Proofs of Work for Blockchain Protocols

Juan A. Garay[*] Aggelos Kiayias[†] Giorgos Panagiotakos[†]

August 9, 2017

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 security for honest provers (termed "uniquely successful under chosen key and message attacks"). Interestingly, our formulation is incomparable with previous formulations of POWs that applied the primitive to contexts other than the blockchain. Our result paves the way for proving the security of blockchain protocols in the standard model assuming our primitive can be realized from computational assumptions.

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 [16], sybil attacks [15], denial of service protection [24, 3], and other applications. A POW scheme can be typified as a primitive that entails two 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 verification algorithm, on the other hand, will need to use similar parameters as the proving algorithm in order to verify the POW.

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*, which when appropriately used, bypasses the previous restriction, and may even allow producing a POW with effort independent of the given difficulty level, (iii) POW *sampling*, which dictates that one can randomly sample POWs,

[*]Yahoo Research, garay@yahoo-inc.com.

[†]University of Edinburgh, {akiayias,giorgos.pan}@inf.ed.ac.uk.

and (iv) *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.

Arguably, the most impactful application of POWs to date is in the design of “permissionless” blockchain protocols [27], where the use of POWs enables the construction of distributed consensus protocols that are not based on a pre-existing public-key infrastructure or authenticated channels. At a very high level, the way POWs help in this context is by slowing down message passing 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 analysis of the Bitcoin protocol that has been performed so far (specifically, [19, 26, 29]), these works have refrained from actually formally defining the underlying POW algorithms as a stand-alone cryptographic primitive and relied instead on the random oracle (RO) model [9] to prove *directly* the properties of the blockchain protocol, as opposed to following a reduction approach to the underlying cryptographic primitive. The same is true for other provably secure POW-based distributed protocols [1, 25, 21]. In fact, to our knowledge, no prior work has identified a set of properties for the POW primitive that are sufficient to imply the security of a blockchain protocol.

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 for reducing in a black-box manner the security of the Bitcoin backbone protocol [19] 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 POW properties we identify that imply the security of the Bitcoin blockchain yield a POW primitive that is *incomparable* with the previous formalizations for other POW-related tasks. In more detail, our contributions are as follows.

Formalization of the POW primitive. Our syntax of the POW primitive entails a proving and verification algorithm, Prove and Verify, respectively. Prove is invoked on an input (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 (r, msg, h, w) where w is (possibly) an output of Prove. There are four properties that we put forth for our primitive. First, as in the case of an interactive proof, we require a *Completeness* property: Proofs produced by Prove should be acceptable by the Verify algorithm. Second, t-Verifiability mandates that the POW can be verified in t steps by the honest verifier (and this can be set to be faster than the running time of Prove). The third property deals with security against malicious provers, and is called *Hardness against Tampering and Chosen-Message Attacks* (H-TCMA). It captures the fact that producing a sequence of POWs, for chosen messages, even while tampering with their keys, does not provide an advantage to an adversary in terms of running time. Furthermore, the property should hold true even in the presence of a Prove oracle (excluding of course any proofs produced by the oracle from the adversary’s output). The fourth and final property deals with security for honest provers: We call a POW *Uniquely Successful against Chosen Key and Message Attacks* (US-CKMA) if in a number of experiments involving the parallel execution of n instances of Prove, the rate of experiments where an honest prover will be the sole POW producer is high. Further, we require this property to hold even when the parallel Prove instances are executed with keys that are maliciously generated.

Security of the Bitcoin backbone assuming POW. We then turn to the analysis of the Bitcoin backbone protocol given our POW primitive. First, we show how Bitcoin’s POW implementation realizes our POW primitive in the random oracle model. This is performed as a sanity check to ensure the realizability of our primitive in an idealized model. We then proceed to the proof of the Bitcoin protocol for a fixed number of parties in the standard model assuming our POW, following the approach of [19][1]. 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. For the proof, we identify an additional assumption for the security theorems to go through regarding the collision and preimage resistance of the underlying hash function that "glues" the blocks together in the blockchain. Subsequently, we show that using the H-TCMA property and our pre-conditions regarding adversarial hashing power, 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 US-CKMA property, it follows that "uniquely successful rounds" happen with sufficient density in any sufficiently long time window. Using these two core lemmas we revisit the security proof of [19] and show how it can be ported to the standard model.

Our POW notion vis-à-vis *previous related notions.* Finally, we compare our POW notion to existing related notions, where the notion of *sampleability* of POW instances has played an instrumental role [13, 5]. In constrast, not only we do not need this property, but we show how it can be inconsistent with our H-TCMA notion. In addition, we prove that H-TCMA implies amortization resistance.

Prior and related work. Dwork and Naor [17] first considered POWs under the term "pricing functions," as a means of protection against spam 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 one 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 [23] and Back [2, 3] use POWs to construct electronic payment systems. In [23] 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 [2, 3] 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.* [12] 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 a non-amortizing languages in the worst case.

Another interesting approach is that of Ball *et al.* [6], 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.

[1]Refer to [20] for an analysis of Bitcoin in the dynamic setting.

Relating to the second part of this paper, i.e., proving that Bitcoin's security can be reduced to an assumption on the underlying POW scheme, Poelstra [30] 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.

Organization of the paper. The basic computational model, definitions and cryptographic building blocks are presented in Section 2. Formal definition of a POW and its security properties are presented in Section 3. Sections 4 and 5 are dedicated to Bitcoin: Analyzing Bitcoin's POW implementation according to our definition, and showing how the Bitcoin blackbone protocol should be modified to incorporate our POW primitive and reducing its security to it, respectively. Finally, Section 6 compares our POW notion to existing related notions.

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 [8, 10, 22, 11] 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. [11]; 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.* [22]; refer to [22] for details on using interactive RAMs in a UC-like framework, as well as to Section 5.1. 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 notions of security for cryptographic hash functions (see, e.g., [31] for a thorough review):

Definition 1. Let $\mathcal{H} = \{H_\lambda : K_\lambda \times M_\lambda \to Y_\lambda\}_{\lambda \in \mathbb{N}}$ be a hash-function family, and $\mathcal{A}$ be a PPT adversary. Then $\mathcal{H}$ is

- *collision resistant* iff for any $\lambda \in \mathbb{N}$ and corresponding $H : K \times M \to Y$ in $\mathcal{H}$

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

- *everywhere preimage resistant* iff for any $\lambda \in \mathbb{N}$ and corresponding $H : K \times M \to Y$ in $\mathcal{H}$

$$\Pr[(y, s) \leftarrow \mathcal{A}(1^\lambda); k \overset{\$}{\leftarrow} K; m \leftarrow \mathcal{A}(1^\lambda, k, s) : H(k, m) = y] \leq \mathsf{negl}(\lambda).$$

As we will see, POW schemes critically depend on a key that ensures the proof is "fresh". Nevertheless, in many settings it is useful to allow the adversary to tamper this key with a certain function f. Given that arbitrary tampering may eliminate freshness, we have to suitably restrict the tampering function class. The following class of functions will be useful in our security analysis.

Definition 2. Let $\mathcal{F} = \{f_{i,\lambda}\}_{i,\lambda\in\mathbb{N}}$ be a function family such that $f_{i,\lambda} : \{0,1\}^{p(\lambda)} \to \{0,1\}^{\lambda}$, for some function $p : \mathbb{N} \to \mathbb{N}$, for all $i, \lambda \in \mathbb{N}$. We say that $\mathcal{F}$ is a *unpredictability-preserving* function family iff for any uniform PPT RAM machine $\mathcal{A}$, the following holds: $\Pr[(y, s) \leftarrow \mathcal{A}(1^{\lambda}); x \overset{\$}{\leftarrow} U_{m(\lambda)}; f \leftarrow \mathcal{A}(1^{\lambda}, y, s, x) : f \in \mathcal{F} \land f(x) = y] \leq \mathsf{negl}(\lambda)$.

