail, each party will run the Prove procedure for $t_{\mathcal{H}}$ steps, with the message parameter being determined by the input contribution function $I(\cdot)$, and the key parameter being the hash of the last block. We assume that the hardness parameter h is fixed for all executions. Finally, if the party is queried by the environment, it outputs $R(\mathcal{C})$ where $\mathcal{C}$ it the chain selected by the party; the chain reading function $R(\cdot)$ interprets $\mathcal{C}$ differently depending on the higher-level application running on top of the backbone protocol.

Algorithm 1 The *proof of work* function, parameterized by q, D and hash functions $H(\cdot), G(\cdot)$. The input is $(x, \mathcal{C})$.

```
function pow($x, \mathcal{C}$)
    $s \leftarrow H(\text{head}(\mathcal{C}))$
    $w \leftarrow \mathsf{Prove}(s, x, h)$                    ▷ Run the prover of the POW scheme.
    $B \leftarrow \varepsilon$
    if $w \neq \perp$ then
        $B \leftarrow \langle s, x, w \rangle$
    end if
    $\mathcal{C} \leftarrow \mathcal{C}B$                          ▷ Extend chain
    return $\mathcal{C}$
end function
```

Algorithm 2 The *chain validation predicate*, parameterized by q, D, the hash functions $G(\cdot), H(\cdot)$, and the *input validation predicate* $V(\cdot)$. The input is $\mathcal{C}$.

```
1: function validate($\mathcal{C}$)
2:     $b \leftarrow V(\mathbf{x}_{\mathcal{C}}) \wedge (C \neq \varepsilon)$        ▷ $\mathbf{x}_{\mathcal{C}}$ describes the contents of the blocks in chain $\mathcal{C}$.
3:     if $b =$ True then        ▷ The chain is non-empty and meaningful w.r.t. $V(\cdot)$
4:         $\langle s, x, w\rangle \leftarrow \text{head}(\mathcal{C})$
5:         $s' \leftarrow H(\text{head}(\mathcal{C}))$
6:         repeat
7:             $\langle s, x, w\rangle \leftarrow \text{head}(\mathcal{C})$
8:             if Verify$(s, x, h, w) \wedge (H(\text{head}(\mathcal{C})) = s')$ then
9:                 $s' \leftarrow s$        ▷ Retain hash value
10:                $\mathcal{C} \leftarrow \mathcal{C}^{\lceil 1}$        ▷ Remove the head from $\mathcal{C}$
11:            else
12:                $b \leftarrow$ False
13:            end if
14:        until $(\mathcal{C} = \varepsilon) \vee (b = \text{False})$
15:    end if
16:    return $(b)$
17: end function
```

The modifications with respect to the original protocol in [24] are as follows: In Algorithm 1 (proof of work function) the Prove function of the underlying POW scheme is invoked for a limited number of steps so that the total number of steps of the invoking party does not exceed the $t_{\mathcal{H}}$ bound per round; in Algorithm 2 (chain validation predicate) the Verify predicate is replaced with a call to the Verify algorithm of the POW scheme; and in Algorithm 3 we assume that the honest parties start the execution with a "genesis" block.

Finally, we will require functions V, I, R to satisfy the following properties.

1. *Input Validity.* The input contribution function $I(\cdot)$ should produce values that verify. Formally, for all $\mathcal{C}$ with $\boldsymbol{x}_{\mathcal{C}} = \langle x_1, \ldots, x_n\rangle$ where $V(\langle x_1, \ldots, x_n\rangle) = 1$, if x is the output of $I(\cdot, \mathcal{C}, \cdot)$, then $V(\langle x_1, \ldots, x_n, x\rangle) = 1$.

2. *Input Entropy.* The probability of the event that two independent invocations of I with arbitrary chosen parameters result in the same value, is negligible in λ.

3. *Input Numbering.* The input contribution function $I(\cdot, \mathcal{C}, \cdot)$ should include the length of $\mathcal{C}$ in a fixed position in the output. $V(\cdot)$ should verify this fact.

The first two requirements were introduced in [24]. We introduce the third one in order to be able to show that the hash chain is collision resistant. Interestingly, a similar choice was made by Bitcoin (see [4]), where the position of the block is included in the coinbase transaction.

Remark 2. There exists a way to transform any set of functions V, I, R that satisfy the input validity condition to one that satisfies the other two properties, by including a random string and the height of the block at the beginning of the output of I. In more detail

$$I'(\cdot, \mathcal{C}, \cdot) = |\mathcal{C}| \;||\; r \;||\; I(\cdot, \mathcal{C}, \cdot)$$

where $|\mathcal{C}|$ is the λ-bit encoding of the length of $\mathcal{C}$, and r is an λ-bit uniformly random string generated by I'. V' and R' should be changed accordingly.

Algorithm 3 The Bitcoin backbone protocol, parameterized by the *input contribution function* $I(\cdot)$ and the *chain reading function* $R(\cdot)$.

```
1: $\mathcal{C} \leftarrow \mathsf{Gen}$                                   ▷ Initialize $\mathcal{C}$ to the genesis block.
2: $st \leftarrow \varepsilon$
3: $round \leftarrow 0$
4: while TRUE do
5:     $\tilde{\mathcal{C}} \leftarrow \mathsf{maxvalid}(\mathcal{C}, \text{any chain } \mathcal{C}' \text{ found in } \textsc{Receive}())$
6:     $\langle st, x\rangle \leftarrow I(st, \tilde{\mathcal{C}}, round, \textsc{Input}(), \textsc{Receive}())$       ▷ Determine the $x$-value.
7:     $\mathcal{C}_{\mathsf{new}} \leftarrow \mathsf{pow}(x, \tilde{\mathcal{C}})$
8:     if $\mathcal{C} \neq \mathcal{C}_{\mathsf{new}}$ then
9:         $\mathcal{C} \leftarrow \mathcal{C}_{\mathsf{new}}$
10:        $\textsc{Broadcast}(\mathcal{C})$
11:    end if
12:    $round \leftarrow round + 1$
13:    if $\textsc{Input}()$ contains $\textsc{Read}$ then
14:        write $R(\mathbf{x}_{\mathcal{C}})$ to $\textsc{Output}()$
15:    end if
16: end while
```

Algorithm 4 The function that finds the "best" chain, parameterized by function max(·). The input is $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$.

```
1: function $\mathsf{maxvalid}(\mathcal{C}_1, \ldots, \mathcal{C}_k)$
2:     $temp \leftarrow \varepsilon$
3:     for $i = 1$ to $k$ do
4:         if $\mathsf{validate}(\mathcal{C}_i)$ then
5:             $temp \leftarrow \max(\mathcal{C}, temp)$
6:         end if
7:     end for
8:     return $temp$
9: end function
```

A.4 Robust public transaction ledgers

A *public transaction ledger* is defined with respect to a set of valid ledgers $\mathcal{L}$ and a set of valid transactions $\mathcal{T}$, each one possessing an efficient membership test. A ledger $\mathbf{x} \in \mathcal{L}$ is a vector of sequences of transactions $\text{tx} \in \mathcal{T}$. Ledgers correspond to chains in the backbone protocol. In the protocol execution there also exists an oracle Txgen that generates valid transactions. Note, that it is possible for the adversary to create two transactions that are conflicting; valid ledgers must not contain conflicting transaction. We will assume that the oracle is unambiguous, i.e., that the adversary cannot create transactions that come in 'conflict' with the transactions generated by the oracle. A transaction is called *neutral* if there does not exist any transactions that comes in conflict with it.

In order to turn the backbone protocol into a protocol realizing a public transaction ledger suitable definitions were given for functions $V(\cdot), R(\cdot), I(\cdot)$ in [24]. Namely, $V(\langle x_1, \ldots, x_m\rangle)$ is true if its input is a valid ledger. Function $R(\mathcal{C})$ returns the contents of the chain if they constitute a valid ledger, otherwise it is undefined. Finally, $I(st, \mathcal{C}, round, \texttt{INPUT}(), \texttt{RECEIVE}())$ returns the largest subsequence of transactions in the input and receive tapes that constitute a valid ledger, with respect to the contents of the chain the party already has, together with a randomly generated neutral transaction. We denote the instantiation of our protocol with these functions by $\Pi_{\mathsf{PL}}^{\mathsf{POW}}$. For more details we refer to [24].

Definition 20. A protocol Π implements a *robust public transaction ledger* if it satisfies the following two properties:

- **Persistence:** Parameterized by $k \in \mathbb{N}$ (the "depth" parameter), if in a certain round an honest player reports a ledger that contains a transaction tx in a block more than k blocks away from the end of the ledger, then tx will always be reported in the same position in the ledger by any honest player from this round on.
- **Liveness:** Parameterized by $u, k \in \mathbb{N}$ (the "wait time" and "depth" parameters, resp.), provided that a transaction either (i) issued by Txgen, or (ii) is neutral, is given as input to all honest players continuously for u consecutive rounds, then there exists an honest party who will report this transaction at a block more than k blocks from the end of the ledger.

A.5 Security proof

Next, we prove that $\Pi_{\mathsf{PL}}^{\mathsf{POW}}$ implements a robust public transaction ledger (Definition 20). First, we introduce some additional notation.

For each round j, we define the Boolean random variables X_j and Y_j as follows. Let $X_j = 1$ if and only if j was a *successful round*, i.e., at least one honest party computed a POW at round j, and let $Y_j = 1$ if and only if j was a *uniquely successful round*, i.e., exactly one honest party computed a POW at round j. With respect to a set of rounds S and some block B, let $Z_B(S)$ denote the number of blocks broadcast by the adversary during S that have B as their ancestor. Also, let $X(S) = \sum_{j \in S} X_j$ and define $Y(S)$ and $X_B(S)$ similarly.

