Consensus from Signatures of Work

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

Aggelos Kiayias*
University of Edinburgh
& IOHK
akiayias@inf.ed.ac.uk

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

March 19, 2019

Abstract

Digital signatures are a fundamental building block in the design of consensus protocols that allow correctness with optimal resilience. With the advent of blockchain protocols like Bitcoin, consensus has been considered in the "permissionless" setting where no authentication or even point-to-point communication is available. Despite some preliminary positive results, there has been no attempt to formalize a building block that is sufficient for designing consensus protocols in this setting.

In this work we fill this void by putting forth a formalization of such a primitive, which we call *signatures of work* (SoW). Distinctive features of our new notion are a lower bound on the number of steps required to produce a signature; fast verification; *moderate unforgeability*—producing a sequence of SoWs, for chosen messages, does not provide an advantage to an adversary in terms of running time; and signing time independence—most relevant in concurrent multi-party applications, as we show. Armed with SoW, we present a new permissionless consensus protocol. The protocol is built on top of a SoW-based blockchain which may be of independent interest and standard properties of the underlying hash function, and is secure assuming an honest majority of computational power.

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

Contents

1 Introduction

The consensus problem—reaching agreement distributedly in the presence of faults—has been extensively studied in the literature starting with the seminal work of Shostak, Pease and Lamport [45, 39]. The problem formulation has a number of servers starting with an individual input and agree at the end to a joint output that has to match the input in the case where all non-faulty servers happened to have the same input value. One of the critical measures of effectiveness for consensus protocols is maximizing their resilience to Byzantine faults, typically denoted by t. It is known that $t < n/2$ is necessary to achieve consensus, where n is the total number of parties, while protocols have been designed that reach that level of resilience assuming synchrony and a way to authenticate messages using digital signatures [22][1] (or "pseudosignatures" [46]). This result is known to be tight since lack of synchrony would imply $t < n/3$ [24] (as well as randomization [27]), while lack of a message authentication mechanism has a similar effect [17].

Recently, with the advent of blockchain protocols like Bitcoin, the problem has experienced renewed interest from a much wider community of researchers and has seen its application expand to various novel settings, such as the so-called "permissionless" setting, where participation in the protocol is both unrestricted and unauthenticated. In fact this setting was initially studied in [42, 43], where it was shown that deterministic consensus algorithms are impossible for even a single failure but that probabilistic consensus is still feasible by suitably adapting the protocols of [13, 26]. Nevertheless, the resulting protocol required exponentially many rounds in n.

The first efficient solutions for consensus in the permissionless setting have been formally shown to be possible utilizing the Bitcoin blockchain protocol in [30], against adversaries controlling less than half of the computational power which, in a uniform configuration (meaning parties with the same computational power), corresponds to the setting of $t < n/2$ as before. At a high level these protocols (as well as the Bitcoin blockchain protocol itself) rely on a concept known as *proofs of work* (PoW), which, intuitively, enables one party to convince others that he has invested some computational power for solving a given task. While being formulated a while back [25] and used for various purposes—e,g, spam mitigation [25], sybil attacks [23], and denial of service protection [36, 5]—their role in the design of permissionless blockchain protocols [41], is arguably their most impactful application.

In the context of permissionless blockchain protocols, the way a PoW-like primitive helps 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 [30, 44, 31, 7], these works have refrained from formally defining such enabling functionality as a stand-alone cryptographic primitive and relied instead on the random oracle (RO) model [11] 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 [4, 37, 32].

In this work we fill the above gap by putting forth a formalization of a PoW-like primitive, which we call *signatures of work* (SoW). Then, we present a new permissionless consensus protocol based on SoWs. The protocol utilizes a SoW-based blockchain which may be of independent interest and standard properties of the underlying hash function, and is secure assuming an honest-majority of computational power. As a result, this protocol can be seen as an exemplar of how a permissionless signature-like primitive facilitates honest majority consensus in the same way that classical digital

[1] Recall that the protocol in [22] tolerates an arbitrary number of Byzantine faults ($n > t$), but in the version of the problem of a single sender (a.k.a. "Byzantine Generals," or just broadcast); in the case of consensus, $t < n/2$ is necessary regardless of the resources available to the parties in the protocol execution (see, e.g., [28]).

signatures imply honest-majority consensus protocols in the traditional setting.

Why *signatures* of work? We first provide some intuition behind the relevance of SoW as a useful primitive for the design of permissionless distributed protocols. Recall the main property of a digital signature in the design of classical consensus protocols: It enables parties to communicate to each other their protocol view and inputs at a certain stage of the protocol execution in a way that is transferable and non-repudiable: Bob, upon receiving Alice's signed message, can demonstrate it to Charlie in a way that the latter can be convinced unequivocally of the message's origin. It follows that Bob cannot modify Alice's messages, playing man-in-the-middle between Alice and Charlie, and thus Alice can be held accountable in case she provides conflicting views to the two parties. A SoW scheme provides a similar capability: Using a SoW, a party like Alice can invest effort into a specific protocol view and inputs, so that when Bob is presented with a SoW produced by Alice it will be infeasible for Alice to provide a conflicting view and inputs to Charlie, unless she invests twice the effort. Moreover, the above argument holds without establishing any set of identities between the parties, so for example Bob does not need to know he talks to Alice *per se* but rather to an arbitrary party that invested some effort with respect to a specific protocol view. Furthermore, exactly like digital signatures, SoWs can be chained recursively, enabling the parties to build on each other's protocol view.

While the above functionalities hint to the usefulness of SoWs in the distributed permissionless setting, formalizing and applying them properly is no simple task. Firstly, in contrast with classical signatures, there is no secret key involved in this primitive. This make sense, since in a permisionless setting signing messages using some kind of secret information is meaningless, as parties do no have any secret setup to begin with. Hence, if they are to sign any message, they should use some other kind of "resource" that only they have access to, such as their computational power. Further, in classical signatures, the exact time when the verification key becomes available to different parties is irrelevant; the key is only useful for verification up to polynomial-time differences. In the context of SoWs, however, this time is of great importance. For example, allowing a party to learn the verification key, say, two days earlier than other parties, means that this party will be able to compute two days worth of signatures more than them. Hence, in contexts where *counting* the number of generated signatures matters, as is the case in blockchain protocols, great care should be taken on guaranteeing that the verification key is "fresh" enough for the relevant application.

Our results. Our contributions are two-fold:

1) Formalization of an SoW scheme. The syntax of A SoW scheme entails four algorithms: Public parameter generation, key generation, signing and verification—$\mathsf{PPub}, \mathsf{KeyGen}$, Sign and Verify, respectively. PPub is invoked on input 1^λ, where λ is the security parameter, and outputs public security parameters pp. KeyGen is invoked on input pp, and outputs a random verification key vk. Sign is invoked on input (pp, vk, msg, h), where msg is the message to be signed, and h is the *hardness* level of the signature generation. Expectedly, Verify is invoked on input (pp, vk, msg, h, σ), where σ is (possibly) an output of Sign. We require a SoW scheme to be:

- **Correct:** As in the case of classical signatures, we require that signatures produced by Sign should be accepted by the Verify algorithm.
- **α-Successful:** This property lower-bounds the probability that an honest signer will successfully produce a SoW in a certain number of steps t_{sign}; α is a function of t_{sign} and the hardness level h.
- **t_{ver}-Verifiable:** The verifier should be able to verify a SoW in t_{ver}. (Typically, $t_{\mathsf{ver}} \ll t_{\mathsf{sign}}$.)
- **Moderately Unforgeable against Tampering and Chosen-Message Attacks ((β, ϵ)-MU-TCMA):** Akin to the property of *existential unforgeability under chosen-message attacks* of digital signatures (EU-CMA), it captures the fact that producing a sequence of SoWs, for chosen messages, does not provide an advantage to an adversary in terms of running time. Specifically, the chances to produce more than $\beta \cdot t$ SoWs in t steps (for any t) are less than ϵ. This should hold against an adversary able to *tamper* with the keys, and even in the presence of a Sign oracle.

– **Run-time independent:** This final property captures the setting where honest signers are potentially invoked on adversarial inputs and ensures that their running time enjoys some degree of independence. Specifically, the random variables defined as the running time of each Sign invocation is a set of almost independent random variables (cf. [2]).