Observation 3. The identity function $\{\mathsf{id}_{\lambda}(\cdot)\}_{\lambda\in\mathbb{N}}$ and an everywhere preimage resistant hash function family (Definition 1; see also Appendix A) are unpredictability-preserving.

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. Later on, in Section 6, we make a comparison between some of the POW properties introduced here and other properties that have appeared in the literature for related primitives.

POW syntax. We formalize the POW notion as follows. Given a specific security parameter λ, let 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 regarding the significance of these input elements.

Definition 4. A POW scheme consists of two algorithms $\mathsf{POW} = (\mathsf{Prove}, \mathsf{Verify})$ where:
- $\mathsf{Prove}(r, msg, h)$ is a randomized algorithm that takes as input 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 as the POW.
- $\mathsf{Verify}(r, msg, h, w)$ is a deterministic algorithm that takes as input 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 [12, 4], 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., there is no way to guarantee freshness), 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 POW's across sessions or to different 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 $r \in R, h \in \mathbb{N}$, and $msg \in M$:

$$\Pr\left[w \leftarrow \mathsf{Prove}(r, msg, h) : \quad w \neq \bot \Rightarrow \mathsf{Verify}(r, msg, h, w) = \mathsf{true} \right] = 1.$$

Next we require that verification time is bounded by a parameter t.

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

Next we need to capture how the adversary can subvert an honest verifier and an honest 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 interactive proofs, As in this 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 create POW's 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. We call this *hardness against tampering and chosen message attack* (H-TCMA). The game defining the attack is shown in Figure 1. In order to capture the interactions that the adversary may have with different parties, we allow the adversary to have access to a Prove oracle, essentially allowing him to simulate any scenario where he interacts with other parties that use the same POW scheme. A subtle point in the modeling of security in the presence of such oracle is that the Prove oracle 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 may influence the adversarial strategy; in order to preserve this dependency on this side-channel we will require from the Prove oracle to leak this information. In addition, we require that the keys used by the adversary to construct proofs should 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 must be generated by some oracle $\mathcal{R}()$ that samples values from a uniformly random distribution and is *external* to the adversary. Hence, the adversary will only know the value after he has queried $\mathcal{R}()$, thus providing us with a point of reference before which this value cannot be predicted. Furthermore we allow the adversary to tamper the POW keys by manipulating them via applying tampering functions that belong to a function family $\mathcal{F}$.

More formally, the adversary will have oracle access to $\mathcal{R}$, which is a sampling oracle of the uniform distribution. We use $\mathsf{Ans}_{\mathcal{R}}$ to denote the set of responses of oracle $\mathcal{R}$. The adversary will also have access to $\mathcal{P}(\cdot,\cdot,\cdot)$, a proof-of-work oracle that on input (f, r, msg), where $f \in \mathcal{F}$, r belongs to $\mathsf{Ans}_{\mathcal{R}}$, and msg is some message, returns the pair (w, t) where w is the output of $\mathsf{Prove}(f(r), msg, h)$ and t is the number of steps taken by the Prove algorithm on these parameters. Function $\mathsf{Asked}(r', msg, w)$ is true if w was a response of $\mathcal{P}$ to some query (f, r, msg), where $f(r) = r'$.

```
                 $\mathsf{Exp}^{\text{H-TCMA}}_{\mathcal{A},\mathcal{F}}(1^\lambda, h, k)$
1.   $\{(r_1, f_1, msg_1, w_1), \ldots, (r_k, f_k, msg_k, w_k)\} \leftarrow \mathcal{A}^{\mathcal{P}(\cdot,\cdot,\cdot),\mathcal{R}()}(1^\lambda, h, k)$;
2.   output $\bigwedge_{i=1}^{k}(\mathsf{Verify}(f_i(r_i), msg_i, w_i) \wedge \neg\mathsf{Asked}(f_i(r_i), msg_i, w_i) \wedge r_i \in \mathsf{Ans}_{\mathcal{R}} \wedge f_i \in \mathcal{F})$.
```

Figure 1: *The Hardness against Tampering and Chosen-Message Attack experiment for a POW scheme. $\mathcal{R}(\cdot)$ is a sampling oracle of the uniform distribution. $\mathcal{F}$ is the class of tampering functions.*

We are now ready to formulate the security property of Hardness against Tampering and Chosen Message Attacks (H-TCMA). It is parameterized by two values β, ϵ, and informally, it states that no adversary $\mathcal{A}$ exists that can participate in the experiment of Figure 1 so that $\mathcal{A}$ takes at most t steps after querying $\mathcal{R}$ for the first time and it produces $k \geq \beta \cdot t$ POWs with probability better than ϵ. Note that in total we allow any polynomial number of steps to $\mathcal{A}$, 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. We say that a POW scheme has (β, ϵ)*-Hardness against Tampering and Chosen-Message Attacks* (H-TCMA), with tampering function class $\mathcal{F}$, if for every polynomial $t_{\mathsf{pre}}(\cdot), t(\cdot)$, for any $h \in \mathbb{N}$, and every adversary $\mathcal{A}$, where $\mathcal{A}$ is t_{pre}-bounded and starting from the query on $\mathcal{R}$ takes t steps and makes at most $q_{\mathcal{P}}$ queries to oracle $\mathcal{P}$, the probability of $\mathcal{A}$ winning in $\mathsf{Exp}^{\text{H-TCMA}}_{\mathcal{A},\mathcal{F}}(1^\lambda, h, \beta(h) \cdot t)$ (Figure 1) is less than $\epsilon(\beta(h), t, q_{\mathcal{P}})$.

Finally, we consider the setting of subverting the 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. In more details, a POW scheme is *uniquely successful* (borrowing the name from the event where only one honest party generates a POW in [19]) *against chosen key and message attacks* (US-CKMA) when the rate of uniquely successful POW outcomes in a sequential execution of s experiments, each one involving n honest proving processes running in parallel for t steps, is γ except with probability ϵ. We allow γ to depend on n, h, t while ϵ depends on γ and s. Formally we have the following.

Definition 8. We say that a POW scheme is (γ, ϵ)*-uniquely successful against chosen key and message attacks* (US-CKMA), if for any polynomial functions n, s, for the arrays $\mathbf{T} = (t_{i,j}) \in \mathbb{N}^{s \times n}, (r_{i,j}) \in R^{s \times n}, (msg_{i,j}) \in M^{s \times n}$ and for $h \in \mathbb{N}$ it holds that:

$$\Pr\left[\sum_{i=1}^{s} b_i < \gamma(n, \mathbf{T}, h) \cdot s \text{ where } \forall i \in [s] : b_i = (\exists! j \in [n] : \mathsf{Steps}_{\mathsf{Prove}}(r_{i,j}, msg_{i,j}, h) < t_{i,j}) \right]$$
$$\leq \epsilon(\gamma(n, \mathbf{T}, h), s).$$

4 Bitcoin's POW Construction

In this section we analyze Bitcoin's POW construction according to the security properties presented in Section 3. Our formulation of Bitcoin's POW, which we will call `BPOW`, is going to be based on Bitcoin's POW implementation[2], which itself is inspired by Hashcash's design [2].