Let t_{bb} (bb for backbone) be an upper bound on the number of steps needed to run the code of an honest party in one round, besides the Prove and Verify calls. By carefully analyzing the backbone protocol one can extract an upper bound on this value.[2] To aid our presentation, we will use $t'_{\mathcal{A}}$ and $t'_{\mathcal{H}}$ to denote the following quantities which roughly correspond to the time needed by a RAM machine to simulate one round in the execution of the Bitcoin protocol, without taking into account calls made to the Prove routine by the honest parties, and to the minimum number of steps spent by an honest party on running the Prove routine in a round, respectively:

$$t'_{\mathcal{A}} = t_{\mathcal{A}} + n \cdot t_{\mathsf{bb}} + \theta t_{\mathsf{ver}} \quad \text{and} \quad t'_{\mathcal{H}} = t_{\mathcal{H}} - t_{\mathsf{bb}} - \theta t_{\mathsf{ver}}.$$

It holds that at least $n - t$ parties will run the Prove routine for at least $t'_{\mathcal{H}}$ steps at every round.

Next, we focus on the hash functions used by the Bitcoin, and the necessary security assumptions to avoid cycles in the blockchains. First, note that in the actual implementation of Bitcoin an unkeyed hash function is used, namely, a double invocation of SHA-256. In previous analyses of the protocol this was modeled as a random oracle. We choose to model it in a strictly weaker way, as a keyed hash function family:

$$\mathcal{H} = \{H_k : \{0,1\}^{\lceil \log(|R|) \rceil + \lceil \log(|W|) \rceil + \lambda} \to \{0,1\}^{\lambda}\}_{k \in K}.$$

that is *collision resistant* (Definition 1); the CRS we have already assumed will contain the key of our hash function. Moreover, as depicted in Figure 4, the protocol makes use of another hash function G to compress the input x of each block, which may of arbitrary size. In our analysis we will require G to be collision resistant. It is well known (see, e.g., [19, 13]) that given a fixed-length collision-resistant hash function family, we can construct an arbitrary-length collision-resistant hash function family. To

[2]Note that t_{bb} depends on the running time of three external functions: $V(\cdot), I(\cdot)$ and $R(\cdot)$. For example, in Bitcoin these functions include the verification of digital signatures, which would require doing modular exponentiations. In any case t_{bb} is at least linear in λ.

aid readability, we will sometimes omit the keys of both functions (as we already do in the description of the protocol). Furthermore, observe that the hash structure of any blockchain (depicted in Figure 4) is similar to the Merkle-Damgard transform [19]:

$$MD_k(IV, x) : z = IV;\ \text{for } i = 1 \text{ to } n \text{ do } z = H_k(z, x[i]);\ \text{return } z,$$

where the fixed-length hash function family used is always assumed to be $\mathcal{H}$. In more detail, for any chain $\mathcal{C}$ we can extract a message *msg* of the form $\langle (G_k(x_1), w_1), \ldots, (G_k(x_n), w_n) \rangle$ such that the hash of the last block, i.e., $H(\text{head}(\mathcal{C}))$, is equal to $MD(H(B), msg)$, where B is *any* ancestor block of $\text{head}(\mathcal{C})$ in $\mathcal{C}$. Based on this observation, we are going to show that the the adversary cannot find distinct chains which *hash* on the same value.

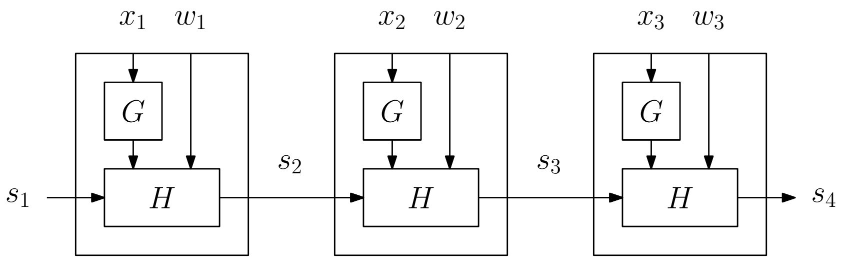

Figure 4: *The hash structure of the blocks in the Bitcoin protocol.*

Lemma 21. *Let $\{H_k(\cdot)\}_{k\in K}$ and $\{G_k(\cdot)\}_{k\in K}$ be collision-resistant hash functions and $I(\cdot)$ satisfy the input entropy and numbering assumptions. Then, the probability that any PPT RAM $\mathcal{A}$ can find two distinct valid chains $\mathcal{C}_1, \mathcal{C}_2$ such that $H(\mathcal{C}_1) = H(\mathcal{C}_2)$ is negligible in λ.*

Proof. Let $m_1 = \langle G(x_i), w_i \rangle_{i\in[|\mathcal{C}_1|]}$ and $m_2 = \langle G(x'_i), w'_i \rangle_{i\in[|\mathcal{C}_2|]}$ so that $MD(H(\mathsf{Gen}), m_1) = H(\mathcal{C}_1)$ and $MD(H(\mathsf{Gen}), m_2) = H(\mathcal{C}_2)$. For the sake of contradiction, assume that the lemma does not hold and there exists an adversary $\mathcal{A}$ that can find valid chains $\mathcal{C}_1$, $\mathcal{C}_2$ such that $H(\mathcal{C}_1) = H(\mathcal{C}_2)$, with non-negligible probability. We will construct an adversary $\mathcal{A}'$ that breaks the collision resistance of H also with non-negligible probability. First, since the two chains are distinct we know that $\langle x_i, w_i \rangle_{i\in[|\mathcal{C}_1|]} \neq \langle x'_i, w'_i \rangle_{i\in[|\mathcal{C}_2|]}$. By the collision resistance property of G it follows that $m_1 \neq m_2$, with overwhelming probability in λ. Next, we show that if the chains have different length, then $\text{head}(\mathcal{C}_1) \neq \text{head}(\mathcal{C}_2)$. By the input numbering assumption $x_{|\mathcal{C}_1|} \neq x_{|\mathcal{C}_2|}$, since the length of the chain is included at the same position in the two strings, and the two chains have different length. The collision resistance of G implies that $\text{head}(\mathcal{C}_1) \neq \text{head}(\mathcal{C}_2)$ with overwhelming probability in λ.

We can now follow the classical inductive argument and show that $\mathcal{A}'$ can find a collision in H, using $\mathcal{C}_1$ and $\mathcal{C}_2$. If $|\mathcal{C}_1| \neq |\mathcal{C}_2|$, then $G(x_{|\mathcal{C}_1|}) \neq G(x_{|\mathcal{C}_2|})$ and $H(\text{head}(\mathcal{C}_1)) = H(\text{head}(\mathcal{C}_2))$, which implies that a collision in H has been found. Otherwise, if $|\mathcal{C}_1| = |\mathcal{C}_2|$, it can be shown inductively that there exists m less or equal to $|\mathcal{C}_1|$, such that $MD(H(\mathsf{Gen}), \langle G(x_i), w_i \rangle_{i\in[m]}) = MD(H(\mathsf{Gen}), \langle G(x'_i), w'_i \rangle_{i\in[m]})$ and $G(x_m)||w_m \neq G(x'_m)||w'_m$. The lemma follows. □

The following two properties, introduced in [24], regarding the way blocks are connected are implied by Lemma 21.

Definition 22. An *insertion* occurs when, given a chain $\mathcal{C}$ with two consecutive blocks B and B_0, a block B^* created after B_0 is such that B, B^*, B_0 form three consecutive blocks of a valid chain. A *copy* occurs if the same block exists in two different positions.

Corollary 23. *Let $\{H_k(\cdot)\}_{k\in K}$ and $\{G_k(\cdot)\}_{k\in K}$ be collision-resistant hash functions and $I(\cdot)$ satisfy the input entropy and numbering assumptions. Then, no insertions and no copies occur with probability $1 - \mathsf{negl}(\lambda)$.*

The hash of a chain is also important because it serves as the POW key of blocks that extend it. We are going to show that the hash structure of a blockchain defines a tampering function class (Definition 7) that satisfies the computational unpredictability property introduced in Definition 8. In more detail, the notion of unpredictability is defined with respect to some specific point in the execution. In our case, this point is going to be the computation of a block B by an honest miner, i.e., the "tampered" POW keys of blocks that extend B will be computationally unpredictable before B is diffused to the network, thus rendering precomputed blocks unusable.

Formally, let C be an arbitrary set of sequences of the form $\langle x_i, w_i\rangle_{i\in[z]}$, where $z \in \mathbb{N}$ and $\langle x_i, w_i\rangle_{i\in[z]} \in (\{0,1\}^* \times \{0,1\}^{\log(|W|)})^z$, with the constraint that for any $\langle x_i, w_i\rangle_{i\in[z]}$, $\langle x'_i, w'_i\rangle_{i\in[z']} \in C$ if $i \neq j$ then $x_i \neq x'_j$. We define $\mathcal{F}$ to be the family of functions of the form

$$f_{(s,e,w,\langle x_i,w_i\rangle_{i\in[z]})}(\Sigma, r) = MD_\Sigma(H_\Sigma(s, G_\Sigma(I(e;r)), w), \langle G_\Sigma(x_i), w_i\rangle_{i\in[z]})$$

for arbitrary $s, e, w, \langle x_i, w_i\rangle_{i\in[z]}$ in $\{0,1\}^\lambda \times \{0,1\}^* \times \{0,1\}^{\log(|W|)} \times C$. We also assume that the randomness of the input contribution function I is sampled from $\{0,1\}^m$ for some $m \in \mathsf{poly}(\lambda)$. Note, that $f_{(s,e,w,\langle x_i,w_i\rangle_{i\in[z]})}(\Sigma, r)$ is equal to the hash of a chain $\mathcal{C}$ where the first block is $\langle s, I(e;r), w\rangle$, and subsequent blocks $\langle s_i, x_i, w_i\rangle_{i\in[z]}$, where s_i is the hash of the previous block in the sequence. We prove that $\mathcal{F}$ is computationally unpredictable.

Lemma 24. *Let $\{H_k(\cdot)\}_{k\in K}$ and $\{G_k(\cdot)\}_{k\in K}$ be collision resistant hash functions and $I(\cdot)$ satisfy the input entropy and numbering assumption. Then, the function family*

$$\mathcal{F} = \{f_{(s,e,w,\langle x_i,w_i\rangle_{i\in[z]})}\}_{(s,e,w,\langle x_i,w_i\rangle_{i\in[z]})\in\{0,1\}^\lambda\times\{0,1\}^*\times\{0,1\}^{\log(|W|)}\times M}$$

is computationally unpredictable.