2) Consensus from SoW. We next show it is possible to design a consensus protocol for an honest majority of computational power that can reduce to the SoW primitive. The core idea behind our protocol is as follows. First, the parties build a blockchain using SoWs in a way reminiscent of the Bitcoin blockchain. Using SoWs we show how to emulate a blockchain protocol by having parties cumulatively performing a SoW, "on top" of the current view of the protocol that incorporates the largest number of SoWs. When working to generate a block the parties include not only their input to the consensus protocol, but also the headers of "orphan" blocks that exist in forks stemming off their main chain and which have not been included so far. The header of a block contains the hash of the previous block in the chain, the signature, 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.

Our new consensus protocol relying on the SoW primitive in the setting where no private setup is available, exemplifies the constrast with consensus in the classical setting, relying on standard signatures and a PKI setup [22] (cf. [29]). It is worth noting that the only known efficient consensus protocol in this setting is given in [30], and takes advantage of a technique called "2-for-1 PoW" where 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, but which relies on the RO model in a fundamental way. Indeed, in the RO model, each witness for a PoW 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. Our new protocol dispenses with this need.

As an intermediate step in our analysis we recall the three basic properties of the blockchain data structure: (strong) common prefix, chain quality and chain growth (see [30]), and show how our SoW-based blockchain protocol satisfies them assuming, beyond the SoW security, standard collision resistance from the underlying hash function that is used to "glue" the blocks together. This is achieved as follows: We first prove that using the MU-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 SoWs of the honest parties. Then, using the α-Successful and (β, ϵ)-MU-TCMA properties in conjunction with run-time independence, we establish that summations of running times of successive Sign invocations have the *variance* needed to ensure that "uniquely successful rounds" (i.e., rounds where exactly one of the honest parties produces a SoW) happen with high density in any sufficiently long time window. Using these last two core results, and under suitable constraints for the basic SoW parameters $\alpha, \beta, \epsilon, h$ and number of parties n, we prove the security of the Bitcoin backbone protocol of [30] in the standard model just assuming our SoW primitive and the security of the underlying hash function.

Finally, and as a sanity check, we show that a SoW scheme can be easily designed and proven secure in the random oracle model and hence in practice can be instantiated by a cryptographic hash function.

Prior and related work. We have already mentioned above relevant related work regarding classical and blockchain-based consensus protocols. For a more exhaustive recent survey, refer to [29]. We also note that the focus of the paper is the original consensus problem [45, 39], and not so-called "ledger consensus" (sometimes referred to as "Nakamoto consensus"), which is an instance of the state machine replication problem [48]; see also [29] for an overview of such protocols. The idea of referencing off-chain

blocks has been considered early on in "ledger consensus" literature (see, e.g., [40, 50, 49, 14]) as a way to obtain fairness, better throughput and faster confirmation times. Our novelty, is that we leverage this technique along with SoW to build a provably secure consensus protocol, that unlike older results, is not based on the "2-for-1 POW" technique described earlier.

There have been a number of attempts to formalize a proof of work (PoW) primitive so that it is also sufficient to imply the security of a blockchain protocol. Nevertheless, such works were either informal [6, 47], or they did not produce a correctness proof for a blockchain or consensus protocol, focusing on other applications instead [16, 3, 21, 8].

Summary of differences with a previous version [33]. The most important difference of this version of the paper from [33] is the introduction of the notion of SoW, replacing the Proof of Work (PoW) notion. Moreover, many proofs have been rewritten in a clearer fashion, and the properties required from the underlying computational primitive have been simplified. Finally, additional related work regarding consensus protocols has been added.

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 the SoW primitive and its security properties are presented in Section 3. Section 4 is dedicated to applications of SoW: First, the Bitcoin backbone protocol based on (and reducing its security to) it (Section 4.1), followed by the new blockchain-based consensus protocol based on SoW (Section 4.2). The comparison of the results we get from our black-box analysis to those from the random-oracle analysis are presented in Section 5.

2 Preliminaries

In this section we introduce basic notation and definitions that we use in the rest of the text.

Definition 1. (NOTATION) For $k \in \mathbb{N}^+$, $[k]$ denotes the set $\{1, \ldots, k\}$. For strings x, z, $x||z$ is the concatenation of x and z, and $|x|$ denotes the length of x. We denote sequences by $(a_i)_{i \in I}$, where I is the index set. For a set X, $x \xleftarrow{\$} X$ denotes sampling a uniform element from X. For a distribution $\mathcal{U}$ over a set X, $x \leftarrow \mathcal{U}$ denotes sampling an element of X according to $\mathcal{U}$. We denote the statistical distance between two random variables X, Z with range $\mathcal{U}$ by $\Delta[X, Y]$, i.e., $\Delta[X, Z] = \frac{1}{2} \sum_{v \in \mathcal{U}} |\Pr[X = v] - \Pr[Z = v]|$. For $\epsilon > 0$, we say that X, Y are ϵ-*close* when $\Delta(X, Y) \leq \epsilon$.

We let λ denote the security parameter. In this paper we will follow the concrete approach [9, 12, 34, 15] 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. [15]; note also that we will be considering uniform machines). We will always assume that t is a polynomial in 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.* [34]; refer to [34] for details on using interactive RAMs in a UC-like framework, as well as to Section 4.1.1. 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 hash functions. We will make use of the following notion of security for cryptographic hash functions:

Definition 2. Let $\mathcal{H} = \{\{H_k : M(\lambda) \to Y(\lambda)\}_{k \in K(\lambda)}\}_{\lambda \in \mathbb{N}}$ 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 $\mathcal{H}$

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

Robust public transaction ledgers. A *public transaction ledger* [30] 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 $\mathsf{tx} \in \mathcal{T}$. 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.

Definition 3. A protocol Π implements a *robust public transaction ledger* if it organizes the ledger as a chain of blocks of transactions and 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.

The consensus problem. We next give a definition of the well-known consensus problem (a.k.a. Byzantine agreement) [45, 39]. 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 4. A protocol Π solves the consensus problem provided it satisfies the following 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.

3 Signatures of Work

At a high level, a *signature of work* (SoW) 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 "message." In this section we formalize this notion and present its desired security properties.

SoW syntax. We formalize the SoW notion as follows. Given a security parameter λ, let PP be the public parameter space, K the key space, M the message space, and S the signature space. With foresight, the role of the key is to provide "freshness" for the signature computation, thus authenticating that the signature was performed in the given time interval.

Definition 5. A SoW scheme consists of four algorithms $\mathsf{SoW} = (\mathsf{PPub}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ where:

- $\mathsf{PPub}(1^\lambda)$ is a randomized algorithm that takes as input the security parameter λ, and returns a set of public parameters $pp \in PP$.
- $\mathsf{KeyGen}(pp)$ is a randomized algorithm that takes as input the public parameters pp, and returns a key $vk \in K$ (See Remark 1 below on the role of keys in SoW schemes.)
- $\mathsf{Sign}(pp, vk, msg, h)$ is a randomized algorithm that takes as input public parameters $pp \in PP$, a key $vk \in K$, a message $msg \in M$ and hardness parameter $h \in \mathbb{N}$, and returns a signature (of work) $\sigma \in S$.
- $\mathsf{Verify}(pp, vk, msg, h, \sigma)$ is a deterministic algorithm that takes as input public parameters $pp \in PP$, a key $vk \in K$, message $msg \in M$, hardness parameter $h \in \mathbb{N}$ and a signature $\sigma \in S$, and returns true or false to indicate the validity of the signature.

Remark 1. SoW schemes only have a public verification key. The role of this key is to guarantee that the computational work spent in order to create a signature of work is "fresh," i.e., executed during a specific time interval (say, from the time the key became known to the signer). In contrast, classical signatures also have a secret key that serves as a trapdoor to compute signatures. In the applications we consider, the existence of trapdoor information is not meaningful, and in fact may hurt the security of the respective constructions.

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

Definition 6. We say that a SoW scheme is *correct* if for every $pp \in PP, vk \in K, h \in \mathbb{N}$, and $msg \in M$:

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

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

Definition 7. We say that a SoW scheme is t_{ver}-*verifiable*, if the Verify algorithm takes time at most t_{ver} (on all inputs).