In a nutshell, Bitcoin's Prove algorithm tries to find a block header with a small hash. The header consists mainly of the following components: (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 getting 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 "freshness" parameter is the hash of the previous block, which by itself depends on Bitcoin's genesis

[2]As described in `https://en.bitcoin.it/wiki/Protocol_documentation`, `https://en.bitcoin.it/wiki/Block_hashing_algorithm`.

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: there is no guarantee it has any entropy when produced by an adversarial prover. The coinbase transaction should be part of the witness in our POW syntax (since it is produced by the proving process and is necessary for verification) and has the role of randomizing the proving procedure. Finally, the counter is also included in the witness. BPOW, a simplified version of the scheme described above (that eliminates the transaction semantics for simplicity), is presented in Figure 2.

Remark 2. 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 freshness parameter in the hash of the message.

```
BPOW: Bitcoin's POW implementation
– Prove($r, msg, h$) :
  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($r, msg, h, w$) :
  1.  return $(H(r, G(r, w_1, msg), w_2) \overset{?}{<} 2^\lambda - h)$, where $w = w_1||w_2$.
```

Figure 2: *Bitcoin's POW scheme. H and G are hash functions instantiated with SHA-256.*

Concrete security in the RO model. We will assume that both H and G are idealized hash functions, i.e., our analysis is in the random oracle (RO) model [9].

Theorem 9. *For any $\delta \in (0, 1)$, and for some unpredictability-preserving tampering function class $\mathcal{F}$,* BPOW *is complete, $O(\lambda)$-verifiable and*

– $((1+\delta)(1-\frac{h}{2^\lambda}), \exp(-\beta(h)(1+\log(q_\mathcal{P}))\delta^2 t/6))$*-H-TCMA secure;*

– $(\frac{1}{s} \cdot \sum_{i\in[s]} n \cdot (\sum_{j\in[n]} t_{i,j}) \cdot (1-\frac{h}{2^\lambda})\frac{h}{2^\lambda}^{n(\sum_{j\in[n]} t_{i,j})-1}, \exp(-\gamma(n, \mathbf{T}, h) \cdot \delta^2 s/2))$ *-US-CKMA secure.*

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.

Completeness. The completeness property is trivially satisfied by the scheme.

H-TCMA security. Let $k = \beta(h)t$. 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 $\mathcal{P}$. Hence, we will not have to take $\mathcal{P}$ into account in our analysis. $\mathcal{A}'$ is going to run $\mathcal{A}$ internally, and all calls made by $\mathcal{A}$ to $\mathcal{P}$ the are going to be simulated, i.e., assuming $\mathcal{A}$ queries $\mathcal{P}$ with values (f, r, m), $\mathcal{A}'$ 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 $t > w_2$. $\mathcal{A}'$ is also going to store this query in some efficient structure that allows for search in logarithmic time. Any calls made by the adversary afterwards to the RO that are related to (f, r, m) will be answered accordingly; if $\mathcal{A}_2$ queries the RO with some string $H(f(r), G(f(r), w_1, msg), w_2')$, where $w' = w_2$, 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. Hence, the view of $\mathcal{A}$ will be the same in both executions and he will output k 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 $\langle f, r, m, (w_1, w_2)\rangle$ and a POW on the output $\langle f', r', m', (w_1', w_2')\rangle$ such that $G(f(r), w_1, m_1) = G(f'(r'), w_1', m_1')$ and $m \neq m'$ 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, $\mathcal{A}'$ will provide the same output as $\mathcal{A}$ and with the same probability (minus some negligible term in λ) it will win the experiment. Moreover, the overhead incurred to $\mathcal{A}'$ running time will be only logarithmic on $q_{\mathcal{P}}$ i.e. $\mathcal{A}'$ can simulate the t steps taken by $\mathcal{A}$ after his first call to $\mathcal{R}$ 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 be the event where $\mathcal{A}$ asks t queries the RO after querying $\mathcal{R}$, and receives more than k 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 $\beta(h)$ equal to $(1+\delta)p_h$ it follows that, for any $\delta \in (0, 1)$:

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

Let B be the event where $\mathcal{A}$ can find an $f \in \mathcal{F}$ where for some query $H(y, x, w)$ he made to the random oracle before querying $\mathcal{R}$ for the first time and some $r \in \mathsf{Ans}_{\mathcal{R}}$, it holds that $f(r) = y$. Notice that he does not know the responses of $\mathcal{R}$ when he picks y. Assume that B happens with non-negligible probability. Then we can use $\mathcal{A}$ to break the unpredictability-preserving property of $\mathcal{F}$. $\mathcal{A}$ will randomly output the y part of one of the queries he has made before querying $\mathcal{R}$, and then he will try to find a function f matching this query. The probability of this event is non-negligible which is a contradiction. Thus, $\Pr[B]$ must be negligible in λ.

Let C be the event where the adversary outputs two POWs that correspond to the same query in 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 be the event where the adversary outputs a valid POW that he has not asked the RO. This event occurs with probability p_h which is $\mathsf{negl}(\lambda)$.

If A, B, C, D do not occur, it is implied that $\mathcal{A}'$ will lose in the experiment. Thus, by an application of the union bound we get:

$$\begin{aligned}\Pr[\mathsf{Exp}^{\text{H-TCMA}}_{\mathcal{A},\mathcal{F}}(1^\lambda, h, k) = 1] &\leq 1 - \Pr[\mathsf{Exp}^{\text{H-TCMA}}_{\mathcal{A}',\mathcal{F}}(1^\lambda, h, k) = 0] \\ &\leq 1 - \Pr[\neg(A \vee B \vee C \vee D)] \\ &\leq 1 - (1 - \Pr[A \vee B \vee C \vee D]) \\ &\leq \Pr[A \vee B \vee C \vee D] \leq e^{-\frac{\beta(h)t(1+\log(q_{\mathcal{P}}))\delta^2}{6}} + \mathsf{negl}(\lambda).\end{aligned}$$

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.

US-CKMA. We can show that collisions happen with negligible probability, and thus all queries to the RO made by the prover will be different. Let the binary random variable Y_i equal 1 if b_i is 1, i.e. only one of the n processes has the time to compute a proof-of-work, and $Y = \sum_{i=1}^{s} Y_i$. Then it holds that

$$\mathbb{E}[Y] = \sum_{i\in[s]} \mathbb{E}[Y_i] = \sum_{i\in[s]} n \cdot (\sum_{j\in[n]} t_{i,j}) \cdot p_h(1-p_h)^{n(\sum_{j\in[n]} t_{i,j})-1} = \gamma(n, \mathbf{T}, h) \cdot s$$

By the Chernoff bound:

$$\Pr\left[\sum_{i=1}^{s} b_i < \gamma(n, \mathbf{T}, h)\cdot s \text{ where } \forall i \in [s] : b_i = (\exists! j \in [n] : \mathsf{Steps}_{\mathsf{Prove}}(r_{i,j}, msg_{i,j}, h) < t_{i,j})\right]$$
$$\le \Pr[Y < (1-\delta)\mathbb{E}[Y]] \le \Pr[Y < (1-\delta)\gamma s] \le e^{-\gamma s \delta^2/2}.$$

□

5 Proving the Bitcoin Backbone from POW in the Standard Model

As mentioned in Section 1, existing analyses of Bitcoin have relied on the random oracle model to prove directly the properties of the blockchain protocol. 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 [19] 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.

5.1 Bitcoin backbone model and definitions

In [19] 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.