Proof. For the sake of contradiction, assume that there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that breaks the computational unpredictability property of $\mathcal{F}$. This implies that

$$\Pr_{\Sigma\leftarrow\mathcal{U}_\lambda; r\leftarrow\mathcal{U}_m}\begin{bmatrix}(st, y) \leftarrow \mathcal{A}_1(\Sigma); \\ f \leftarrow \mathcal{A}_2(st, r): \\ f \in \mathcal{F} \wedge f(\Sigma, r) = y\end{bmatrix}$$

is non-negligible. We are going describe an adversary $\mathcal{A}'$ that uses $\mathcal{A}$ to break the collision resistance property of H. Given Σ, $\mathcal{A}'$ first runs $\mathcal{A}_1(\Sigma)$ and obtains a prediction y and state st. Next, $\mathcal{A}'$ randomly samples r_1, r_2 from $\mathcal{U}_m$ and runs $\mathcal{A}_2$ twice on inputs st, r_1 and st, r_2 respectively. By an application of the splitting lemma we can show that with non-negligible probability $\mathcal{A}_2$ will output (not necessarily different) functions f_1, f_2 such that $y = f_1(\Sigma, r_1) = f_2(\Sigma, r_2)$. As noted earlier, this corresponds to the hash of two chains, that due to input entropy assumption and the collision resistance of G start with different honestly mined blocks. Using similar techniques as in Lemma 21, we can show that $\mathcal{A}'$ can find a collision in H using f_1, f_2, r_1, r_2 with non-negligible probability in λ, which is a contradiction. □

For the rest of this section we will assume that the underlying POW scheme used in the Bitcoin backbone protocol is:
- complete;
- (β, ϵ)-H-TCMA with respect to any *computationally unpredictable* tampering function class;
- α-successful;
- run-time m-wise independent;
- t_{ver}-verifiable,

λ :	security parameter
n :	number of parties
$t_{\mathcal{H}}$:	number of steps per round per honest party
$t_{\mathcal{A}}$:	total number of adversarial steps per round
θ :	upper bound on the number of messages sent by the adversary per round
β :	upper bound on POW computation rate per step
γ :	lower bound on the rate of uniquely successful rounds
f :	lower bound on the rate of successful rounds
δ :	advantage from the honest majority assumption
σ :	quality of concentration of random variables in typical executions
k :	number of blocks for the common-prefix property
ℓ :	number of blocks for the chain-quality property
ε:	probability that an execution is typical

Table 1: The parameters in our analysis.

where $\epsilon \in \mathsf{negl}(t)$ and $m \geq n \log \lambda$.

Next, we prove that the adversary cannot mine blocks that extend an honest block created recently at a very high rate with probability better than that of breaking the H-TCMA property. For a summary of our notation we refer to Table 1.

Lemma 25. *For any set of consecutive rounds S and for any party P, the probability that P mined some honest block B at the first round of S such that $Z_B(S) > \beta t'_{\mathcal{A}}|S|$, is at most $\epsilon(\beta, t'_{\mathcal{A}} \cdot |S|, n \cdot |S|)$.*

Proof. Let $S = \{i' | i \leq i' < i + s\}$ and let E be the event where in $\text{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$ the adversary has mined at least $\beta t'_{\mathcal{A}} s$ blocks until round $i + s$, that descend some honest block B mined by party P at round i. For the sake of contradiction, assume that the lemma does not hold, and thus the probability that E holds is greater than $\epsilon(\beta, t'_{\mathcal{A}} s, ns)$. Using $\mathcal{A}$, we will construct an adversary $\mathcal{A}'$ that wins the H-TCMA game with probability greater than that. $\mathcal{A}'$ is going to run internally $\mathcal{A}$ and $\mathcal{Z}$, while at the same time perfectly simulating the view of honest parties using the proving oracle that he has in his disposal on the H-TCMA game. This way, the view of $\mathcal{A}, \mathcal{Z}$ will be indistinguishable both in the real and the simulated runs, and thus the probability that E happens will be the same in both cases. Moreover, the reduction will be with respect to the H-TCMA experiment with tampering function class $\mathcal{F}$. By Lemma 24, $\mathcal{F}$ is computationally unpredictable as required.

We are going to describe the two stages of $\mathcal{A}'$ separately, i.e. before and after obtaining r. First, $\mathcal{A}'_1$ creates the genesis block and sets the fixed length hash key and the POW public parameters to be Σ and pp, respectively. Then, he perfectly simulates honest parties up to round $i - 1$ and at the same time runs $\mathcal{A}$ and $\mathcal{Z}$ in a black-box way. Finally, it outputs the contents of the registers of $\mathcal{A}$ and $\mathcal{Z}$ as variable st. He can do this since he has polynomial time on λ on his disposal. Note, that up until this point in the eyes of $\mathcal{A}$ and $\mathcal{Z}$ the simulated execution is indistinguishable compared to the real one.

For the second stage, $\mathcal{A}'_2$ is again going to simulate honest parties behavior, from round i until round $i + s$, but in a different way. Instead of running the Prove algorithm for each non-corrupted honest party at every round, it makes a query to the $\mathcal{P}$ oracle with the respective parameters. Then, it checks if the honest party succeeded in making a POW in this round by comparing the number of steps needed to make this POW to the number of proving steps available to the party at this round. Hence, $\mathcal{A}'_2$ has to do n queries to the proving oracle per round. The adversary can also send up to θ messages per round to honest parties which they have to verify, thus inducing an additional $\theta \cdot t_{\mathsf{ver}}$ overhead in the simulation. Note that $\mathcal{A}'_2$ has to run the verification procedure only once per message. Moreover, $\mathcal{A}'_2$ is going to use st to reset $\mathcal{A}$ and $\mathcal{Z}$ to the same state that they were. We assume that this can be done efficiently, e.g., by having $\mathcal{A}$ and $\mathcal{Z}$ read from the registers where st is stored whenever they perform some operation on their registers.

For the reduction to work it is required that if party P generates a block B at round i, then B should be related to the randomly generated string r through some function in $\mathcal{F}$. In our scenario, r will be of the same length as the randomness needed by the input contribution function I. At round i, $\mathcal{A}'_2$ will use r to run $I(\cdot;r)$ for P. If an honest block is mined using r, then any block descending it will be related to it through some function in $\mathcal{F}$. More precisely, if $\langle s, I(x;r), w\rangle$ is the honest block mined by P, then any block descending it is of the form $\langle MD(H_k(\langle s, I(x;r), w\rangle), msg), x', w'\rangle$ for some $(msg, x', w') \in M \times \{0,1\}^\lambda \times \{0,1\}^{\log(|W|)}$, or equivalently $\langle f_{s,x,w,msg}(\Sigma, r), x', w'\rangle$. Hence, the "key" of the POW is related to $f_{s,x,w,msg}$ as required by the H-TCMA security definition.

Since $\mathcal{A}$ and $\mathcal{Z}$ cannot distinguish between the bitcoin execution and the one we described above, E will occur with probability at least $\epsilon(\beta, t'_{\mathcal{A}}s, ns)$, i.e. $\mathcal{A}$ will compute at least $\beta t'_{\mathcal{A}}s$ blocks starting from round i and up to round $i+s$ that descend B. Note, that these blocks are also valid POWs, whose keys are of the form $f(\Sigma, r)$, for (possibly different) f's. Moreover, the event that the adversary outputs different f_i, f_j such that $f_i(\Sigma, r) = f_j(\Sigma, r)$, corresponds to finding chains $\mathcal{C}_1, \mathcal{C}_2$ such that $H(\mathcal{C}_1) = H(\mathcal{C}_2)$. By Lemma 21, this happens with negligible probability. Hence, $\mathcal{A}'$ will win the H-TCMA game with probability greater than $\epsilon(\beta, t'_{\mathcal{A}}s, ns)$, while being $s \cdot (t_{\mathcal{A}} + \theta \cdot t_{\mathsf{ver}} + t_{\mathsf{bb}} \cdot n) = s \cdot t'_{\mathcal{A}}$-bounded and having made at most ns queries to the proving oracle, which is a contradiction to our initial assumption. A sketch of the reduction is given at Figure 5.

Note that we can do exactly the same reduction without using the oracle to simulate the proving procedure of the honest parties. Then, the total running time of the second stage of $\mathcal{A}'$ is at least $s \cdot (t'_{\mathcal{A}} + nt'_{\mathcal{H}})$-bounded and hence the probability he can win is $\epsilon(\beta, s \cdot (t'_{\mathcal{A}} + nt'_{\mathcal{H}}), 0)$

□

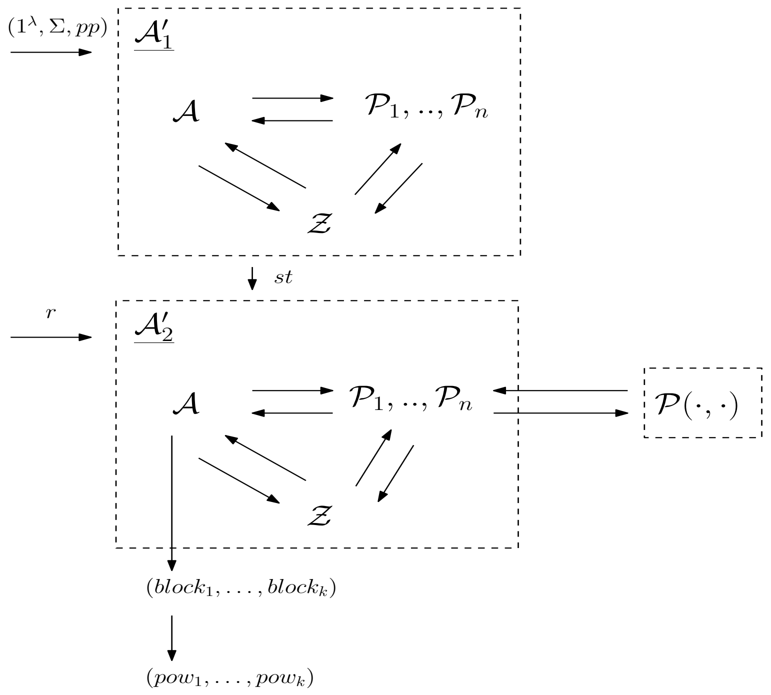
$(1^\lambda, \Sigma, pp)$
$\mathcal{A}'_1$
$\mathcal{A}$
$\mathcal{P}_1, .., \mathcal{P}_n$
$\mathcal{Z}$
st
r
$\mathcal{A}'_2$
$\mathcal{A}$
$\mathcal{P}_1, .., \mathcal{P}_n$
$\mathcal{P}(\cdot,\cdot)$
$\mathcal{Z}$
$(block_1, \ldots, block_k)$
$(pow_1, \ldots, pow_k)$

Figure 5: *The reduction from the Bitcoin backbone to the H-TCMA game of Lemma 25.*

Moreover, if in the previous proof instead of using the proving oracle provided in the H-TCMA game to simulate honest parties' work, we run their code as it is, we can derive the following bound on the total number of blocks produced by both honest and malicious parties during a certain number of rounds.