Next, we capture the cases of a malicious signer (resp., verifier) in the context of SoWs. In the first case, the adversary's objective is to compute a number of signatures a lot faster than an honest signer would, while in the second case it is to make the honest signer take too much time to generate a signature.

We deal with malicious signers first. We put forth an attack that we will use to express a class of adversaries that attempt to forge signatures 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 signatures. The game defining the attack is shown in Figure 1; we call the corresponding security property *Moderate Unforgeability against Tampering and Chosen Message Attack* (MU-TCMA). As in the security definitions of standard signatures (e.g., EU-CMA), we allow the adversary to have access to a signing oracle $\mathcal{S}$. Every time the oracle is queried, we assume that it runs the Sign procedure with uniformly sampled randomness. A subtle point in the modeling of security in the presence of such oracle is that $\mathcal{S}$ 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 signatures, time is a side channel that may influence the adversarial strategy; in order to preserve the dependency on this side channel we will require from $\mathcal{S}$ to leak this information. We note that in the classical signatures literature, timing attacks have also been a serious consideration (see, e.g., [18]).

In addition, we require that the key used by the adversary to construct signatures be fresh, i.e., we want to avoid situations where the adversary outputs signatures that he has precomputed a long time ago. We model this by providing the fresh key after the adversary has finished running his

$\mathsf{Exp}^{\text{MU-TCMA}}_{\mathcal{A},\mathcal{F}}(1^\lambda, h, \ell)$

$\Sigma \leftarrow U_\lambda$; $pp \leftarrow \mathsf{PPub}(1^\lambda)$; *(Public parameters)*

$st \leftarrow \mathcal{A}_1(1^\lambda, \Sigma, pp)$; *(Precomputation)*

$vk \leftarrow \mathsf{KeyGen}(pp)$; *(Verification key)*

$(f_i, msg_i, \sigma_i)_{i\in[\ell]} \leftarrow \mathcal{A}_2^{\mathcal{S}(\cdot,\cdot)}(1^\lambda, vk, st)$; *(SOW computation)*

return $\bigwedge_{i=1}^{\ell} \left(\begin{array}{l} \mathsf{Verify}(pp, f_i(\Sigma, vk), msg_i, \sigma_i) \wedge \neg\mathsf{Asked}(f_i(\Sigma, vk), msg_i, \sigma_i) \\ \wedge\, (f_i \in \mathcal{F}) \wedge (\forall j \in [\ell] : f_i(\Sigma, vk) = f_j(\Sigma, vk) \rightarrow f_i = f_j) \end{array} \right)$

Figure 1: *The Moderate Unforgeability against Tampering and Chosen-Message Attack (MU-TCMA) experiment for a SoW scheme.*

precomputation phase. Further, we allow the adversary to tamper with the key by manipulating it via tampering functions belonging to a family of functions $\mathcal{F}$.

Looking ahead, the tampering function in our applications 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 vk. Moreover, the output of the adversary is deemed invalid if he tampers vk with functions f_1, f_2 in such a way that $f_1(\Sigma, vk) = f_2(\Sigma, vk)$. Otherwise, the adversary could launch a generic attack that is unrelated to the SoW scheme, and produce signatures at twice the rate of an honest signer, as follows. The adversary first finds f_1, f_2 that have this property, and then computes signatures using the tampered key $f_1(\Sigma, vk)$. The trick is that each of them will also correspond to a signature with key $f_2(\Sigma, vk)$. Hence, he effectively can double the rate at which he produces signatures.

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

We are now ready to formulate the security property of *Moderate Unforgeability against Tampering and Chosen Message Attacks* (MU-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 key vk and produces $\ell \geq \beta \cdot t$ signatures 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 signatures that are independent of vk. In the definition below, we allow β to depend on the hardness level h, and ϵ on h, t and $q_{\mathcal{S}}$, the number of queries the adversary makes to the signing oracle.

Definition 8. Let $\mathcal{F}$ be a family of functions $f : \{0,1\}^\lambda \times K \rightarrow K$. A SoW scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is (β, ϵ)-*Moderately Unforgeable against Tampering and Chosen-Message Attacks* (MU-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 $q_{\mathcal{S}}$ queries to oracle $\mathcal{S}$, the probability of $\mathcal{A}$ winning in $\mathsf{Exp}^{\text{MU-TCMA}}_{\mathcal{A},F}(1^\lambda, h, \lceil\beta(h)\cdot t\rceil)$ (Figure 1) is less than $\epsilon(h, t, q_{\mathcal{S}})$.

In the MU-TCMA definition we are going to consider tampering functions classes that at the very least preserve the unpredictability of vk. Otherwise, the adversary can generically attack any SoW scheme by predicting the tampered key and precomputing signatures. Formally, we will say that $\mathcal{F}$ is *computationally unpredictable* if the adversary, given the CRS Σ, cannot guess a value y that he will be able to "hit" when he gains access to vk 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., [35].

Definition 9. Let $\mathcal{F}$ be a family of functions $f : \{0,1\}^\lambda \times K \to K$. 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

$$\Pr_{\substack{pp \leftarrow \mathsf{PPub}(1^\lambda); \\ vk \leftarrow \mathsf{KeyGen}(pp); \\ \Sigma \leftarrow \mathcal{U}_\lambda}} \left[\begin{array}{l} (st, y) \leftarrow \mathcal{A}_1(\Sigma, pp); \\ f \leftarrow \mathcal{A}_2(st, vk) : \\ f \in \mathcal{F} \wedge f(\Sigma, vk) = y \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Next, we consider the case of attacking an honest signer. Attacking an honest signer amounts to finding a certain set of keys over which the honest signer algorithm fails to produce signatures of work sufficiently fast and regularly. We say that a SoW scheme is (t_{sign}, α)-*successful* when the probability that the signer computes a signature in t_{sign} steps is at least α.

Definition 10. We say that SoW scheme is (t_{sign}, α)-*successful* if for sufficiently large $\lambda \in \mathbb{N}$ and any $h \in \mathbb{N}$ it holds that:

$$\Pr_{\substack{pp \leftarrow \mathsf{PPub}(1^\lambda); \\ vk \leftarrow \mathsf{KeyGen}(pp); \\ msg \leftarrow M}} \left[\mathsf{Steps}_{\mathsf{Sign}}(pp, vk, msg, h) < t_{sign} \right] \geq \alpha(h).$$

Finally, in the same corrupt-verifier setting, we will require the signing time of honest signers to have some (limited) independence, which will be important for the applications we have in mind. This property, in combination with the efficiency and MU-TCMA properties, will prove crucial in ensuring that when multiple signers work together, the distribution of the number of them who succeed in producing a signature has some "good" variance and concentration properties.

Definition 11. We say that a SoW scheme has *almost-independent runtime* iff for any polynomial $p(\cdot)$, for sufficiently large $\lambda \in \mathbb{N}$, for any $pp \in PP, h \in \mathbb{N}$, there exists a set of mutually independent random variables $\{Y_i\}_{i \in [p(\lambda)]}$ such that for any $(vk_1, m_1), \ldots, (vk_{p(\lambda)}, m_{p(\lambda)}) \in K \times M$ it holds that $\Delta[(\mathsf{Steps}_{\mathsf{Sign}}(pp, vk_i, m_i, h)_i, (Y_i)_i] \leq \mathsf{negl}(\lambda)$.

Next, we turn to applications of our SoW primitive.

4 Applications

In this section we showcase two applications of SoWs, the first one being implementing robust transaction ledgers: Using our primitive and standard properties of the underlying hash function, we establish the security of the Bitcoin backbone protocol [30] without relying on random oracles. The second application is realizing consensus in the permisionless setting: We construct a new blockchain-based consensus protocol for an honest majority provably secure under the same assumptions as above, thus providing a blockchain counterpart to the classical result in the cryptographic setting with a private setup [22].