Next, we consider the complications in the modeling due to the analysis of Bitcoin in the concrete security setting. Both in [19] and [28] a modified version of the standard simulation-based paradigm [14] 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 multiple 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 [14], 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. More specifically, if Q is some property we want to prove, our results will be stated in the following form: With overwhelming probability either Q holds or the execution is not good.

Definition 10. Let $(t_{\mathcal{A}}, t_{\mathcal{H}}, \theta)$-good be a predicate defined on executions in the hybrid setting described above. Then E *is* $(t_{\mathcal{A}}, t_{\mathcal{H}}, \theta)$*-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 $t_{\mathcal{A}}$;[3]
- each honest party takes at least $t_{\mathcal{H}}$ steps per round; and
- the adversary sends at most θ messages per round.

In [19, 26] the desired security properties of a blockchain protocol were formulated, i.e., common-prefix, chain-quality and chain-growth. These properties capture respectively, the hardness of inducing a fork in the honest parties' chains, the existence of honestly created blocks in the chains of honest players and the growth of the honest players' chains at a certain rate. The properties are sufficient to ensure the Persistence and Liveness of the bitcoin protocol at least assuming a static number of parties. For more details about the properties as well as for the modified bitcoin backbone protocol we defer the reader to Appendix B.

[3]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.

5.2 Security proof

Next, we follow the framework set in [19] to prove that the modified Bitcoin backbone protocol is secure assuming that the underlying POW scheme is secure. First, we introduce some additional notation.

For each round j, we define the Boolean random variables X_j and Y_j as follows. Let $X_j = 1$ iff j was a *successful round*, i.e., at least one honest party computed a POW at round j, and let $Y_j = 1$ iff 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 $Y(S)$ similarly.

For the rest of this section we will assume that $t_{\mathcal{A}}, t_{\mathcal{H}}, \theta, h$ are fixed constants and that the underlying POW scheme used in the Bitcoin backbone protocol with respect to some unpredictability-preserving tampering function class is:

- Complete;
- (β, ϵ_1)-H-CMA secure;
- (γ, ϵ_2)-uniquely successful; and
- (t_{ver})-verifiable.

Furthermore, we will assume that the steps needed to simulate running the code of an honest party in one round, besides the Prove and Verify calls, is upper bounded by t_{bb}. By carefully analyzing the backbone protocol one can extract[4] an upper bound on this value. 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 in account calls made to the Prove routine by the honest parties, and to the minimum number of steps spent by honest parties running the Prove routine at each round respectively.

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

Finally, 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 each element of the array is lower bounded by $t_{\mathcal{H}} - t_{\mathsf{bb}} - \theta t_{\mathsf{ver}}$ and upper bounded by $t_{\mathcal{H}}$. We define γ to be the minimum rate of uniquely successful rounds in any set of rounds S.

$$\gamma_{min} = \min\{\gamma(n, \mathbf{T}_{S \times n}, h) |\ S \subseteq [poly(\lambda)] \wedge \forall i, j \in S \times [n] :\ t_{\mathcal{H}} - t_{\mathsf{bb}} - \theta t_{\mathsf{ver}} \leq t_{i,j} \leq t_{\mathcal{H}}\}$$

In [19] it was assumed that the input contribution function $I(\cdot)$ has some entropy. Here, we need a similar but stronger assumption, namely, that the hash function family that "binds" blocks together in the blockchain is collision and everywhere preimage resistant (Definition 1) with respect to the random coins of the input contribution function. This assumption is needed in particular so that the adversary will not be able to make different chains "collide" or predict the hash of future blocks.

Unpredictability Assumption. The following hash function family is collision and everywhere preimage (ePre) resistant:

$$\mathcal{H}_{\mathsf{chain}} = \{h(K, M) \overset{\text{def}}{=} H^*(H(r, G(r, w_1, I(m; K)), w_2), M) : \{0,1\}^* \rightarrow \{0,1\}^{\lambda}\}_{m \in \mathsf{Dom}_I; r, w_1, w_2 \in \{0,1\}^{\lambda}},$$

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

where $H^*(\cdot,\cdot)$ is the basic cascade construction [7]. Notice that its first argument is the hash of some block. In Appendix A we prove that we can instantiate a hash function with these properties in the random oracle model. Moreover, as mentioned in Section 2 (a proof is presented in the Appendix A) ePre resistance of $\mathcal{H}_{\mathsf{chain}}$ implies that the following function family is unpredictability preserving:

$$\mathcal{F}_{\mathsf{chain}} = \{h(K, M)\}_{M \in \{\{0,1\}^\lambda\}^*}.$$

Next, using the unpredictability assumption, we prove a number of properties related to the way blocks are connected.

Definition 11. 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. A *prediction* occurs when a block extends one honest block which was computed at a later round.

Lemma 12. *Assuming the Unpredictability Assumption, no insertions, no copies and no predictions occur with probability* $1 - \mathsf{negl}(\lambda)$.

Proof. Insertions and copies imply that one block extents two distinct blocks, i.e. there exist blocks B_1, B_2, B_3 such that $s_3 = H(B_1) = H(B_2)$. Since all blocks extend the genesis block this in turn implies that there exists M_1, M_2 that break the collision resistance property of $\mathcal{H}_{\mathsf{chain}}$.

Next, for the sake of contradiction assume that prediction happens with non-negligible probability. We can use this adversary that breaks prediction to break the ePre property. The adversary would just output at random the seed of one of the blocks he has mined (since they are polynomial in number at most), say B_2, as y in the ePre game. By our assumption with non negligible probability some honest block B_1 will be mined afterwards such that $y = H(B_1)$, hence we can use the parameters of this block to generate an output to win the ePre game. □

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.

Lemma 13. *For any set of consecutive rounds S it holds that the probability that an execution is $(t_\mathcal{A}, t_\mathcal{H}, \theta)$-good and there exists some honest block B mined in S such that $Z_B(S) > \beta t'_\mathcal{A}|S|$, is at most $\epsilon_1(\beta, t'_\mathcal{A} \cdot |S|, n \cdot |S|)$.*

Proof. Let $S = \{r' | r \leq r' < r + s\}$ 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 that descend some honest block B until round $r + s$. For the sake of contradiction, assume that the lemma does not hold, and thus the probability that the execution is $(t_\mathcal{A}, t_\mathcal{H}, \theta)$-*good* and E holds is greater than $\epsilon_1(\beta, t'_\mathcal{A} s, ns)$. 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 two oracles 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.

We are going to describe the two stages of $\mathcal{A}'$ separately, i.e. before and after querying $\mathcal{R}$ for the first time. First, $\mathcal{A}'$ perfectly simulates honest parties up to round $r - 1$ and at the same time runs $\mathcal{A}$ and $\mathcal{Z}$ in a black-box way. 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. If r is the first round, $\mathcal{A}'$ will do nothing.

For the second stage, $\mathcal{A}'$ is again going to simulate honest parties behavior, from round r until round $r+s$, but in a different way. Instead of running the Prove algorithm for each 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, honest parties have 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 $\mathcal{A}'$ has to run the verification procedure only once per message.