Corollary 26. *For any set of consecutive rounds S and for any party P, the probability that P mined some honest block B at the first round of S such that $Z_B(S) + X_B(S) > \beta(t'_A + nt'_{\mathcal{H}}) \cdot |S|$ is less than $\epsilon(\beta, |S| \cdot (t'_A + nt'_{\mathcal{H}}), 0)$.*

Next, we prove lower bounds on the rate of successful and uniquely successful rounds. Our proof crucially depends on the runtime independence property of the POW scheme. More specifically, the property implies that for some set of rounds the sum of the Bernoulli random variables of the event that a round is uniquely successful, concentrate around the mean. Which in turn, implies that we can lower-bound the rate of uniquely successful rounds with good probability. Note, that the model that we described at the beginning of this section is captured by the class of models mentioned in the definition of the runtime independence property.

Lemma 27. *For any set of consecutive rounds S and for any $\sigma \in (0, 1)$ it holds that:*
– *The probability that less than $\gamma \cdot |S|$ uniquely successful rounds occur in S is at most $\epsilon_s(\gamma, |S|)$;*
– *the probability that less than $f \cdot |S|$ successful rounds occur in S is at most $\epsilon_s(f, |S|)$; and*
for $\gamma = (1-\sigma)(n-t)\alpha(h, t'_{\mathcal{H}})(1-\epsilon(\beta, t'_{\mathcal{H}}, 0))^{n-t-1}$, $f = (1-\sigma)(1-(1-\alpha(h, t'_{\mathcal{H}}))^{n-t})$, and $\epsilon_s(x, |S|) = \left(\frac{(1-\sigma)^2 \log \lambda}{4|S|\sigma^2 x^2}\right)^{\lfloor \log \lambda/2 \rfloor}$.

Proof. For some fixed execution we will denote by the array $\mathbf{T}_{S\times n} = (t_{i,j}) \in \mathbb{N}^{|S|\times n}$ the number of steps each honest party takes running the Prove routine, for each round in the set S. It holds that at most t elements of each column are zero, i.e. corrupted, and the rest are lower bounded by $t'_{\mathcal{H}}$ and upper bounded by $t_{\mathcal{H}}$. W.l.o.g let $S = \{1, \dots, s\}$.

Since this lemma talks about the steps taken by the Prove function, we are going to use the almost m-wise independence property of the POW scheme, and do all the analysis on the m-wise independent random variable defined by this property. For the rest of this proof, unless explicitly stated, assume that the $Steps_{Prove}(pp, r, m, h)$ random variable refers to its idealized m-wise independent version. We first buildup some notation to help in our analysis. For $pp \in PP$, arrays $(r_{i,j}) \in R^{s\times n}$, $(msg_{i,j}) \in M^{s\times n}$ and for $h \in \mathbb{N}$ let:

- random variable $P_{i,j} = 1$ if $Steps_{Prove}(pp, r_{i,j}, msg_{i,j}, h) \leq t_{i,j}$, and 0 otherwise;
- random variable $Y_i = 1$ if $\sum_{j=1}^{n} P_{i,j} = 1$ and 0 otherwise.
- random variable $Y = \sum_{i\in[s]} Y_i$.

It easily follows from the Successful property that $\Pr[P_{i,j} = 1] \geq \alpha(h, t_{i,j})$.

Claim 1. *The random variable families $(P_{i,j})_{i\in[s],j\in[n]}$ and $(Y_i)_{i\in[s]}$ are m and $\lfloor m/n \rfloor$-wise independent respectively.*

Proof of Claim. First, notice that the m-wise independence of the scheme implies m-wise independence of $(P_{i,j})$. We will show this for two random variables and the extension to m variables will be obvious.

Let $P_1, P_2 \in (P_{i,j})$ and $x_1, x_2 \in \{0,1\}$, then

$$\begin{aligned}
\Pr[P_1 =& x_1 \wedge P_2 = x_2] = \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_1, m_1, h) \in S_1 \wedge \mathit{Steps}_{\mathsf{Prove}}(pp, r_2, m_2, h) \in S_2] \\
=& \sum_{(s_1,s_2)\in S_1\times S_2} \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_1, m_1, h) = s_1 \wedge \mathit{Steps}_{\mathsf{Prove}}(pp, r_2, m_2, h) = s_2] \\
=& \sum_{(s_1,s_2)\in S_1\times S_2} \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_1, m_1, h) = s_1] \cdot \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_2, m_2, h) = s_2] \\
=& \sum_{s_1\in S_1} \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_1, m_1, h) = s_1] \cdot \sum_{s_2\in S_2} \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_2, m_2, h) = s_2] \\
=& \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_1, m_1, h) \in S_1] \cdot \Pr[\mathit{Steps}_{\mathsf{Prove}}(pp, r_2, m_2, h) \in S_2] \\
=& \Pr[P_1 = x_1] \cdot \Pr[P_2 = x_2]
\end{aligned}$$

where S_1, S_2 are either $[0,t]$ or (t,∞) depending on x_1, x_2, and $pp, r_1, m1, r_2, m_2$ are the parameters of the random processes. We use the m-wise independence property on the third line, hence as long as the number of processes is not bigger than m the claim will follow.

Next, we prove the second point of the claim. Again, w.l.o.g we only show it for 2 random variables, Y_1, Y_2 and the extension to m/n is obvious. Let $y_1, y_2 \in \{0,1\}$, then

$$\begin{aligned}
\Pr[Y_1 = y_1 \wedge Y_2 = y_2] =& \Pr[\sum_{j\in[n]} P_{1,j} \in S_1 \wedge \sum_{j\in[n]} P_{2,j} \in S_2] \\
=& \sum_{(s_1,s_2)\in S_1\times S_2} \Pr[\sum_{j\in[n]} P_{1,j} = s_1 \wedge \sum_{j\in[n]} P_{2,j} = s_2] \\
=& \sum_{(s_1,s_2)\in S_1\times S_2} \Pr[\sum_{j\in[n]} P_{1,j} = s_1] \cdot \Pr[\sum_{j\in[n]} P_{2,j} = s_2] \\
=& \sum_{s_1\in S_1} \Pr[\sum_{j\in[n]} P_{1,j} = s_1] \cdot \sum_{s_2\in S_2} \Pr[\sum_{j\in[n]} P_{2,j} = s_2] \\
=& \Pr[Y_1 = y_1] \cdot \Pr[Y_2 = y_2]
\end{aligned}$$

where S_1, S_2 are $\{1\}$ or $\{0, 2, 3, \ldots\}$ depending on y_1, y_2. Since each variable of the family (Y_i) is an n-term sum of variables from the family $(P_{i,j})$, independence drops by a factor of n for these variables. ⊣

Claim 2. *It holds that for any $i \in S : \mathbb{E}[Y_i] \geq \gamma$*

Proof of Claim. Since $m > n$ it follows that

$$\begin{aligned}
\mathbb{E}[Y_i] =& \Pr[Y_i = 1] = \Pr[\sum_{j\in[n]} P_{i,j} = 1] \\
=& \sum_{j\in[n]} \Pr[P_{i,j} = 1] \cdot \prod_{m\in[n]\setminus\{j\}} \Pr[P_{i,m} = 0] \\
\geq& \sum_{j\in[n]} \alpha(h, t_{i,j}) \prod_{m\in[n]\setminus\{j\}} (1 - \epsilon(\beta, t_{i,m}, 0)) \\
\geq& (n-t)\alpha(h, t'_{\mathcal{H}})(1 - \epsilon(\beta, t'_{\mathcal{H}}, 0))^{n-t-1} = \gamma
\end{aligned}$$

The inequalities follow from the efficiency and H-TCMA properties. Note that $\beta t_{i,j}$ is greater than 0, hence the adversary has to compute at least one POW in the H-TCMA experiment. Also note, that in order for $\mathbb{E}[Y_i]$ to be big, α must be as big as possible and ϵ must be as small as possible. Hence, as we will see later, our goal would be to calibrate h so that it maximizes the ratio γ/β. ⊣

The lemma follows from a direct application of the following claim about the concentration of m-wise independent r.v.'s, found as a problem statement in [39], and improved here.