In both applications we assume a $\mathsf{SoW} = (\mathsf{PPub}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ scheme that is:

- Correct;
- (β, ϵ)-MU-TCMA with respect to any computationally unpredictable tampering function class (cf. Definition 9);
- $(t'_{\mathcal{H}}, \alpha)$-successful;
- almost run-time independent;
- t_{ver}-verifiable,

where $\epsilon \in \mathsf{negl}(\beta \cdot t)$, and $t'_{\mathcal{H}}$ is a lower bound on the running time of honest parties that we introduce later in detail. Moreover, we assume that the parameter spaces K, M, S of the scheme are equal to $\{0,1\}^{\log |K|}, \{0,1\}^*, \{0,1\}^{\log |S|}$, respectively.

Remark 2. For a SoW to be used in the context of the Bitcoin blockchain protocol choosing K, M, S as above is important due to the underlying hash-chain structure of the blockchain: the hash of each block acts as a key of the SoW scheme, thus the output of the hash function should match the key space of the SoW.

4.1 Public Transaction Ledger from Signatures of Work

In this section we take a reduction approach to the underlying cryptographic primitive—SoW, as defined in Section 3—to prove the security of the Bitcoin backbone protocol [30]. We start with some pertinent details about the model and relevant definitions. We then continue with the security proof of the Bitcoin backbone protocol in the black-box SoW setting.

4.1.1 Peer-to-peer model and definitions

In [30] 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.

Next, we describe how different parties communicate. Initially, the diffusion functionality sets the 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 $\{\text{VIEW}^{P,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}$ the random variable ensemble that corresponds to the view of party P at the end of an execution where $\mathcal{Z}$ takes z as input. We will consider stand-alone executions, hence z will always be of the form 1^λ, for $\lambda \in \mathbb{N}$. For simplicity, to denote this random variable ensemble we will use $\text{VIEW}^{P,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$. By $\text{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$ we denote the concatenation of the views of all parties. The probability space where 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 [30] and [44] a modified version of the standard simulation-based paradigm of [19] is followed, where there exist both a malicious environment and a malicious adversary. In addition, the SoW scheme (called POW in [30, 44]) 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 SoW 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 signatures, 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 [19], 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. If we don't bound θ, then the adversary will be able to launch a DOS attack and spend all the resources the honest parties have[2].

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 12. 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 $t_{\mathcal{A}}$;[3]
- the adversary sends at most θ messages per round.

Definition 13. Given a predicate Q and bounds $t_{\mathcal{A}}, \theta, t, n \in \mathbb{N}$, with $t < n$, we say that protocol Π satisfies property Q for n parties assuming the number of corruptions is bounded by t, provided that for all PPT $\mathcal{Z}, \mathcal{A}$, the probability that $Q(\text{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}})$ is false and the execution is $(t_{\mathcal{A}}, \theta)$-good is negligible in λ.

4.1.2 The Bitcoin backbone protocol

The Bitcoin backbone protocol [30], 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, and then cast the protocol making use of our SoW primitive.

We will use the terms *block* and *chain* to refer to tuples of the form $\langle s, x, \sigma\rangle$ and sequences of such tuples, respectively. The rightmost (resp. leftmost) block of chain $\mathcal{C}$ is denoted by head($\mathcal{C}$) (resp. tail($\mathcal{C}$)). Each block contains a seed, data, and a signature denoted by s, x, σ, respectively. At the start of the execution all parties have access to a common *genesis* block $B_{\mathtt{Gen}}$ that contains the public parameter pp of the SoW scheme and the key k of the hash functions H, G used. Stated differently, we assume the existence of a *common reference string* (CRS), that becomes available to all parties at

[2]This problem is extensively discussed in [4], Section 3.4.

[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.

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 $\langle s_i, x_i, \sigma_i \rangle, \langle s_{i+1}, x_{i+1}, \sigma_{i+1} \rangle$ it holds that $H_k(s_i, G_k(x_i), \sigma_i) = s_{i+1}$, (iii) each block contains a valid signature, i.e., $\mathsf{Verify}(pp, s_i, x_i, \sigma_i) = \mathsf{true}$, and (iv) the content validation predicate $V(\langle x_1, \dots, x_m \rangle)$ outputs true. We call $H_k(s_i, G_k(x_i), \sigma_i)$ the *hash of block* B_i and denote it by $H_k(B_i)$. 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 signature. If it succeeds, it diffuses the new block to the network. In more detail, each party will run the Sign procedure, 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 $\mathcal{C}$ is 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. Each honest party runs for at most $t_{\mathcal{H}}$ steps.

We summarize the modifications with respect to the original [30] protocol: In Algorithm 1 (signature of work function) the Sign function of the underlying SoW 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 SoW scheme; and in Algorithm 3 (backbone protocol) we assume that the honest parties start the execution with a "genesis" block. We leave Algorithm 4 intact.

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

```
1: function sow(x, C)
2:     s ← H(head(C))
3:     σ ← Sign(pp, s, x, h)          ▷ Run the prover of the SoW scheme.
4:     B ← ε
5:     if σ ≠ ⊥ then
6:         B ← ⟨s, x, σ⟩
7:     end if
8:     C ← CB                         ▷ Extend chain
9:     return C
10: end function
```

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 [30]. We change these definitions slightly as shown in Table 1.

4.1.3 Security properties of the blockchain

A number of desired basic properties for the blockchain were introduced in [30, 38, 44]. 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 [44], 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).

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 $\mathcal{C}$.

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

function $\mathsf{validate}(\mathcal{C})$
  $b \leftarrow V(\mathbf{x}_{\mathcal{C}}) \wedge (\mathrm{tail}(\mathcal{C}) = B_{\mathtt{Gen}})$ ▷ $\mathbf{x}_{\mathcal{C}}$ describes the contents of chain $\mathcal{C}$.
  if $b = \mathrm{True}$ **then** ▷ The chain is non-empty and meaningful w.r.t. $V(\cdot)$
    $s' \leftarrow H(\mathrm{head}(\mathcal{C}))$
    while $(\mathcal{C} \neq B_{\mathtt{Gen}}) \wedge (b = \mathrm{True})$ **do**
      $\langle s, x, \sigma \rangle \leftarrow \mathrm{head}(\mathcal{C})$
      if $\mathsf{Verify}(pp, s, x, h, \sigma) \wedge (H(\mathrm{head}(\mathcal{C})) = s')$ **then**
        $s' \leftarrow s$ ▷ Retain hash value
        $\mathcal{C} \leftarrow \mathcal{C}^{\lceil 1}$ ▷ Remove the head from $\mathcal{C}$
      else
        $b \leftarrow \mathrm{False}$
      end if
    end while
  end if
  return (b)
end function

Content validation predicate $V(\cdot)$	$V(\cdot)$ is true if its input $\langle x_1, \ldots, x_m \rangle$ is a valid ledger, i.e., it is in $\mathcal{L}$, and each x_i starts with a neutral transaction of the form $r\|\|i$, where r is a string of length $\log \lvert K \rvert$ and i is the "height" of the respective block.
Chain reading function $R(\cdot)$	$R(\cdot)$ returns the contents of the chain if they constitute a valid ledger, otherwise it is undefined.
Input contribution function $I(\cdot)$	$I(\cdot)$ 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, preceded by a neutral transaction of the form $\mathsf{KeyGen}(pp)\|\|\|\mathcal{C}\|$.

Table 1: *The instantiation of functions* $I(\cdot), V(\cdot), R(\cdot)$ *for protocol* $\Pi_{\mathsf{PL}}^{\mathsf{SOW}}$.

Definition 14 ((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 in $\mathrm{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$, 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 15 (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 $\mathrm{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$ it holds that for any k consecutive blocks of $\mathcal{C}$ the ratio of adversarial blocks is at most μ.

Further, in the derivations in [30] 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 [38] as a property under the name *chain growth.*

Definition 16 (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 $\mathrm{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$, it holds that $|\mathcal{C}_2| - |\mathcal{C}_1| \geq \tau \cdot s$.