Moreover, for the reduction to work we also want all honest blocks created after round r to be related to the outputs of the randomness oracle $\mathcal{R}$ through some function in $\mathcal{F}_{\mathsf{chain}}$. In our scenario, $\mathcal{R}$ will generate uniformly random strings of the same length as the randomness needed by the input contribution function I. At every round starting from r and for each honest party, $\mathcal{A}'$ will query $\mathcal{R}$ and use the response to run I. If an honest block is mined using this response, then any block descending it will be related to the randomness used through some function in $\mathcal{F}_{\mathsf{chain}}$. More precisely, if $\langle s, I(m; r_i), (w_1, w_2)\rangle$ is the honest block mined using the response r_i of oracle $\mathcal{R}$, then any block descending it is of the form $\langle H^*(H(s, G(s, w_1, I(m; r_i)), w_2), M), x', w'\rangle$ for some M, x', w'. Hence, it is related to r_i through some function in $\mathcal{F}_{\mathsf{chain}}$ 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 $\epsilon_1(\beta, t'_{\mathcal{A}}s, ns)$, i.e. $\mathcal{A}$ will compute at least $\beta t'_{\mathcal{A}}s$ blocks starting from round r and up to round $r+s$ that descend some honest block B mined during these rounds. Note, that these blocks are also valid POWs, that are related through $\mathcal{F}_{\mathsf{chain}}$ to the responses of $\mathcal{R}$. Hence, $\mathcal{A}'$ will win the H-CMA game with probability greater than $\epsilon_1(\beta, t'_{\mathcal{A}}s, ns)$, 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 3.

Note that we can do exactly the same reduction without using the oracle to simulate the proving procedure of the honest parties. The total running time of the second stage of $\mathcal{A}'$ is at least $s \cdot (t'_{\mathcal{A}} + t'_{\mathcal{H}})$-bounded and hence the probability he can win is $\epsilon_1(\beta, s \cdot (t'_{\mathcal{A}} + t'_{\mathcal{H}}), 0)$

□

By considering honest parties malicious and *not* using the prove oracle to simulate them, we can derive the following corollary.

Corollary 14. *For any set of consecutive rounds S it holds that the probability that an execution is $(t_{\mathcal{A}}, t_{\mathcal{H}}, \theta)$-good and there exists some honest block B mined in S such that $Z_B(S) + X(S) < \beta(t'_{\mathcal{A}} + t'_{\mathcal{H}}) \cdot |S|$ is less than $\epsilon_1(\beta, |S| \cdot (t'_{\mathcal{A}} + t'_{\mathcal{H}}), 0)$.*

We are now ready to define the set of *typical* executions for this setting. This strategy was also followed in [19]. However, here we will need to adapt the definition due to the difficulties of performing a black-box reduction to a POW scheme.

Definition 15. (Typical execution) An execution is *η-typical* iff the execution is *$(t_{\mathcal{A}}, t_{\mathcal{H}}, \theta)$-good* and for any set S of consecutive rounds with $|S| \geq \eta\lambda$, the following hold:

1. $X(S) \geq Y(S) \geq \gamma(n, \mathbf{T}_{S\times n}, h) \cdot |S|$ and $X(S) \leq \beta t'_{\mathcal{H}} \cdot |S|$;
2. for any block B mined by an honest party during S, $Z_B(S) \leq \beta t'_{\mathcal{A}} \cdot |S|$; and
3. no insertions, no copies and no predictions occurred.

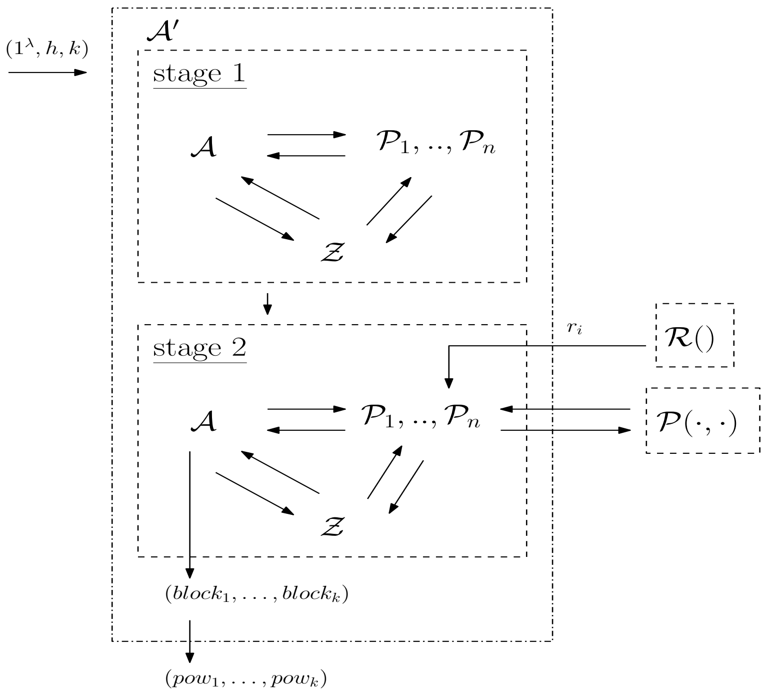

Figure 3: *The reduction from the bitcoin backbone to the H-TCMA game of Lemma 13.*

Theorem 16. *An execution is typical with probability* $1 - \sum_{|S| \geq \eta\lambda}(\epsilon_1(\beta, t'_{\mathcal{A}}|S|, n|S|) + \epsilon_2(\gamma(n, \mathbf{T}_{S\times n}, h), |S|) + \epsilon_1(\beta, |S| \cdot (t'_A + t'_{\mathcal{H}}), 0))$.

Proof. First, we lower bound the number of u.s. rounds in any set of s rounds.

Claim 1. *For any set of rounds S it holds that the probability that the execution is $(t_{\mathcal{A}}, t_{\mathcal{H}}, \theta)$-good and less than $\gamma(n, \mathbf{T}_{S\times n}, h)\cdot|S|$ uniquely successful rounds occur in S is at most $\epsilon_2(\gamma(n, \mathbf{T}_{S\times n}, h), |S|)$.*

Proof of Claim. Since the execution is $(t_{\mathcal{A}}, t_{\mathcal{H}}, \theta)$-good, each honest party will run the Prove algorithm for the number of steps specified in $\mathbf{T}_{S\times n}$. Hence, no matter which messages the adversary chooses for honest parties to mine on, directly from the uniquely successful property of the POW scheme it holds that the number of u.s. rounds will be less than $\gamma(n, \mathbf{T}_{S\times n}, h) \cdot |S|$ with probability at most $\epsilon_2(\gamma(n, \mathbf{T}_{S\times n}, h), |S|)$. $\dashv$

By applying the union bound on Lemmas 13, Claim 1 and Corollary 14 it holds that the probability that there exists any set of consecutive rounds S, where $|S| > \eta\lambda$, and either $Y(S) < \gamma(n, \mathbf{T}_{S\times n}, h)|S|$ or for some honest block mined during S, $Z_B(S) > \beta t'_{\mathcal{A}}|S|$ or $Z_B(S) + X(S) > \beta(t'_A + t'_{\mathcal{H}}) \cdot |S|$, is at most $\sum_{|S| \geq \eta\lambda} \epsilon_1(\beta, t'_{\mathcal{A}}|S|, n|S|) + \epsilon_2(\gamma(n, \mathbf{T}_{S\times n}, h), |S|) + \epsilon_1(\beta, |S| \cdot (t'_A + t'_{\mathcal{H}}), 0))$. 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 12. □

Honest majority assumption. It holds that $\gamma_{min} \geq (1+\delta)\beta \cdot t'_{\mathcal{A}}$ for some $\delta \in (0,1)$.