Claim 3. *If Y is the sum of t k-wise independent r.v.'s each of which is confined to the interval $[0,1]$ with $\mu = \mathbb{E}[Y]$ and $\epsilon > 0$, then*

$$\Pr[|Y-\mu| \geq t\epsilon] \leq \left(\frac{k}{t\epsilon^2}\right)^{\lfloor k/2 \rfloor}$$

Proof of Claim. Assume k is even. First, by the linearity of expectation it holds that:

$$\mathbb{E}[(Y-\mathbb{E}[Y])^k] = \sum_{S \subseteq [t]^k} \mathbb{E}[\prod_{i \in S}(Y_i - \mathbb{E}[Y_i])]$$

Due to the k-wise independence property, all terms in the sum where an index occurs only one time are zero, since $\mathbb{E}[Y_i - \mathbb{E}[Y_i]] = \mathbb{E}[Y_i] - \mathbb{E}[Y_i] = 0$. Hence, each index should show up twice at S. The number of possible pairs of indexes is upper bounded by $k^{k/2}$ and the number of possible assignments for a specific set of pairs is $t^{k/2}$. Thus, there are at most $(tk)^{k/2}$ non-zero terms in the sum. Moreover, each term is less than 1. Therefore

$$\mathbb{E}[(Y-\mathbb{E}[Y])^k] \leq (tk)^{k/2}$$

By Markov's inequality we get the desired result:

$$\Pr[Y - \mathbb{E}[Y] \geq t\epsilon] \leq \frac{\mathbb{E}[(Y-\mathbb{E}[Y])^k]}{(t\epsilon)^k} \leq \frac{(tk)^{k/2}}{(t\epsilon)^k} \leq \left(\frac{k}{t\epsilon^2}\right)^{k/2}$$

$\dashv$

By the linearity of expectation we have that $(1-\delta)E[Y] \geq \gamma|S|$. For $\epsilon = \delta\gamma/(1-\delta)$, $t = |S|$, $k = \log\lambda$ we get that:

$$\begin{aligned}
\Pr[Y \leq \gamma|S|] &\leq \Pr[Y \leq (1-\delta)E[Y]] \\
&\leq \Pr[|Y - E[Y]| \geq \delta E[Y]] \\
&\leq \Pr[|Y - E[Y]| \geq \delta\gamma|S|/(1-\delta)] \\
&\leq \left(\frac{(1-\delta)^2 \log\lambda}{4|S|\delta^2\gamma^2}\right)^{\lfloor \log\lambda/2 \rfloor}
\end{aligned}$$

The same fact follows with only negligible difference in probability for our scheme due to the $(\mathsf{negl}(\lambda))$-closeness property and since Y is a function of the distribution referred by the property.

Moreover, we can calculate in a similar way a bound for $X(S)$. First, note that $(1-\delta)\mathbb{E}[X_i] \geq f$. By applying the concentration bound for $\epsilon = \delta f/(1-\delta), t = |S|$ we get:

$$\begin{aligned}
\Pr[X \leq f|S|] &\leq \Pr[X \leq (1-\delta)E[X]] \\
&\leq \Pr[|X - E[X]| \geq \delta E[X]] \\
&\leq \Pr[|X - E[X]| \geq \delta|S|f/(1-\delta)] \\
&\leq \left(\frac{(1-\delta)^2 \log\lambda}{4|S|\delta^2 f^2}\right)^{\lfloor \log\lambda/2 \rfloor}
\end{aligned}$$

We note that our results also hold when some of the honest parties are corrupted, i.e. the number of steps they execute is 0. $\square$

We are now ready to define the set of *typical executions* for this setting. This strategy was also followed in [24]. However, here we will need to adapt the definition due to the difficulties associated with performing a black-box reduction to the security of the POW scheme.

Definition 28 (Typical execution)**.** An execution is *typical*[3] if and only if for any set S of consecutive rounds with $|S| \geq \lambda$, the following hold:

1. $Y(S) \geq \gamma|S|$ and $X(S) \geq f|S|$;
2. for any block B mined by an honest party at the first round of S, $Z_B(S) \leq \beta t'_{\mathcal{A}} \cdot |S|$ and $Z_B(S) + X_B(S) \leq \beta(t'_{\mathcal{A}} + nt'_{\mathcal{H}}) \cdot |S|$; and
3. no insertions and no copies occurred.

Theorem 29. *An execution is not typical with probability at most*

$$\varepsilon = \sum_{|S| \geq \eta\lambda} (\epsilon(\beta, t'_{\mathcal{A}}|S|, n|S|) + \epsilon_s(\gamma, |S|) + \epsilon_s(f, |S|) + \epsilon(\beta, |S| \cdot (t'_A + nt'_{\mathcal{H}}), 0)) \leq \mathsf{negl}(\lambda).$$

Proof. By applying the union bound on Lemmas 25 and 27 and Corollary 26 it holds that the probability that there exists any set of consecutive rounds S, where $|S| > \lambda$, and either $Y(S) < \gamma|S|$ or $X(S) < f|S|$ or the exists some honest party P that at the first round S mines block B such that, $Z_B(S) > \beta t'_{\mathcal{A}}|S|$ or $Z_B(S) + X_B(S) > \beta(t'_A + nt'_{\mathcal{H}}) \cdot |S|$, is at most

$$\begin{aligned}
\varepsilon &\leq \sum_{|S| \geq \lambda} (n\epsilon(\beta, t'_{\mathcal{A}}|S|, n|S|) + n\epsilon(\beta, |S| \cdot (t'_A + nt'_{\mathcal{H}}), 0) + \epsilon_s(\gamma, |S|) + \epsilon_s(f, |S|)) \\
&\leq \sum_{|S| \geq \lambda} [\mathsf{negl}(t'_{\mathcal{A}}|S|) + \mathsf{negl}((t'_A + nt'_{\mathcal{H}})|S|) + \left(\frac{(1-\sigma)^2 \log\lambda}{4|S|\sigma^2\gamma^2}\right)^{\lfloor \log\lambda/2 \rfloor} + \left(\frac{(1-\sigma)^2 \log\lambda}{4|S|\sigma^2 f^2}\right)^{\lfloor \log\lambda/2 \rfloor}] \\
&\leq \sum_{|S| \geq \lambda} [\mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda) + \mathsf{negl}(\lambda)] \leq \mathsf{negl}(\lambda)
\end{aligned}$$

By taking the negation of this event, properties 1 and 2 of a typical execution follow with the desired probability. Property 3 follows from Lemma 23. □

Our proof strategy is going to be based on the fact that the rate of uniquely successful rounds exceeds the rate at which the adversary produces blocks. In previous works the main security assumption was that the total running time of honest parties per round exceeds that of the adversary. More realistically, in our approach the running time of the adversary and the running time of honest parties do not have the same value, i.e., the adversary may use a superior proving algorithm. To take this into account we introduce the Honest Majority Assumption, which also depends on the security parameters of the POW scheme used. Note that γ, as defined in Lemma 27, depends on the parameters of the efficiency and H-TCMA properties.

Honest Majority Assumption. It holds that $\gamma \geq (1+\delta)\beta \cdot t'_{\mathcal{A}}$, for some $\delta \in (0,1)$.

$t'_{\mathcal{A}}$ is directly related to the computational power of the adversary. As protocol designers, our goal is to be able to tolerate $t'_{\mathcal{A}}$'s that are as big as possible. The Honest Majority Assumption implies that to achieve that we need to maximize the ratio of γ over β, i.e., the ratio of uniquely successful rounds over the rate at which adversary mines blocks. The next lemma exactly highlights this implication.

[3]In [24] parameters ϵ and η are used to parametrize the typicality of an execution. In our work parameter ϵ is absorbed in the respective definitions of γ, f and β, while we assume that η is equal to 1 for simplicity.

λ :	security parameter
n :	number of parties
$t_{\mathcal{H}}$:	number of steps per round per honest party
$t_{\mathcal{A}}$:	total number of adversarial steps per round
θ :	upper bound on the number of messages sent by the adversary per round
β :	upper bound on POW computation rate per step
γ :	lower bound on the rate of uniquely successful rounds
f :	lower bound on the rate of successful rounds
δ :	advantage from the honest majority assumption
σ :	quality of concentration of random variables in typical executions
k :	number of blocks for the common-prefix property
ℓ :	number of blocks for the chain-quality property
ε:	probability that an execution is typical

Table 2: The parameters in our analysis.

Lemma 30. *For any set S of at least λ rounds in a typical execution and for any block B mined by an honest party during S, it holds that $Z_B(S) \leq (1 - \frac{\delta}{2})Y(S)$.*

Proof.

$$Z_B(S) \leq \beta t'_{\mathcal{A}} \cdot |S| \leq \frac{1}{1+\delta}\gamma|S| < (1 - \frac{\delta}{2})Y(S)$$

The first and the last inequality follow from the typical execution assumption. The one in the middle from the Honest Majority Assumption. □

We can now use the machinery built in [24] to prove the common prefix, chain quality and chain growth properties, with only minor changes. Refer to Subsection A.6 for the details. Using these properties we prove that the modified Bitcoin backbone protocol implements a robust transaction ledger.

Theorem 13. *Assuming the existence of a common reference string, a collision-resistant hash function, a POW scheme that satisfies $(\beta, \mathsf{negl}(\lambda))$-H-TCMA and $(n \log \lambda)$-wise runtime independence with respect to any computationally unpredictable tampering function class, and model parameters $\{n, t, h, t_{\mathcal{H}}, t_{\mathcal{A}}, \theta\}$ that comply with the Honest Majority Assumption, protocol $\Pi_{\mathsf{PL}}^{\mathsf{POW}}$ implements a robust public transaction ledger with parameters $u = 4\lambda$ and $k = \lambda(f + \beta n t_{\mathcal{H}})$ except with negligible probability in λ.*

A.6 Proof of the Modified Bitcoin Backbone Protocol

The notion of a typical execution is at the core of the proof of security of Bitcoin in [24]. Here, we describe the minor changes one has to do after proving the typical execution theorem with respect to the analysis of [24], in order to prove the security of the protocol in our model. We only give brief proof sketches of lemmas or theorems from [24] that are exactly the same for our own setting.