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

$\mathcal{C} \leftarrow \mathtt{Gen}$ ▷ Initialize $\mathcal{C}$ to the genesis block.
$st \leftarrow \varepsilon$
$round \leftarrow 0$
while TRUE **do**
  $\tilde{\mathcal{C}} \leftarrow \mathsf{maxvalid}(\mathcal{C}, \text{any chain } \mathcal{C}' \text{ found in } \textsc{Receive}())$
  $\langle st, x\rangle \leftarrow I(st, \tilde{\mathcal{C}}, round, \textsc{Input}(), \textsc{Receive}())$ ▷ Determine the x-value.
  $\mathcal{C}_{\mathsf{new}} \leftarrow \mathsf{sow}(x, \tilde{\mathcal{C}})$
  if $\mathcal{C} \neq \mathcal{C}_{\mathsf{new}}$ **then**
    $\mathcal{C} \leftarrow \mathcal{C}_{\mathsf{new}}$
    $\textsc{Broadcast}(\mathcal{C})$
  end if
  $round \leftarrow round + 1$
  if $\textsc{Input}()$ contains $\textsc{Read}$ **then**
    write $R(\mathbf{x}_{\mathcal{C}})$ to $\textsc{Output}()$
  end if
end while

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

function $\mathsf{maxvalid}(\mathcal{C}_1, \ldots, \mathcal{C}_k)$
  $temp \leftarrow \varepsilon$
  for $i = 1$ to k **do**
    if $\mathsf{validate}(\mathcal{C}_i)$ **then**
      $temp \leftarrow \max(\mathcal{C}, temp)$
    end if
  end for
  return $temp$
end function

4.1.4 Security proof

We now prove that $\Pi_{\mathsf{PL}}^{\mathsf{SOW}}$ implements a robust public transaction ledger (Definition 3), assuming the underlying SoW 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 $X_j = 1$ if and only if j was a *successful round*, i.e., at least one honest party computed a SoW 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 SoW 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 Sign and Verify calls. By carefully analyzing the backbone protocol one can extract an upper bound on this value.[4] To aid our presentation, we will use $t'_{\mathcal{A}}$ and

[4]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

$t'_{\mathcal{H}}$ to denote: (i) 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 Sign routine by the honest parties, and (ii) the minimum number of steps that an honest party takes running the Sign routine per 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 $n - t$ (non-corrupted) parties will run the Sign routine for at least $t'_{\mathcal{H}}$ steps at every round.

Next, we focus on the hash functions used by 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:

$$\mathcal{H} = \{H_k : \{0,1\}^{\log|K|+\lambda+\log|S|} \to \{0,1\}^{\log|K|}\}_{k\in K'}.$$

that is *collision resistant* (Definition 2); the CRS we have already assumed will contain the key of our hash function. Moreover, as depicted in Figure 2, the protocol makes use of another hash function G to compress the input x of each block, which may be of arbitrary size. In our analysis we will require G to be collision resistant. It is well known (see, e.g., [20, 10]) that given a fixed-length collision-resistant hash function family, we can construct an arbitrary-length collision-resistant hash function family. To 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 2) is similar to the Merkle-Damgard transform [20]:

$$\mathsf{MD}_k(IV, (x_i)_{i\in[m]}) : z = IV;\ \text{for } i = 1 \text{ to } m \text{ do } z = H_k(z, x_i);\ \text{return } z,$$

where the fixed-length hash function family used is always assumed to be $\mathcal{H}$. To show that the adversary cannot find distinct chains with the same hash, we are going to take advantage of the following property of the MD transform.

Fact 1. For any non-empty valid chain $\mathcal{C} = B_1, \ldots, B_k$, where $B_i = \langle s_i, x_i, \sigma_i\rangle$, it holds that for any $j \in [k]$: $H_k(\text{head}(\mathcal{C})) = \mathsf{MD}_k(H_k(B_j), ((G_k(x_i), \sigma_i))_{i\in\{j+1,\ldots,k\}})$.

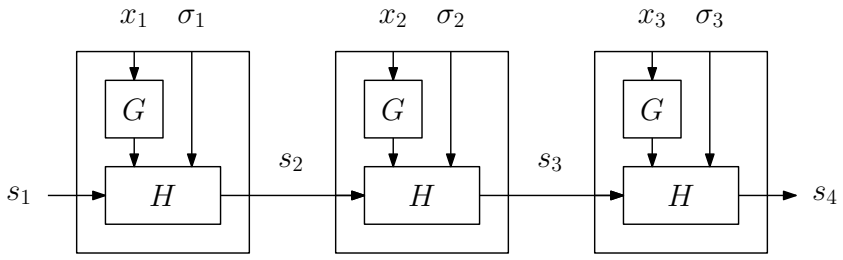

Figure 2: *The hash structure of the blocks in the Bitcoin protocol.*

Lemma 17. *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 $\mathcal{C}_1 = B_{\mathtt{Gen}}, B_1, \ldots, B_{\lfloor|\mathcal{C}_1|\rfloor}$, $\mathcal{C}_2 = B_{\mathtt{Gen}}, B'_1, \ldots, B'_{\lfloor|\mathcal{C}_2|\rfloor}$, $m_1 = ((G(x_i), \sigma_i))_{i\in\lfloor|\mathcal{C}_1|\rfloor}$ and $m_2 = ((G(x'_i), \sigma'_i))_{i\in\lfloor|\mathcal{C}_2|\rfloor}$. 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. By Fact 1, this implies that $\mathsf{MD}(H(B_{\mathtt{Gen}}), m_1) = \mathsf{MD}(H(B_{\mathtt{Gen}}), m_2)$.

case t_{bb} is at least linear in λ.

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 SoW 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

Table 2: *The parameters in our analysis.*

We will construct an adversary $\mathcal{A}'$ that breaks the collision resistance of H also with non-negligible probability. We take two cases. In the first case, $|\mathcal{C}_1| \neq |\mathcal{C}_2|$. Then, since the height of the chain is included in a fixed position in $x_{|\mathcal{C}_1|}, x'_{|\mathcal{C}_2|}$ (cf. Table 1), it follows that $x_{|\mathcal{C}_1|} \neq x'_{|\mathcal{C}_2|}$ and with overwhelming probability $G(x_{|\mathcal{C}_1|}) \neq G(x'_{|\mathcal{C}_2|})$, which in turn implies that $B_{|\mathcal{C}_1|} \neq B'_{|\mathcal{C}_2|}$. Since $H(\text{head}(\mathcal{C}_1)) = H(\text{head}(\mathcal{C}_2))$, it follows that a collision in H has been found. In the second case, where $|\mathcal{C}_1| = |\mathcal{C}_2|$, following the classical inductive argument for the MD transform, it can be shown that there exists ℓ less or equal to $|\mathcal{C}_1|$, such that $\mathsf{MD}(H(\mathsf{Gen}), ((G(x_i), \sigma_i))_{i\in[\ell]}) = \mathsf{MD}(H(\mathsf{Gen}), ((G(x'_i), \sigma'_i))_{i\in[\ell]})$ and $(G(x_\ell), \sigma_\ell) \neq (G(x'_\ell), \sigma'_\ell)$. The lemma follows.

□

The following two properties[5], introduced in [30], regarding the way blocks are connected are implied by Lemma 17.

Definition 18. 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 19. *Let $\{H_k(\cdot)\}_{k\in K}$ and $\{G_k(\cdot)\}_{k\in K}$ be collision-resistant hash functions. Then, no insertions and no copies occur with probability $1 - \mathsf{negl}(\lambda)$.*

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 MU-TCMA property. For a summary of our notation we refer to Table 2.

Lemma 20. *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 and $Z_B(S) > \beta t'_{\mathcal{A}}|S|$, is at most $\epsilon(h, t'_{\mathcal{A}} \cdot |S|, n \cdot |S|)$.*

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 MU-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 signing oracle that he has in

[5] A third property, called "prediction," also introduced in [30], is not needed in our proof as it is captured by the fact that SoW is MU-TCMA secure even in the presence of adversarial precomputation.

his disposal on the MU-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 obtaining vk. First, $\mathcal{A}'_1$ creates the genesis block and sets the fixed length hash key and the SOW 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 first 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. Moreover, it is again going to simulate honest parties behavior, from round i until round $i+s$, but in a different way. Instead of running the Sign algorithm for each non-corrupted honest party at every round, it makes a query to the signing oracle $\mathcal{S}$ with the respective parameters. Then, it checks if the honest party succeeded in making a signature in this round by comparing the number of steps needed to make this signature to the number of steps available to the party at this round. Hence, $\mathcal{A}'_2$ has to do n queries to the signing 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.