Lemma 17. *For any set S of at least $\eta\lambda$ rounds in a η-typical execution and for any block B mined by an honest party during S, it holds that*

- $Z_B(S) \leq (1 - \frac{\delta}{2})X(S)$

- $Z_B(S) < Y(S)$

Proof. For the first item:

$$Z_B(S) \leq \beta t'_{\mathcal{A}} \cdot |S| \leq \frac{1}{1+\delta}\gamma(n, \mathbf{T}_{S\times n}, h)|S| < (1-\frac{\delta}{2})X(S)$$

The first and the last inequality hold by the typicality of the execution. The middle one holds from the Honest majority assumption. For the second item we have that

$$Z_B(S) \leq \beta t'_{\mathcal{A}} \cdot |S| < \gamma(n, \mathbf{T}_{S\times n}, h) \cdot |S| \leq Y(S)$$

□

We can now use the machinery built in [19] to prove the common-prefix, chain-quality and chain-growth properties, with only minor changes. We refer the reader to Appendix B.4 for more details. Using these properties we can prove that bitcoin implements a robust transaction ledger.

Lemma 18. *(Persistence). Under the Honest Majority Assumption, if an execution is typical it holds that Π_{PL} with $k = \eta\lambda(\gamma(n, \gamma_{min} + \beta t'_{\mathcal{H}})$ satisfies Persistence.*

Lemma 19. *(Liveness). Under the Honest Majority Assumption, if an execution is typical it holds that Π_{PL} with $u = 4\eta\lambda$ rounds and $k = \eta\lambda(\gamma_{min} + \beta t'_{\mathcal{H}})$ satisfies Liveness.*

Finally, we derive concrete bounds for the POW implementation presented at section 4. More specifically, we showed that we can construct[5] a POW scheme in the random oracle model, with respect to an unpredictability-preserving tampering function class, which is:
- Complete;
- $O(\lambda)$-verifiable;
- $((1+\epsilon)(1-\frac{h}{2^\lambda}), \exp(-\beta \cdot (1+\log(q_{\mathcal{P}}))\epsilon^2 t/6))$-H-TCMA secure;
- $(\frac{1}{s} \cdot \sum_{i\in[s]} n \cdot (\sum_{j\in[n]} t_{i,j}) \cdot (1-\frac{h}{2^\lambda})\frac{h}{2^\lambda}^{n(\sum_{j\in[n]} t_{i,j})-1}, \exp(-\gamma(n, \vec{t}, h) \cdot \delta^2 s/2))$-US-CKMA secure

For this implementation, and assuming the honest majority assumption holds, we can show that an execution is typical with probability:

$$1 - \sum_{|S|\geq\eta\lambda} (e^{-\frac{\beta t'_{\mathcal{A}}|S|(1+\log(n|S|))\delta^2}{6}} + e^{-\frac{\gamma(n,\mathbf{T}_{S\times n},h)|S|\delta^2}{2}} + e^{-\frac{\beta\cdot(t'_{\mathcal{A}}+t'_{\mathcal{H}})|S|\delta^2}{6}}) \leq 1 - e^{-\Omega(\lambda)}.$$

Therefore, by the analysis we did in Section B.4, it follows that the bitcoin backbone under this implementation of POW will satisfy persistence and liveness with overwhelming probability in λ.

6 H-TCMA Security vs. POW Sampling

POW "sampleability," i.e., the existence of a program that can *efficiently* generate POWs has been a key property in related work [4, 12]. However, in contrast with our adaptive H-TCMA security notion, this notion provides an average-case security guarantee. In this section we investigate the relations between POW sampleability, H-TCMA security, and average case security, which, following [4], we call *amortization resistance.*

[5]In Section 4, in order to make our presentation easier to follow, we choose to allow the `BPOW Prove` routine to run until it finds a solution. However, in the 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.

Definition 20. Let S be a program that runs in time t and its output has min-entropy[6] at least λ and $\mathsf{POW} = (\mathsf{Prove}, \mathsf{Verify})$ be a POW scheme. We say that S is a t-sampler for POW if

$$\Pr[(m,w) \leftarrow S(1^\lambda, h, r) : \mathsf{Verify}(r,m,h,w) = 1] = 1$$

where r, m, w, h are in $R, M, W, \mathbb{N}$ respectively.

The adaptive nature of H-TCMA security is in contrast with POW samplers, in the sense that the rate at which the adversary can produce POWs is at least equal to the running time of the sampler, as the next lemma shows.

Lemma 21. *Let S be a t-sampler for some POW scheme. Then, the scheme cannot be $(1/t, 1 - \mathsf{negl}(\lambda))$-H-TCMA secure.*

Proof. Since the sampler's output has min-entropy λ, in time t' the adversary can run the sampler t'/t times with some input $(1^\lambda, h, r)$ it got from oracle $\mathcal{R}$, and with overwhelming probability obtain t'/t different and correct POWs. Hence, he can break the H-TCMA property with parameter $1/t$ and probability at least $1 - \mathsf{negl}(\lambda)$. □

On the other hand, it could be the case that for some application we need a POW sampler, e.g. a server wants to be protected against DDOS attacks and periodically publishes one fresh POW key, while on the same time he wants trusted clients to be able to freely communicate, and thus using a sampler he distributes "cheap" POWs. Similarly in [5], sampling is used to produce an "indistinguishable proof of work or knowledge": In the proof of knowledge mode they need the sampler in order to achieve indistinguishability from the proof of work mode. Following [4, 12, 18], we can define a weaker security notion for POWs called *amortization resistance*. Here we give a simplified definition compared to Definition 7 which nonetheless is still sufficient to make our point that there exists a meaningful security definition that is compatible with samplers. Note that in the amortization-resistance experiment the adversary does not have the ability to adaptively choose the messages he wants to encode. As such, it seems that this property is more suitable for authenticated ("permissioned") settings, where there are other means of establishing the ownership of POWs.

```
                    $\mathsf{Exp}^{\mathrm{AR}}_{\mathcal{A}}(1^\lambda, h, k)$
1.   For $\mathsf{i} = 1$ to $k$ : $(r_i, msg_i) \xleftarrow{\$} (\mathcal{U}_\lambda, \mathcal{U}_\lambda)$;
2.   $(w_1, \ldots, w_k) \leftarrow \mathcal{A}(1^\lambda, h, \{r_i, msg_i\}_{i \in [k]})$;
3.   output $\bigwedge_{i=1}^{k} \mathsf{Verify}(r_i, msg_i, w_i)$.
```

Figure 4: *The amortization resistance experiment for POWs.*

Definition 22. We say that a POW scheme has (β, ϵ)-*amortization resistance* if for every polynomial $h(\cdot), t(\cdot)$, and for every t-bounded adversary $\mathcal{A}$, $\mathcal{A}$'s probability of winning in $\mathsf{Exp}^{\mathrm{AR}}_{\mathcal{A}}(1^\lambda, h, \beta(h) \cdot t)$ is less than $\epsilon(\beta(h), t)$.

Next we prove that H-TCMA security implies amortization resistance.

Lemma 23. *Let* $\mathsf{POW} = (\mathsf{Prove}, \mathsf{Verify})$ *be a (β, ϵ)-H-TCMA secure POW scheme, for some functions β, ϵ. Then* POW *is also (β, ϵ)-amortization resistant.*

[6]We want our sampler to have different output everytime it is invoked with overwhelming probability.