Lemma 31. *(Chain-Growth Lemma). Suppose that at round r an honest party has a chain of length ℓ. Then, by round $s \geq r$, every honest party has adopted a chain of length at least $\ell + \sum_{i=r}^{s-1} X_i$.*

Proof. The main idea of the proof of this lemma is that, after each successful round at least one honest party will have received a chain that is at least one block longer than the chain it had, and all parties pick only chains that are longer than the ones they had. □

Theorem 32. *(Chain-Growth). In a typical execution the chain-growth property holds with parameters $\tau = f$ and $s \geq \lambda$.*

Proof. Let S be any set of at least s consecutive rounds. Then, since the execution is typical: $X(S) \geq f \cdot |S| \geq \tau \cdot |S|$. By Lemma 31, each honest player's chain will have grown by that amount of blocks at the end of this round interval. Hence, the chain growth property follows. $\square$

Lemma 33. *Let B be some honest block in a typical execution. Any sequence of $k \geq \lambda(f + \beta n t'_{\mathcal{H}})$ consecutive blocks in some chain $\mathcal{C}$, where the first block in the sequence directly descends B, have been computed in at least λ rounds, starting from the round that B was computed.*

Proof. Assume there is a set of rounds S' , such that $|S'| < \lambda$ and more than $\lambda(f + \beta n t'_{\mathcal{H}})$ blocks that descend block B have been computed. Then, there is a set of rounds S, where $|S| \geq \lambda$ such that $X_B(S) + Z_B(S) \geq |S|(f + \beta n t'_{\mathcal{H}}) \geq |S|\beta(t'_{\mathcal{A}} + n t'_{\mathcal{H}})$. This contradicts the typicality of the execution, hence the lemma follows. $\square$

Lemma 34. *(Common-prefix Lemma). Assume a typical execution and consider two chains $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $len(\mathcal{C}_2) \geq len(\mathcal{C}_1)$. If $\mathcal{C}_1$ is adopted by an honest party at round r, and $\mathcal{C}_2$ is either adopted by an honest party or diffused at round r, then $\mathcal{C}_1^{\lceil k} \leq \mathcal{C}_2$ and $\mathcal{C}_2^{\lceil k} \leq \mathcal{C}_1$, for $k \geq \lambda(f + \beta n t'_{\mathcal{H}})$.*

Proof. In Lemma 25, instead of bounding the number of blocks mined by the adversary in a set of rounds, we bound the number of blocks mined by the adversary with the additional condition that these blocks extend some specific honest block. If we also use the previous lemma, the proof is exactly the same as in [24]. Note, that all adversarial blocks in the matching between uniquely successful rounds and adversarial blocks are descendants of the last honest block in the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$. $\square$

Theorem 35. *(Common-prefix). In a typical execution the common-prefix property holds with parameter $k \geq \lambda \cdot (f + \beta n t'_{\mathcal{H}})$.*

Proof. The main idea of the proof is that if there exists a deep enough fork between two chains, then the previously proved lemma cannot hold. Hence, the theorem follows. $\square$

Theorem 36. *(Chain-Quality). In a typical execution the chain-quality property holds with parameter $\mu < 1 - \delta/2$ and $\ell \geq \lambda(f + \beta n t'_{\mathcal{H}})$.*

Proof. The main idea of the proof is the following: a large enough number of consecutive blocks will have been mined in a set rounds that satisfies the properties of Definition 28. Hence, the number of blocks that belong to the adversary will be upper bounded, and all other blocks will have been mined by honest parties. $\square$

Finally, the Persistence and Liveness properties follow from the three basic properties, albeit with different parameters than in [24].

Lemma 37. *(Persistence). It holds that Π_{PL} with $k = \lambda(f + \beta n t'_{\mathcal{H}})$ satisfies Persistence with probability at least $1 - \varepsilon$.*

Proof. The main idea is that if persistence is violated, then the common-prefix property will also be violated. Hence, if the execution is typical the lemma follows. $\square$

Lemma 38. *(Liveness). It holds that Π_{PL} with $u = 4\lambda$ rounds and $k = \lambda(f + \beta n t'_{\mathcal{H}})$ satisfies Liveness with probability at least $1 - \varepsilon$.*

Proof. The main idea here is that after u rounds at least $2k$ successful rounds will have occurred. Thus, by the chain growth lemma the chain of each honest party will have grown by $2k$ blocks, and by the chain quality property at least one of these blocks that is deep enough in the chain is honest. $\square$

A.7 Comparison with the random-oracle analysis

In this subsection, and as a sanity check, we compare the results we got from our black-box analysis to those from the random oracle analysis of [24]. To do this, we first show that the POW scheme used in the Bitcoin protocol (call it BPOW) is secure in the random oracle model according to our definitions. Then, based on the security parameters we obtain for BPOW, we show that the security guarantees we get from our analysis of the Bitcoin backbone protocol are similar to those proved in [24, 35].

In a nutshell, Bitcoin's Prove algorithm tries to find a block header with a small hash. The main components of the header are as follows: (i) the hash of the header of the previous block, (ii) the hash of the root of the Merkle tree of the transactions that are going to be added to Bitcoin's ledger, including the *randomly* created coinbase transaction, and (iii) a counter. The algorithm works by first fetching available transactions from the network, then computing a random public key that will be used for the coinbase transaction, and then iteratively incrementing the counter and calculating the hash of the header of the block until a small value is found. Casting this in our terms, the key is the hash of the previous block, which by itself depends on Bitcoin's genesis block, and the transactions received by the network as well as the coinbase transaction constitute the message. It is important to note that it is not possible to consider the key to be the coinbase transaction, as there is no guarantee it has any entropy when produced by an adversarial prover. To abstract the randomization of the proving procedure, which in the actual implementation is captured by the coinbase transaction, we hash *msg* together with a randomly generated string. This should be part of the witness in our POW syntax since it is produced by the proving process and is necessary for verification. Similarly, the counter is also part of the witness produced by the proving process. BPOW, a simplified version of the scheme described above with the transaction semantics omitted for simplicity, is presented in Figure 6.

Remark 3. In the Bitcoin implementation, the hash of the root of the Merkle tree of the transactions is not "*salted.*" This means that if we consider the adversary to be non-uniform, she could get collisions for free in her advice string and use them to compute two POWs at the cost of one. This would be problematic for our H-TCMA security game. Thus, in order to strengthen the security of the scheme, we choose to also include the key in the hash of the message.

```
BPOW: Bitcoin's POW implementation
– Gen($1^\lambda$) :
  1. return $1^\lambda$

– Prove($pp, r, msg, h$) :
  1. while(true) {
  2.     $w_1 \leftarrow \mathcal{U}_\lambda$;
  3.     $dig \leftarrow G(r, w_1, msg)$;
  4.     for $w_2 = 0^\lambda|_2$ to $1^\lambda|_2$ do {
  5.         if $(H(r, dig, w_2) < 2^\lambda - h)$
  6.             return $w_1||w_2$;  }  }

– Verify($pp, r, msg, h, w$) :
  1. return $(H(r, G(r, w_1, msg), w_2) \overset{?}{<} 2^\lambda - h)$, where $w = w_1||w_2$.
```

Figure 6: *Bitcoin's POW scheme. H and G are hash functions instantiated with SHA-256.*

We will assume that both H and G are idealized hash functions, i.e., our analysis is in the random oracle.

Theorem 39. *Let $\mathcal{F}$ be a computationally unpredictable function family. Then, for $h \in [2^\lambda(1 - e^{-1}), 2^\lambda - 1]$* BPOW *is*

- *complete;*
- *$O(\lambda)$-verifiable;*
- *$1 - \left(\frac{h}{2^\lambda}\right)^t$-successful;*
- *run-time m-wise independent, for any $m > 0$;*
- *$((1+\sigma)(1-\frac{h}{2^\lambda}), \epsilon(\beta(h), t, q_{\mathcal{P}}))$-H-TCMA secure with tampering function class $\mathcal{F}$;*

  and $\epsilon(\beta(h), t, q_{\mathcal{P}}) = \begin{cases} 1 - (\frac{h}{2^\lambda})^{t+1} & \text{, if } (1+\sigma)(1-\frac{h}{2^\lambda})t \leq 1 \\ exp(-\beta(h,t)t(1 - \frac{6}{6+(1+\log(q_{\mathcal{P}}))\delta^2}) + \log(\lambda)) & \text{, otherwise.} \end{cases}$

Proof. Let $p_h = 1 - \frac{h}{2^\lambda}$ be the probability that a query to the random oracle returns a value less than $2^\lambda - h$, and let $q_{\mathcal{H}}$ be the number of queries the adversary makes to the RO. We consider each property in turn.

Complete. By the collision resistance of G, it follows that $|\{G(r, w_1, msg)|w_1 \in \{0,1\}^\lambda\}|$ is greater than λ with overwhelming probability in λ. Hence, the probability that the Prove algorithm cannot find any witness for the given parameters is upper bounded by the probability that $\lambda \cdot 2^\lambda$ different queries to the $\mathcal{RO}$ return a value greater or equal than $2^\lambda - h$. This is upper bounded by:

$$(\frac{h}{2^\lambda})^{\lambda 2^\lambda} \leq (1 - \frac{1}{2^\lambda})^{\lambda 2^\lambda} \leq e^{-\lambda}$$

The completeness property follows.

H-TCMA. Let $\ell = \lceil \beta(h)t \rceil$. First, we show that for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{A}'$ that succeeds in winning $\mathsf{Exp}^{\text{H-CMA}}_{\mathcal{A}',\mathcal{F}}$ (Figure 1) with almost the same time complexity and probability that $\mathcal{A}$ wins, without using the proving oracle $\mathcal{P}$. $\mathcal{A}'$ is going to run $\mathcal{A}$ internally, and all calls made by $\mathcal{A}$ to $\mathcal{P}$ are going to be simulated, i.e., assuming $\mathcal{A}$ queries $\mathcal{P}$ with values (f, msg), $\mathcal{A}'$ will respond with some number t' sampled from the time distribution of $\mathcal{P}$ (t' can be efficiently sampled from a geometric distribution, since queries are i.i.d Bernoulli trials) and some random witness $w = (w_1, w_2)$, where $w_2 < t'$. $\mathcal{A}'$ is also going to store this query in some efficient data structure that allows for search in logarithmic time. Any calls made by the adversary afterwards to the RO that are related to (f, msg) will be answered accordingly; if $\mathcal{A}_2$ queries the RO with some string $f(\Sigma, r)||G(f(\Sigma, r), w_1, msg)||w_2'$, where $w_2' = w_2$, then $\mathcal{A}'$ will respond with the same value he responded on the initial query to $\mathcal{P}$, otherwise if $w_2' < w_2$, he responds by $2^\lambda - h + (y \mod h)$, where y is the output of the real RO in this query. Note, that since w_1 is chosen at random and the fact that the RO is unpredictable, the probability that $\mathcal{A}$ has queried the RO with a string of this format before querying $\mathcal{P}$ is negligible. Hence, the view of $\mathcal{A}$ in both experiments is computationally indistinguishable, and he will output ℓ valid POWs with respect to the simulated view with the same probability that he wins in the real experiment.