Continuing with the description of $\mathcal{A}'_2$, as shown in Figure 3, it takes as input a key vk generated from $\mathsf{KeyGen}(pp)$. We should somehow relate vk to the blocks the internal adversary is going to produce. In our reduction, this is achieved by: (i) relating the block B that party P generates at round i with vk through the input contribution function $I(\cdot)$, and (ii) by the fact that the seed of all blocks that have B as an ancestor is related to $H(B)$. In more detail, at round i, $\mathcal{A}'_2$ will use vk in the neutral transaction included in $I(\cdot)$ for P; denote by $vk||x_0$ the output of I for P at this round. If P is successful at this round and mines a block $B = \langle s_0, vk||x_0, \sigma_0\rangle$, then any block $B' = \langle s, x, \sigma\rangle$ descending B will be related to it as follows:

$$\begin{aligned} s &= \mathsf{MD}(H_\Sigma(B), ((G_\Sigma(x_i), \sigma_i))_i) \\ &= \mathsf{MD}(H_\Sigma(s, G_\Sigma(vk||x_0), \sigma_0), ((G_\Sigma(x_i), \sigma_i))_i) \\ &\stackrel{\text{def}}{=} f_{(s,\{x_i,\sigma_i\}_i)}(\Sigma, vk) \end{aligned}$$

for some $((x_i, \sigma_i))_i$ due to Fact 1. Observe, that the seed of B' is a function of Σ and vk, as required by the MU-TCMA game. In fact the tampering function class we will consider is going to be exactly the set of all these functions f. More formally, let C be the set of sequences $((x_i, \sigma_i))_i$ that correspond to a valid chain in the way described before. Then, the tampering function class we will be considering is defined as follows:

$$\mathcal{F} = \{f_{s,a}\}_{s\in\{0,1\}^\lambda, a\in C}$$

We show next, that $\mathcal{F}$ is computationally unpredictable as required by our assumption regarding the signature scheme.

Claim 1. *$\mathcal{F}$ 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

$$\Pr_{\substack{pp\leftarrow\mathsf{PPub}(1^\lambda);\\ vk\leftarrow\mathsf{KeyGen}(pp);\\ \Sigma\leftarrow\mathcal{U}_\lambda;}} \left[\begin{array}{l} (st, y) \leftarrow \mathcal{A}_1(\Sigma, pp); \\ f \leftarrow \mathcal{A}_2(st, vk) : \\ f \in \mathcal{F} \wedge f(\Sigma, vk) = y \end{array}\right]$$

is non-negligible. We are going to describe an adversary $\mathcal{A}'$ that uses $\mathcal{A}$ to break the collision resistance property of H. Given Σ, $\mathcal{A}'$ first runs $\mathcal{A}_1(\Sigma, pp)$ and obtains a prediction y and state st. Next, $\mathcal{A}'$ randomly samples vk_1, vk_2 using KeyGen and runs $\mathcal{A}_2$ twice on inputs st, vk_1 and st, vk_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 $y = f_1(\Sigma, vk_1) = f_2(\Sigma, vk_2)$. As noted earlier, this corresponds to the hash of two chains, that due to the entropy of vk_1, vk_2 and the collision resistance of G start with different honestly mined blocks. Using similar techniques as in Lemma 17, we can show that $\mathcal{A}'$ can find a collision in H using f_1, f_2, vk_1, vk_2 with non-negligible probability in λ, which is a contradiction. ⊣

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 $\epsilon(h, t'_{\mathcal{A}}s, ns)$, 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 signatures, whose keys are of the form $f(\Sigma, vk)$, for (possibly different) f's. Moreover, the event that the adversary outputs different f_i, f_j such that $f_i(\Sigma, vk) = f_j(\Sigma, vk)$, corresponds to finding chains $\mathcal{C}_1, \mathcal{C}_2$ such that $H(\mathcal{C}_1) = H(\mathcal{C}_2)$. By Lemma 17, this happens with negligible probability. Hence, $\mathcal{A}'$ will win the MU-TCMA game with respect to tampering function class $\mathcal{F}$ with probability greater than $\epsilon(h, t'_{\mathcal{A}}s, ns)$, 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 signing oracle, which is a contradiction to our initial assumption. A sketch of the reduction is given at Figure 3. □

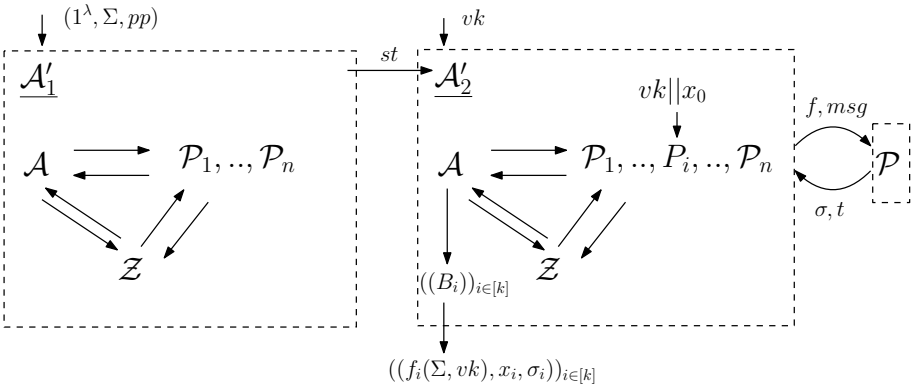

Figure 3: *The figure depicts a schematic of the reduction from the Bitcoin backbone to the MU-TCMA game of Lemma 20.*

Note that we can do exactly the same reduction without using the oracle to simulate the signing procedure of the honest parties. Then, the total running time of the second stage of $\mathcal{A}'$ is on the worst case $s \cdot (t'_{\mathcal{A}} + nt'_{\mathcal{H}})$-bounded and hence the probability he can win is $\epsilon(h, s \cdot (t'_{\mathcal{A}} + nt'_{\mathcal{H}}), 0)$. Hence, 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 21. *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 and $Z_B(S) + X_B(S) > \beta(t'_{\mathcal{A}} + nt'_{\mathcal{H}}) \cdot |S|$ is less than $\epsilon(h, |S| \cdot (t'_{\mathcal{A}} + nt'_{\mathcal{H}}), 0)$.*

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 SoW 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 22. *Let $\gamma = (n-t)\cdot\alpha(h)\cdot(1-\epsilon(h,t_{\mathcal{H}},0))^{n-1}$, $f = (1-(1-\alpha(h))^{n-t})$, and $\sigma \in (0,1)$. For any set of consecutive rounds S, with $|S| \geq \frac{\lambda}{\gamma\sigma^2}$, the following two events occur with negligible probability in λ:*

- *The number of uniquely successful rounds in S is less than $(1-\frac{\sigma}{4})\gamma\cdot|S|$;*
- *the number of successful rounds in S is less than $(1-\frac{\sigma}{4})f\cdot|S|$.*

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 Sign 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 $S = \{1,\ldots,s\}$.

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

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

It easily follows from the Successful property that $\Pr[P_{i,j}=1] \geq \alpha(h)$. Next, we show that the random variables we have defined are mutually independent.