Proof. For the sake of contradiction, assume POW is not amortization resistant. We will construct an adversary $\mathcal{A}'$ which breaks the H-TCMA security of the scheme. Let $\mathcal{A}$ be the adversary that breaks AR. $\mathcal{A}'$ will first simulate the message sampling process of $\mathsf{Exp}^{\mathrm{AR}}$ and then invoke $\mathcal{A}$ using these freshness and message pairs as input. The probability that $\mathcal{A}$ will output enough POWs and thus break the H-TCMA property is the same as that of breaking AR, while the overhead of simulating the sampling process is minimal compared to the total running time of $\mathcal{A}$. This is a contradiction and the lemma follows. $\square$

In Figure 5 we present an amortization-resistant POW scheme based on a scheme from [4]. Interestingly, we can construct a sampler for this scheme[7]. Hence, by Lemma 21, this scheme cannot be H-TCMA secure with a better parameter than the running time of the sampler. Taking into account Lemma 23, it follows that H-TCMA is a strictly stronger security notion compared to amortization resistance. Refer to Figure 6 for the relations among all the POW security notions discussed in this section.

An amortization-resistant POW scheme

- $\mathsf{Prove}(r, msg, h)$: Find w s.t. $\mathsf{LSB}_h(H(r,w)) = \mathsf{LSB}_h(msg)$.
- $\mathsf{Verify}(r, msg, h, w)$: Return $(\mathsf{LSB}_h(H(r,w)) \overset{?}{=} \mathsf{LSB}_h(msg))$.
- $S(1^\lambda, h, r)$: $w \overset{\$}{\leftarrow} \mathcal{U}_\lambda$; return $(H(r,w), w)$.

Figure 5: *An adaptation of the amortization-resistant scheme from [4] to our setting.* $\mathsf{LSB}_h(x)$ *denotes the* h *least significant bits of* x.

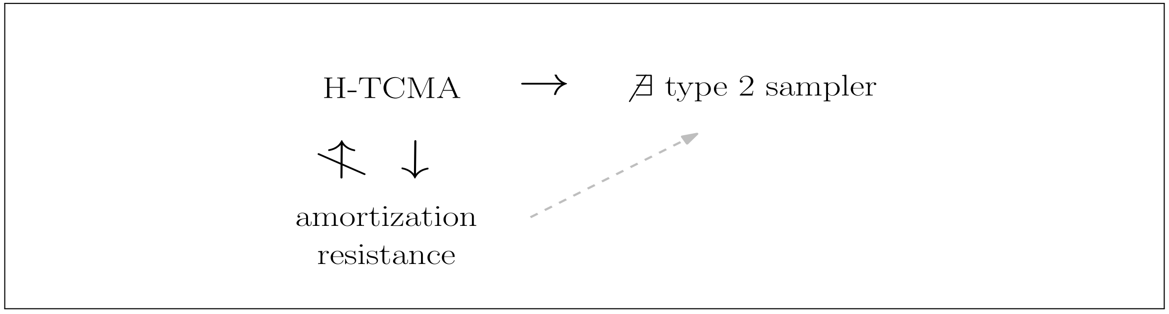
$$\begin{tikzcd} \text{H-TCMA} \arrow[r] \arrow[d, shift left] & \nexists \text{ type 2 sampler} \\ \text{amortization resistance} \arrow[u, shift left, "/" marking] \arrow[ur, dashed] & \end{tikzcd}$$

Figure 6: *The relations among POW sampleability, H-TCMA and amortization resistance. The dotted line signifies plausibility i.e. there exists an amortization resistant scheme that has a sampler.*

Remark 3. Another notion 'similar' to sampleability is the existence of a trapdoor [17]. We can formalize this notion as follows: a POW scheme has a trapdoor iff there exists a t-bounded RAM S such that for any PPT RAM $\mathcal{A}$

$$\Pr[(r,\tau) \leftarrow S(1^\lambda, h); m \leftarrow \mathcal{A}(1^\lambda, r, h); w \leftarrow S(1^\lambda, h, \tau, r, m) : \mathsf{Verify}(r, m, h, w) = 1] \geq 1 - negl(\lambda)$$

The BPOW scheme presented in Section 4 does not seem to have this property. On the other hand, one of the schemes presented in [17] has this property. We leave the question of whether a scheme can be H-TCMA secure and also have a trapdoor as future work.

[7]Note that if in the scheme presented in Figure 5 we swap r and msg and vice-versa, it is not obvious how to construct a sampler. However, the construction presented suffices for our objective, which is establishing the separation between H-TCMA security and amortization resistance.

References

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

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

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

[4] 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`.

[5] F. Baldimtsi, A. Kiayias, T. Zacharias, and B. Zhang. Indistinguishable proofs of work or knowledge. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 902–933, 2016.

[6] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Proofs of useful work, 2017.

[7] M. Bellare, D. J. Bernstein, and S. Tessaro. Hash-function based prfs: Amac and its multi-user security. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 566–595. Springer, 2016.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-lock puzzles from randomized encodings. Cryptology ePrint Archive, Report 2015/514, 2015. `http://eprint.iacr.org/2015/514`.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] J. Garay, D. Johnson, A. Kiayias, and M. Yung. Resource-based corruptions and the combinatorics of hidden diversity. Cryptology ePrint Archive, Report 2012/556, 2012. `http://eprint.iacr.org/2012/556`.

[19] 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.

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

[21] J. A. Garay, A. Kiayias, N. Leonardos, and G. Panagiotakos. Bootstrapping the blockchain — directly. Cryptology ePrint Archive, Report 2016/991, 2016. `http://eprint.iacr.org/2016/991`.

[22] 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.

[23] 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.

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

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

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

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

[28] R. Pass, L. Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016. `http://eprint.iacr.org/2016/454`.

[29] 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.

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

[31] 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.

A Hash Functions and Unpredictability Preservation

In [7], Bellare *et al.* describe the notion of the basic cascade as follows:

$$\begin{aligned}&\mathcal{H}^*(K, X):\\ &Y \leftarrow K;\ \text{for } i = 1 \text{ to } n \text{ do } Y \leftarrow H(Y, X[i]);\ \text{return } Y\end{aligned}$$

We are going to prove that if we instantiate H and G in the $\mathcal{RO}$ model and assume that I has some entropy (similar to the input entropy assumption in [19]), then the following function family is ePre and collision resistant.

$$\mathcal{H}_{chain} = \{h(K, M) \overset{\text{def}}{=} H^*(H(r, G(r, w_1, I(m; K)), w_2), M) : \{0,1\}^* \to \{0,1\}^\lambda\}_{m \in \mathsf{Dom}_I; r, w_1, w_2 \in \{0,1\}^\lambda}$$

Lemma 24. *$\mathcal{H}_{chain}$ is ePre and collision resistant.*

Proof. If the adversary is able to find a collision in $\mathcal{H}_{\mathsf{chain}}$ if follows that he can find a collision in H or G, which happens with negligible probability in λ. Regarding the ePre property, note that the input entropy assumption states that the probability that two invocations of I return the same value for arbitrary inputs is negligible. It follows that the adversary cannot predict its output and thus the probability that the adversary can win the ePre game is negligible in λ. □

Having a function family that is ePre resistant implies that we can construct a function family that is unpredictability preserving, since the two notions are closely related.

Lemma 25. *If $\mathcal{H} = \{h(K, M)\}$ is an ePre resistant hash function family then $\mathcal{F} = \{h(K, M)\}_{M \in \mathcal{M}}$ is an unpredictability preserving function family.*

Proof. Assume $\mathcal{F}$ was not unpredictability preserving. Then there exists some adversary $\mathcal{A}$ that breaks the property with non-negligible probability. We can use him exactly as he is to break the ePre property with non-negligible probability. The fuction that $\mathcal{A}$ outputs in the second stage of the game corresponds to a message that breaks the ePre property. Hence, the lemma follows. □

B The Bitcoin Backbone Protocol

B.1 Security properties of the blockchain

A number of desired basic properties for the blockchain were introduced in [19, 26]. 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 [26, 28], 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 B.2).