It could be the case that the output of $\mathcal{A}$ contains a POW related to the queries asked to (the simulated) oracle $\mathcal{P}$, and thus it does not correspond to a winning output for $\mathcal{A}'$, i.e., there exists a query $\langle (f, msg), (w_1, w_2) \rangle$ on $\mathcal{P}$ and a POW on the output of the form $\langle f', msg', (w_1', w_2') \rangle$ such that $G(f(\Sigma, r), w_1, msg) = G(f'(\Sigma, r), w_1', msg')$ and either $msg \neq msg'$ or $w_1 \neq w_1'$. This implies that the adversary has found a collision in G, which only happens with negligible probability in λ. Hence, $\mathcal{A}'$'s will win $\mathsf{Exp}^{\text{H-CMA}}_{\mathcal{A}',\mathcal{F}}$ with the same probability (minus some negligible term in λ) as $\mathcal{A}$. Moreover, the overhead incurred to $\mathcal{A}'$'s running time will be only logarithmic on $q_{\mathcal{P}}$ i.e. $\mathcal{A}'$ can simulate the t steps taken by $\mathcal{A}_2$ in time $t \cdot (1 + \log(q_{\mathcal{P}}))$; he has to maintain a heap of the queries made to $\mathcal{P}$ and search it each time the RO is queried.

Let A_ℓ be the event where $\mathcal{A}$ asks t queries the RO after querying $\mathcal{R}$, and receives at least ℓ responses that have value less than $2^\lambda - h$. Let random variable X be equal to the number of these responses that are less than $2^\lambda - h$. Since the queries are i.i.d. Bernoulli random variables with probability of success p_h, we can use the Chernoff bound to bound the probability of A. By setting ℓ equal to $(1+\delta)p_h t$ it follows that, for any $\delta \in (0,1)$:

$$\Pr[A] = \Pr[X \geq \ell] = \Pr[X \geq (1+\delta)p_h t] = \Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq e^{-\frac{\mathbb{E}[X]\delta^2}{3}} \leq e^{-\frac{\ell\delta^2}{6}}.$$

For the special case where $\ell \leq 1$ we can derive a stricter bound for the probability of A as follows:

$$\Pr[A] = \Pr[X \geq 1] = 1 - (1-p_h)^t$$

Let B be the event where $\mathcal{A}_2$ outputs f, m, w such that $f \in \mathcal{F}$ and there exists a query made to the random oracle by $\mathcal{A}_1$ of the form $f(\Sigma, r)||x||w$. Assume that B happens with non-negligible probability. Then, we can use $\mathcal{A}$ to break the computational unpredictability of $\mathcal{F}$. Let $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ be the attacker in the computational unpredictability game. $\mathcal{A}'_1$ will first run $\mathcal{A}_1(1^\lambda, \Sigma, pp)$. It will output st and y, the prefix of a random query that $\mathcal{A}_1$ made to the RO with length equal to the size of the images of functions in $\mathcal{F}$. Then, $\mathcal{A}'_2$ will run $\mathcal{A}_2(1^\lambda, r, st)$ until it halts and possibly outputs a number of POWs. Since $\mathcal{A}$ is a PPT algorithm, the number of queries made to the RO is at most polynomial in number. Hence, with non-negligible probability B will occur, and y will be the prefix of the RO query that matches the key of the POW output by $\mathcal{A}$. This violates the computational unpredictability property, and hence B only occurs with negligible probability.

Let C be the event where the adversary wins and outputs two distinct POWs that correspond to the same query to the RO. This implies that the adversary can find a collision on G. In time $L = t + t_{\mathsf{pre}}$ polynomial in λ, the probability that $\mathcal{A}$ finds a collision is $\binom{L}{2}2^{-\lambda+1} = e^{-\Omega(\lambda)} = \mathsf{negl}(\lambda)$.

Finally, let D_i be the event where the adversary outputs i POWs that he has not asked the RO, and E_i be the event where i new queries to the RO return values less than $2^\lambda - h$. It holds that $\Pr[E_i] = p_h^i$. Note, that if A_ℓ, B, C do not occur and D_0 occurs, it is implied that $\mathcal{A}'$ will lose in the experiment. Thus:

$$\begin{aligned}
\Pr[\mathsf{Exp}_{\mathcal{A},\mathcal{F}}^{\text{H-TCMA}}(1^\lambda, h, \ell) = 1] &= \sum_{i=0}^{\ell} \Pr[\mathsf{Exp}_{\mathcal{A},\mathcal{F}}^{\text{H-TCMA}}(1^\lambda, h, \ell) = 1 \wedge D_i] \\
&\leq \sum_{i=0}^{\ell} \Pr[(A_{\ell-i} \vee B \vee C) \wedge E_i \wedge D_i] \\
&= \sum_{i=0}^{\ell} \Pr[(A_{\ell-i} \vee B \vee C) \wedge D_i] \Pr[E_i] \\
&\leq \sum_{i=0}^{\ell} \Pr[(A_{\ell-i} \vee B \vee C)] \Pr[E_i] \\
&\leq \begin{cases} 1 - (1-p_h)^{t+1} + \mathsf{negl}(\lambda), & \text{when } (1 - \frac{h}{2^\lambda})t \leq 1 \\ \sum_{i=0}^{\ell} (e^{-\frac{(\ell-i)(1+\log(q_{\mathcal{P}}))\delta^2}{6}} + \mathsf{negl}(\lambda)) p_h^i, & \text{otherwise} \end{cases} \\
&\leq \begin{cases} 1 - (1-p_h)^{t+1} + \mathsf{negl}(\lambda), & \text{when } (1 - \frac{h}{2^\lambda})t \leq 1 \\ e^{-\ell(1-\frac{6}{6+(1+\log(q_{\mathcal{P}}))\delta^2}) + \log(\ell)} + \mathsf{negl}(\lambda), & \text{otherwise} \end{cases}
\end{aligned}$$

For the first equality we use the law of total probability for the events $\{D_i\}_{i \in \{0,\ldots,\ell\}}$. The second and fourth inequalities follow from the fact that E_i is independent of any of the other events in the conjunction and the bounds we have set for h, respectively.

Verifiability. Assuming H and G take constant time, verification takes time $c_{\mathsf{ver}}\lambda$, for some small constant c_{ver} which can be easily computed by careful inspection of the verification protocol.

Efficiency. For any $t \in \mathbb{N}$, $r \in R, msg \in M$ and $h \in \mathbb{N}$ it holds that:

$$\Pr\left[\mathsf{Steps}_{\mathsf{Prove}}(r, msg, h) < t\right] = 1 - (1 - p_h)^t$$

Independence. Let $\{Y_i\}_{i \in I}$ be the same as $\{X_i\}_{i \in I}$ with the only difference that the random oracle is replaced with a random function, i.e., every time Prove is called and the oracle H is queried it generates a random output. Obviously the random variables in $\{Y_i\}_{i \in I}$ are mutually independent, since their output only depends on their own local coins.

Regarding the second property, let E be the event that all w_1 sampled are different among all the invocations of Prove, and that no collisions occurs in G. Note, that in any polynomially bounded execution this event happens with overwhelming probability in λ. Moreover, in any execution that E occurs, $\{X_i\}_{i \in I}$ and $\{Y_i\}_{i \in I}$ are equal, since the random oracle behaves exactly as the random function we have replaced it with. Therefore, if $p(\cdot)$ is a polynomial that upper bounds the duration of the execution, it holds that for any $z \in [p(\lambda)]^{|I|}$

$$\begin{aligned}
&\Pr[\{X_i\}_{i \in I} = z] - \Pr[\{Y_i\}_{i \in I} = z] = \\
&= \Pr[\{X_i\}_{i \in I} = z|E]\Pr[E] + \Pr[\{X_i\}_{i \in I} = z|\neg E]\Pr[\neg E] \\
&\quad - \Pr[\{Y_i\}_{i \in I} = z|E]\Pr[E] - \Pr[\{Y_i\}_{i \in I} = z|\neg E]\Pr[\neg E] = \\
&= (\Pr[\{X_i\}_{i \in I} = z|\neg E] - \Pr[\{Y_i\}_{i \in I} = z|\neg E])\Pr[\neg E]
\end{aligned}$$

Hence, it follows that the two distributions are $\mathsf{negl}(\lambda)$-close:

$$\begin{aligned}
2\Delta[\{X_i\}_{i \in I}, \{Y_i\}_{i \in I}] &= \sum_z |\Pr[\{X_i\}_{i \in I} = z] - \Pr[\{Y_i\}_{i \in I} = z]| \\
&\leq \sum_z |(\Pr[\{X_i\}_{i \in I} = z|\neg E] - \Pr[\{Y_i\}_{i \in I} = z|\neg E])\Pr[\neg E]| \\
&\leq \Pr[\neg E]\sum_z |(\Pr[\{X_i\}_{i \in I} = z|\neg E] - \Pr[\{Y_i\}_{i \in I} = z|\neg E])| \\
&\leq \mathsf{negl}(\lambda)(\sum_z \Pr[\{X_i\}_{i \in I} = z|\neg E] + \sum_z \Pr[\{Y_i\}_{i \in I} = z|\neg E]) \leq \mathsf{negl}(\lambda)
\end{aligned}$$

The last inequality follows from the fact that each of the sums should be less or equal to 1, as the events described are disjoint and their union covers the entire sample space. □