Claim 2. *The random variable families $(P_{i,j})_{i\in[s],j\in[n]}$, $(Y_i)_{i\in[s]}$, $(X_i)_{i\in[s]}$ are mutually independent.*

Proof. First, notice that the runtime independence of the scheme implies independence of $(P_{i,j})$. We will show this for two random variables and the extension to m variables will be obvious. Let $P_1, P_2 \in (P_{i,j})_{i,j}$ and $x_1, x_2 \in \{0,1\}$, then

$$
\begin{aligned}
\Pr[P_1 =& x_1 \wedge P_2 = x_2] = \Pr[Steps_{\mathsf{Sign}}(pp,vk_1,m_1,h) \in S_1 \wedge Steps_{\mathsf{Sign}}(pp,vk_2,m_2,h) \in S_2] \\
&= \sum_{(s_1,s_2)\in S_1\times S_2} \Pr[Steps_{\mathsf{Sign}}(pp,vk_1,m_1,h) = s_1 \wedge Steps_{\mathsf{Sign}}(pp,vk_2,m_2,h) = s_2] \\
&= \sum_{(s_1,s_2)\in S_1\times S_2} \Pr[Steps_{\mathsf{Sign}}(pp,vk_1,m_1,h) = s_1] \cdot \Pr[Steps_{\mathsf{Sign}}(pp,vk_2,m_2,h) = s_2] \\
&= \sum_{s_1\in S_1} \Pr[Steps_{\mathsf{Sign}}(pp,vk_1,m_1,h) = s_1] \cdot \sum_{s_2\in S_2} \Pr[Steps_{\mathsf{Sign}}(pp,vk_2,m_2,h) = s_2] \\
&= \Pr[Steps_{\mathsf{Sign}}(pp,vk_1,m_1,h) \in S_1] \cdot \Pr[Steps_{\mathsf{Sign}}(pp,vk_2,m_2,h) \in S_2] \\
&= \Pr[P_1 = x_1] \cdot \Pr[P_2 = x_2]
\end{aligned}
$$

where S_1, S_2 are either $[0,t]$ or (t,∞) depending on x_1, x_2, and pp, vk_1, m_1, vk_2, m_2 are the parameters of the random processes. We use the independence property on the third line.

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. The same follows for $(X_i)_{i\in[s]}$. ⊣

Next, we lower bound the expected value of random variables $(Y_i)_i$ and $(X_i)_i$.

Claim 3. *It holds that for any $i \in S : \mathbb{E}[Y_i] \geq \gamma$*

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

$$\begin{aligned}
\mathbb{E}[Y_i] &= \Pr[Y_i = 1] = \Pr[\sum_{j\in[n]} P_{i,j} = 1] \\
&= \sum_{j\in[n]} \Pr[P_{i,j} = 1] \cdot \prod_{m\in[n]\setminus\{j\}} \Pr[P_{i,m} = 0] \\
&\geq \sum_{j\in[n]} \alpha(h, t_{i,j}) \prod_{m\in[n]\setminus\{j\}} (1 - Pr[P_{i,m} = 1]) \\
&\geq (n-t) \cdot \alpha(h) \cdot (1 - \epsilon(h, t_{\mathcal{H}}, 0))^{n-1} = \gamma
\end{aligned}$$

The inequalities follow from the efficiency and MU-TCMA properties. Note that $\beta t_{\mathcal{H}}$ is greater than 0, hence the adversary has to compute at least one SoW in the MU-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. ⊣

Claim 4. *It holds that for any $i \in S : \mathbb{E}[X_i] \geq f$*

Proof.

$$\begin{aligned}
\mathbb{E}[X_i] &= \Pr[X_i = 1] = \Pr[\sum_{j\in[n]} P_{i,j} \geq 1] \\
&= 1 - \Pr[\sum_{j\in[n]} P_{i,j} = 0] \\
&= 1 - \prod_{m\in[n]} \Pr[P_{i,m} = 0] \\
&\geq 1 - (1 - \alpha(h))^{n-t} = f
\end{aligned}$$

The inequality follows from the successful property. ⊣

By the linearity of expectation we have that $\mathbb{E}[Y(S)] \geq \gamma|S|$ and $\mathbb{E}[X(S)] \geq f|S|$. Since all variables in $(Y_i)_i$ and $(X_i)_i$ are mutually independent, by an application of the Chernoff Bound we have that for any $\sigma \in (0,1)$ it holds that:

$$\Pr[Y(S) \leq (1-\frac{\sigma}{4})\gamma|S|] \leq \Pr[Y(S) \leq (1-\frac{\sigma}{4})\mathbb{E}[Y(S)]] \leq e^{-\Omega(\sigma^2\gamma|S|)}$$

Similarly, we can show that $\Pr[X(S) \leq (1-\frac{\sigma}{4})f|S|] \leq e^{-\Omega(\sigma^2 f|S|)}$.

These results, with only negligible difference in probability, follows for the random variables in the real execution due to the almost runtime independence property and the fact that Y and X are functions of the joint distribution referred by this property. □

We are now ready to define the set of *typical executions* for this setting. This strategy was also followed in [30]. 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 SoW scheme.

Definition 23. [Typical execution] An execution is σ-*typical*[6] if and only if for any set S of consecutive rounds with $|S| \geq \frac{2\lambda}{\gamma\sigma^2}$, the following hold:

1. $Y(S) \geq (1-\frac{\sigma}{4})\gamma|S|$ and $X(S) \geq (1-\frac{\sigma}{4})f|S|$;
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 24. *An execution is σ-typical with overwhelming probability in λ.*

Proof. In order for an execution to not be typical on of the three points of Definition 23 must not hold. We also know that point 3 is implied by Corollary 19, and for a specific set of rounds S, point 1 is implied by Lemma 22 and point 2 is implied by Lemma 20 and Corollary 21. Hence, we can bound the probability that an execution is not typical by applying the union bound on the negation of these events over all sets of consecutive rounds of sufficiently large size, where the probability of each occurring is negligible in λ. □

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 signing algorithm. To take this into account we introduce the Computational Power Assumption, which also depends on the security parameters of the SoW scheme used. Note that γ, as defined in Lemma 22, depends on the parameters of the efficiency and MU-TCMA properties.

Computational Power 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 Computational Power 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.

[6] In [30] 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.

Lemma 25. *For any set S of at least $\frac{2\lambda}{\gamma\delta^2}$ rounds in a σ-typical execution and for any block B mined by an honest party during S, it holds that $Z_B(S) \leq (1-\frac{\delta}{4})Y(S)$.*

Proof.

$$Z_B(S) \leq \beta t'_{\mathcal{A}} \cdot |S| \leq \frac{1}{1+\delta}\gamma|S| < (1-\frac{\delta}{4})Y(S)$$

The first and the last inequality follow from the assumption that the execution is typical. The middle inequality follows from the Computational Power Assumption. □

We can now use the machinery built in [30] to prove the common prefix, chain quality and chain growth properties, with only minor changes. Using these properties we prove that the modified Bitcoin backbone protocol implements a robust transaction ledger.

Higher level properties. The notion of a typical execution is at the core of the proof of security of Bitcoin in [30]. Here, we describe the minor changes one has to do after proving the typical execution theorem with respect to the analysis of [30], in order to prove the security of the protocol in our model. We only give brief proof sketches of lemmas or theorems from [30] that are exactly the same for our own setting.

Lemma 26. *(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 27. *(Chain-Growth). In a σ-typical execution the chain-growth property holds with parameters $\tau = (1-\frac{\sigma}{4})f$ and $s \geq \frac{2\lambda}{\gamma\sigma^2}$.*

Proof. Let S be any set of at least s consecutive rounds. Then, since the execution is typical: $X(S) \geq (1-\frac{\sigma}{4})f \cdot |S| \geq \tau \cdot |S|$. By Lemma 26, 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 28. *Let B be some honest block in a δ-typical execution. Any sequence of $k \geq \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$ consecutive blocks in some chain $\mathcal{C}$, where the first block in the sequence directly descends B, have been computed in at least $k/(\gamma + \beta n t'_{\mathcal{H}})$ rounds, starting from the round that B was computed.*

Proof. For some $k \geq \frac{2\lambda}{\gamma\delta^2}(\gamma+\beta n t'_{\mathcal{H}})$, assume there is a set of rounds S' , such that $|S'| < k/(\gamma+\beta n t'_{\mathcal{H}})$, and more than k blocks that descend block B have been computed. Then, there is a set of rounds S, where $|S| \geq \frac{2\lambda}{\gamma\delta^2}$, such that $X(S) + Z_B(S) \geq k \geq |S|(\gamma + \beta n t'_{\mathcal{H}}) \geq |S|\beta(t'_{\mathcal{A}} + n t'_{\mathcal{H}})$. This contradicts the typicality of the execution, hence the lemma follows. □

Lemma 29. *(Common-prefix Lemma). Assume a δ-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 $k \geq \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$.*

Proof. In Lemma 20, 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 [30]. 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 30. *(Common-prefix). In a δ-typical execution the common-prefix property holds with parameter $k \geq \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$.*

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. □

Theorem 31. *(Chain-Quality). In a δ-typical execution the chain-quality property holds with parameter $\mu < 1 - \delta/4$ and $\ell \geq \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$.*

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 23. 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 [30].