Definition 26 ((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 $r_1 \le r_2$, 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 27 (Chain Quality)**.** The *chain quality property* Q_{cq} with parameters $\mu \in \mathbb{R}$ and $k, k_0 \in \mathbb{N}$ states that for any honest party P with chain $\mathcal{C}$ in $\textsc{view}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$, it holds that for any k consecutive blocks of $\mathcal{C}$, excluding the first k_0 blocks, the ratio of adversarial blocks is at most μ.

Further, in the derivations in [19] 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 [26] 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 B.2).

Definition 28 (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 $\textsc{view}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$, it holds that $|\mathcal{C}_2| - |\mathcal{C}_1| \ge \tau \cdot s$.

B.2 Robust public transaction ledgers

In [19] the robust public transaction ledger primitive was presented. It tries to capture the notion of a book where transactions are recorded, and it is used to implement Byzantine Agreement in the honest majority setting.

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 $\mathrm{tx} \in \mathcal{T}$. Each transaction tx may be associated with one or more *accounts*, denoted $a_1, a_2, \ldots$ Ledgers correspond to chains in the backbone protocols. An oracle Txgen is allowed in the protocol execution that generates valid transactions (this represents transactions that are issued by honest parties). For more details we refer to [19].

Definition 29. A protocol Π implements a *robust public transaction ledger* in the q-bounded synchronous setting 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.

B.3 The modified Bitcoin backbone protocol

In this section we describe the Bitcoin backbone protocol in the black-box POW setting. First, we introduce some notation 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. Each block contains a seed, data, and a witness denoted by s, x, w, respectively. We say a chain is *valid* iff 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 $H(s_i, G(s_i, w_{i,1}, x_i), w_{i,2}) = s_{i+1}$, where $w_{i,1}$ represents the first λ bits of w_i. We call this value the *hash of block* B_i and denote it by $H(B)$.

The modifications with respect to the original protocol in [19] are: In Algorithm 1 (proof of work function), where black-box calls to the underlying POW scheme are made, and in Algorithm 2, where the Verify predicate is replaced with a call to the Verify algorithm of the POW scheme. Also note in Algorithm 1 parties run the Prove algorithm for a specific number of steps, so that in total they have run $t_{\mathcal{H}}$ steps. This does not affect the application of the properties described in Section 3.

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($x, \mathcal{C}$)
    if $\mathcal{C} = \varepsilon$ then                         ▷ Determine proof of work instance
        $s \leftarrow 0$
    else
        $\langle s', x', w' \rangle \leftarrow \mathrm{head}(\mathcal{C})$
        $s \leftarrow H(w', G(s', x'))$
    end if
    $B \leftarrow \varepsilon$
    $w \leftarrow \mathsf{Prove}(s, x, h)$                  ▷ Run the prover of the POW scheme.
    if $ctr \neq \perp$ 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 $\mathcal{C}$.

```
function validate($\mathcal{C}$)
    $b \leftarrow V(\mathbf{x}_{\mathcal{C}}) \wedge (C \neq \varepsilon)$
    if $b =$ True then               ▷ The chain is non-empty and meaningful w.r.t. $V(\cdot)$
        $\langle s, x, w \rangle \leftarrow \mathrm{head}(\mathcal{C})$
        $s' \leftarrow H(w, G(s, x))$
        repeat
            $\langle s, x, w \rangle \leftarrow \mathrm{head}(\mathcal{C})$
            if $\mathsf{Verify}(s, x, h, w) \wedge (H(w, G(s, x)) = s')$ then   ▷ Verify that this is a valid POW
                $s' \leftarrow s$                                    ▷ Retain hash value
                $\mathcal{C} \leftarrow \mathcal{C}^{\lceil 1}$                          ▷ Remove the head from $\mathcal{C}$
            else
                $b \leftarrow$ False
            end if
        until $(\mathcal{C} = \varepsilon) \vee (b =$ False$)$
    end if
    return $(b)$
end function
```

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

```
1: $\mathcal{C} \leftarrow \varepsilon$
2: $st \leftarrow \varepsilon$
3: $round \leftarrow 0$
4: while TRUE do
5:     $\tilde{\mathcal{C}} \leftarrow \mathsf{maxvalid}(\mathcal{C}, \text{any chain } \mathcal{C}' \text{ found in RECEIVE()})$
6:     $\langle st, x \rangle \leftarrow I(st, \tilde{\mathcal{C}}, round, \text{INPUT}(), \text{RECEIVE}())$        ▷ 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:        BROADCAST($\mathcal{C}$)
11:    end if
12:    $round \leftarrow round + 1$
13:    if INPUT() contains READ then
14:        write $R(\mathbf{x}_{\mathcal{C}})$ to OUTPUT()
15:    end if
16: end while
```

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

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

B.4 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 [19]. Here, we describe the minor changes one has to do after proving the typical execution theorem with respect to the analysis of [19], in order to prove the security of the protocol in our model. We only give brief proof sketches of lemmas or theorems from [19] that are exactly the same for our own setting.

Lemma 30. *(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 31. *(Chain-Growth). In a typical execution the chain-growth property holds with parameters $\tau = \gamma_{min}$ and $s \geq \eta\lambda$.*

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 unique POW computation
δ :	advantage from the honest majority assumption
k :	number of blocks for the common-prefix property
ℓ :	number of blocks for the chain-quality property
η :	parameter determining block to round translation

Table 1: The parameters in our analysis.

Proof. Let S be any set of at least s consecutive rounds. Then, since the execution is η typical $X(S) \geq \gamma(n, \mathbf{T}_{S\times n}, h) \cdot |S| \geq \gamma_{min} \cdot |S| \geq \tau \cdot |S|$. By Lemma 30, 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. □

Lemma 32. *Let B be some honest block mined and assume an execution that is typical. Any $k \geq \eta\lambda(\gamma_{min} + \beta t'_{\mathcal{H}})$ consecutive blocks descending B in a chain in this execution have been computed in at least $\eta\lambda$ rounds, starting from the round that B was computed.*

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

Lemma 33. *(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 $k \geq \eta\lambda(\gamma_{min} + \beta t'_{\mathcal{H}})$.*

Proof. In Lemma 13, 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 [19]. 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 $\mathcal{C}_2$. □

Theorem 34. *(Common-prefix). In a typical execution the common-prefix property holds with parameter $k \geq \eta\lambda \cdot (\gamma_{min} + \beta t'_{\mathcal{H}})$.*

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. □

Theorem 35. *(Chain-Quality). In a typical execution the chain-quality property holds with parameter $\mu < 1 - \delta/2$ and $\ell \geq \eta\lambda(\gamma_{min} + \beta t'_{\mathcal{H}})$.*

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 15. 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. □

Finally, the Persistence and Liveness properties follow from the three basic properties, albeit with different parameters than in [19].

Lemma 36. *(Persistence). Under the BB-Honest Majority Assumption, if an execution is typical it holds that* Π_{PL} *with* $k = \eta\lambda(\gamma_{min} + \beta t'_{\mathcal{H}})$ *satisfies Persistence.*

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. □

Lemma 37. *(Liveness). Under the BB-Honest Majority Assumption, if an execution is typical it holds that* Π_{PL} *with* $u = 4\eta\lambda$ *rounds and* $k = \eta\lambda(\gamma_{min} + \beta t'_{\mathcal{H}})$ *satisfies Liveness.*

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. □