Since parameter ϵ of the H-TCMA property of `BPOW` is negligible in t and the scheme is m-wise independent for any $m > 0$, we can use Theorem 39 and obtain meaningful bounds for the γ, f quantities introduced in the previous subsection.[4] These quantities are important since γ determines how powerful the adversary our system can handle can be, and f is related to how fast blocks are produced. Both of them also appear in [24] and are (expressed in the language of our model) equal to

$$\gamma_{\mathsf{RO}} = (1-\sigma)(n-t)\left(1 - \frac{h}{2^\lambda}\right)t'_{\mathcal{H}}\left(\frac{h}{2^\lambda}\right)^{((n-t)-1)t'_{\mathcal{H}}}, f_{\mathsf{RO}} = (1-\sigma)(1 - \left(\frac{h}{2^\lambda}\right)^{(n-t)t'_{\mathcal{H}}})$$

[4]Earlier, in order to make our presentation easier to follow, we chose to allow the `BPOW` Prove routine to run until it finds a solution. However, in the Bitcoin backbone protocol it should be stopped after running a specific number of steps. We can implement this modification using a step counter, without affecting any of the security properties of the scheme.

respectively. The next lemma shows that our bounds are close[5] to the ones calculated there. Moreover, our analysis also reveals other factors that affect security such as the verification time, the number of messages the adversary sends to the honest parties, as well as the cost of reducing the security to that of the POW primitive. Let $\Pi_{\mathsf{PL}}^{\mathtt{BPOW}}$ denote the Bitcoin protocol using BPOW. Then:

Lemma 40. *For $\Pi_{\mathsf{PL}}^{\mathtt{BPOW}}$, it holds that $\frac{\gamma}{\gamma_{\mathsf{RO}}} \geq \left(\frac{h}{2^\lambda}\right)^{n-t-1}$ and $f \geq f_{\mathsf{RO}}$.*

Proof. For γ we have:

$$\begin{aligned}
\frac{\gamma}{\gamma_{\mathsf{RO}}} \geq& \frac{(1-\delta')(n-t)(1-\left(\frac{h}{2^\lambda}\right)^{t'_{\mathcal{H}}})(1-(1-\left(\frac{h}{2^\lambda}\right)^{t'_{\mathcal{H}}+1}))^{n-t-1}}{(1-\delta')(n-t)\left(1-\frac{h}{2^\lambda}\right)t'_{\mathcal{H}}\left(\frac{h}{2^\lambda}\right)^{((n-t)-1)t'_{\mathcal{H}}}} \\
\geq& \frac{(1-\left(\frac{h}{2^\lambda}\right)^{t'_{\mathcal{H}}})(1-(1-\left(\frac{h}{2^\lambda}\right)^{t'_{\mathcal{H}}+1}))^{n-t-1}}{\left(1-\frac{h}{2^\lambda}\right)t'_{\mathcal{H}}\left(\frac{h}{2^\lambda}\right)^{((n-t)-1)t'_{\mathcal{H}}}} \\
\geq& \frac{(\left(\frac{h}{2^\lambda}\right)^{t'_{\mathcal{H}}+1})^{n-t-1}}{\left(\frac{h}{2^\lambda}\right)^{(n-t-1)t'_{\mathcal{H}}}} \geq \left(\frac{h}{2^\lambda}\right)^{n-t-1}
\end{aligned}$$

The first inequality follows from the honest majority assumption. Also, w.l.o.g assume that $t'_{\mathcal{A}} \geq t'_{\mathcal{H}}$, i.e. the adversary has at least the computational power of one honest party. Then:

$$(1+\delta')(1-\frac{h}{2^\lambda})t'_H = \beta t'_{\mathcal{H}} \leq \beta t'_{\mathcal{A}} < \gamma \leq 1$$

On the other hand, for f we have that:

$$f = (1-\delta)(1-(1-(1-\left(\frac{h}{2^\lambda}\right)^{t'_{\mathcal{H}}}))^{n-t}) = (1-\delta)(1-\left(\frac{h}{2^\lambda}\right)^{(n-t)t'_{\mathcal{H}}})$$

□

Hence, $\Pi_{\mathsf{PL}}^{\mathtt{BPOW}}$ implements a robust transaction ledger with overwhelming probability in λ and with bounds comparable to those in [24].

B Consensus from Proofs of Work without Random Oracles (cont'd)

In this section we describe the new consensus protocol—in particular the modifications to the content validation predicate from [24]—in detail. To overcome the challenge of preventing the honest parties' chains to grow too large, we change POW generation (Algorithm 1, line 8) and POW verification (Algorithm 2, line 8) so that they are run on the header of the block, i.e., $\mathsf{Prove}(s, G(x)||vote, h)$ and $\mathsf{Verify}(s, G(x)||vote, h, w)$, respectively. This way we are able to verify the validity of a block as a POW and determine the block's vote by only knowing its header. This is exactly the properties we need for the consensus application.

To efficiently check for membership in the hash tree, in line 2 we use an AVL tree [1]. (Any other data structure supporting efficient updates and search would also work.) In line 5 the referenced blocks are extracted and pushed into a queue. We note that during this process it is also checked that the contents of the block have a correct format, i.e., a vote field and list of block headers, and that the order in which the headers are organized in the block is the same as the order in which they are popped from the queue.

[5]The probability of failure for bitcoin currently is approximately $\frac{h}{2^\lambda} \approx 1 - 2^{-72}$.

Algorithm 5 The *content validation predicate.* The input is the contents of the blocks of some chain.

```
function $V(\langle x_1, \ldots, x_m \rangle)$
    $D \leftarrow$ new AVL()                                  ▷ Create a new (empty) AVL tree.
    $D.add(H(\mathsf{Gen}))$                                   ▷ Add the hash of the genesis block on the tree.
    for $i = 1, ..., m$ do
        $queue \leftarrow$ references$(x_i)$                   ▷ Add all block references in a queue.
        while $queue \neq \emptyset$ do
            $\langle s, G(x)||vote, w \rangle \leftarrow queue.top()$
            if $((D.exists(s)) \wedge \mathsf{Verify}(s, G(x)||vote, h, w))$ then
                $D.add(H(\langle s, G(x)||vote, w \rangle))$   ▷ Add new entry on the tree.
                $queue.pop()$
            else
                return False                                    ▷ If not, the chain is invalid.
            end if
        end while
    end for
    return True
end function
```

The algorithm is run for $L = O(\lambda)$ rounds, after which it outputs the majority of the votes found in a prefix of the selected chain, of a predetermined length M. We call the resulting protocol $\Pi_{\mathsf{BA}}^{\mathtt{POW}}$ ("BA" for Byzantine agreement, to be consistent with [24].).

Theorem 15. *Assuming the existence of a common reference string, a collision-resistant hash function, a* POW *scheme that satisfies* $(n \log \lambda)$*-wise runtime independence and* $(\beta, \mathsf{negl}(\lambda))$*-H-TCMA with respect to any computationally unpredictable tampering function class, and model parameters* $\{n, t, h, t_{\mathcal{H}}, t_{\mathcal{A}}, \theta\}$ *that comply with the Honest Majority Assumption, protocol* $\Pi_{\mathsf{BA}}^{\mathtt{POW}}$ *solves consensus in* $O(\lambda)$ *rounds with overwhelming probability.*

Proof. We are going to show that protocol $\Pi_{\mathsf{BA}}^{\mathtt{POW}}$, parameterized with $k \geq \lambda(f + \beta n t'_{\mathcal{H}})$, $L \geq 2k \frac{1}{1-(1-\frac{\delta}{2})\frac{\gamma^2}{f}}$ and $M = (\gamma \cdot L - k)$ solves consensus with overwhelming probability in λ.

First, note that for any chain $\mathcal{C}$ where the head of the chain is mined by an honest party it holds that $V(\mathcal{C})$ is equal to True. This follows from the modifications we did on the protocol and the fact that the adversary will only include headers on his list of references that satisfy the requirements of predicate $V(\cdot)$. Moreover, since the output of the predicate only depends on the chain that is being validated, if one honest party accepts a chain as valid, all honest parties accept.

Next, we prove Agreement. Assume that an execution is typical. Due to the chain growth property, after L rounds the chain of honest parties will have length at least $\gamma \cdot L$ blocks, and due to the common prefix property they will all agree on the first $\gamma \cdot L - k$ blocks. Hence, all honest parties will decide on their output value based on the "votes" mentioned in each block header that is referenced in these blocks, and they will all agree on the same value.

Regarding Validity, we are going to show that the majority of the counted "votes," i.e., from blocks and block headers found in blocks $B_1, \ldots, B_{\gamma \cdot L - k}$ of the selected chain, have been mined by honest parties. By the chain quality property, at least one block from $B_{\gamma \cdot L - 2k}, \ldots, B_{\gamma \cdot L - k}$ is honest. Moreover, by chain growth the blocks preceding this honest block must have been computed up to round $\frac{L-2k}{\gamma}$. Hence, all honest blocks mined up to this round will contribute votes to the agreement process. On the other hand, the last block in the sequence of $\gamma \cdot L - k$ blocks was computed at most at round L, due to chain growth and common prefix. Hence, it remains to show that for $S_1 = \{1, \ldots, \frac{L-2k}{\gamma}\}$ and $S_2 = \{1, \ldots, L\}$ it holds that $Z(S_2) < X(S_1)$:

$$Z(S_2) \leq \beta t'_{\mathcal{A}} L < \frac{1}{1+\delta}\gamma L \leq (1 - \frac{\delta}{2})\gamma L \leq f\frac{L-2k}{\gamma} \leq X(S_1).$$

The first and the last inequalities hold due to the fact that the execution is typical. The fourth inequality follows from our assumption about L. The theorem follows since the majority of the referenced blocks in the chain agreed upon, have been mined by honest parties.

Concluding, notice that the total size of any chain is bounded by the total number of blocks mined, since each block's header is mentioned at most once in a single chain. Hence, in r rounds of a typical execution a chain has size at most $O(r \cdot \beta(t'_A + nt'_{\mathcal{H}}) \cdot \lambda)$ bits. $\square$