Lemma 32. *(Persistence). It holds that Π_{PL} with $k = \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$ satisfies Persistence with overwhelming probability in λ.*

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 33. *(Liveness). It holds that Π_{PL} with $u = \frac{2k}{(1-\frac{\delta}{4})f}$ rounds and $k = \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$ satisfies Liveness with overwhelming probability in λ.*

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. □

Theorem 34. *Assuming the existence of a common reference string, a collision-resistant hash function, a* SoW *scheme with parameter spaces K, M, S of the form $\{0,1\}^{\log|K|}, \{0,1\}^*, \{0,1\}^{\log|S|}$ that satisfies $(\beta, \mathsf{negl}(\lambda))$-MU-TCMA and 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 Computational Power Assumption, protocol $\Pi^{\mathsf{SOW}}_{\mathsf{PL}}$ implements a robust public transaction ledger with parameters $u = \frac{2k}{(1-\frac{\delta}{4})f}$ and $k = \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$ except with negligible probability in λ.*

As a "sanity check," we show in Section 5 that the Bitcoin SoW 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 [30, 44]

4.2 Consensus from Signatures of Work

In this section we show how to achieve consensus (a.k.a. Byzantine agreement [45, 39]) under exactly the same assumptions used for proving the security of the Bitcoin backbone protocol in Section 4.1. In [30], consensus is achieved under the Honest Majority Assumption by using a proof-of-work 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 signatures 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 SoW schemes, as they all apply to the execution of a *single* Sign 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 [30]. This yields the first consensus protocol for honest majority reducible to a SoW 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 4.1.

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 $header(\langle s, x||vote, \sigma\rangle)$ to denote the "compressed" version of block $\langle s, x||vote, \sigma\rangle$[7], equal to $\langle s, G(x)||vote, \sigma\rangle$. 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., $H((header(B)) = H(B) = H(s, G(x)||vote, \sigma)$ (note that the header of B provides all the information needed 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 4.1. 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 wonder 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 [30], 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 Sign algorithm so that it is run on the header of the block, i.e., $\mathsf{Sign}(pp, s, G(x)||vote, h)$ and $\mathsf{Verify}(pp, s, G(x)||vote, h, \sigma)$, respectively. This way we are able to verify the validity of a block as a SoW and determine the block's vote by only knowing its header. These are 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 blockchain contents to have a special structure in order to be considered valid by the content validation predicate $V(\cdot)$ (Algorithm 5). 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.

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 checked that: (i) the contents of the block have a correct format, i.e., a vote field and list of block headers, (ii) 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, and (iii) that the first reference is dummy and includes a string r and the height of the block as required in the security analysis of Section 4.1.4 and described in Table 1.

The algorithm runs for L 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 $\Pi_{\mathsf{BA}}^{\mathsf{SOW}}$ ("BA" for

[7]We augment the block content x with a vote bit. This does not change the results of the analysis of the previous section.

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.
    $\langle r || height \rangle \leftarrow queue.top()$
    if $height \neq i$ **then**
      return False ▷ Check that the first reference has the correct block "height".
    end if
    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

Byzantine agreement). A description of the consensus protocol (specifically, the V, R, I functions) is presented in Figure 4, and an example in Figure 5. Note that all parties terminate the protocol simultaneously.

Theorem 35. *Assuming the existence of a common reference string, a collision-resistant hash function, a* SoW *scheme with parameter spaces K, M, S of the form $\{0,1\}^{\log|K|}, \{0,1\}^*, \{0,1\}^{\log|S|}$ that satisfies runtime independence and $(\beta, \mathsf{negl}(\lambda))$-MU-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 Computational Power Assumption, protocol $\Pi_{\mathsf{BA}}^{\mathsf{SOW}}$ solves consensus in $O(\frac{\lambda}{\gamma^3\delta^3})$ rounds with overwhelming probability.*

Proof. We are going to show that protocol $\Pi_{\mathsf{BA}}^{\mathsf{SOW}}$, parameterized with $k \geq \frac{2\lambda}{\gamma\delta^2}(\gamma + \beta n t'_{\mathcal{H}})$, $M = k + \frac{8k}{\delta\gamma}(\gamma + \beta t'_{\mathcal{H}})$, and $L \geq \frac{M+k}{(1-\frac{\delta}{4})\gamma}$ 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 δ-typical. Due to the chain growth property, after $L \geq \frac{2\lambda}{\gamma\delta^2}$ rounds the chain of honest parties will have length at least $M + k$ blocks, and due to the common prefix property they will all agree on the first M 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 $B_1, \ldots, B_M$ of the selected chain, have been mined by honest parties.

Content validation predicate $V(\cdot)$	The validation function $V(\cdot)$ with input $\langle x_1, \ldots, x_m \rangle$ checks that the block headers included in the chain create a block tree of valid SoW's that starts from the genesis block, and each x_i starts with a neutral transaction of the form $r\|\|i$, where r is a string of length $\log \lvert K \rvert$ and i is the "height" of the respective block (as in Table 1).
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 are not mentioned in the chain that the party is currently extending, and (ii) the party's input (i.e., 0 or 1), preceded by a neutral transaction of the form $\mathsf{KeyGen}(pp)\|\|\|\mathcal{C}\|$.

Figure 4: *The instantiation of functions* $I(\cdot), V(\cdot), R(\cdot)$ *for protocol* $\Pi_{\mathsf{BA}}^{\mathsf{SOW}}$.

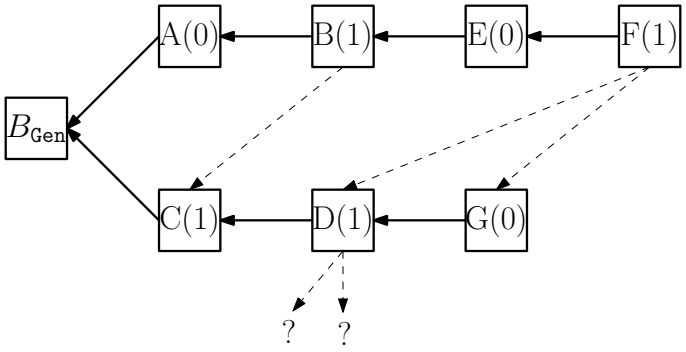

Figure 5: *The data structure maintained by* $\Pi_{\mathsf{BA}}^{\mathsf{SOW}}$. *Block B references block C, block F references blocks D and G, and block D references some invalid blocks. This is not a problem, since (i) any chain that contains D will not be selected by any party, and (ii) D's vote is correctly counted since D is a descendant of* Gen.

By the chain quality property, at least one block from $B_{M-k}, \ldots, B_M$ is honest. Assume that the last honest block in this chain has been diffused in the network at round r. Since $M - k \geq \frac{8k}{\delta\gamma}(\gamma + \beta t'_{\mathcal{H}})$, by Lemma 28 it holds that $r \geq \frac{8k}{\delta\gamma}$. After $\frac{2k}{(1-\frac{\delta}{4})\gamma}$ rounds, by the chain growth property all parties will have chain of length at least $M + k$, and by the common-prefix property all block up to the M-th position will be fixed. Hence, the last adversarial block in $B_1, \ldots, B_M$ must have been computed before round $r + \frac{2k}{(1-\frac{\delta}{4})\gamma}$.

Let $r' = r + \frac{2k}{(1-\frac{\delta}{4})\gamma}$. It remains to show that for $S_1 = \{1, \ldots, r\}$ and $S_2 = \{1, \ldots, r'\}$ it holds that $Z(S_2) < X(S_1)$:

$$Z(S_2) \leq \beta t'_{\mathcal{A}} r' < \frac{1}{1+\delta}\gamma r' \leq (1 - \frac{\delta}{4})(1 - \frac{\delta}{4})\gamma r' \leq (1 - \frac{\delta}{4})\gamma r \leq X(S_1).$$

The first and the last inequalities hold due to the fact that the execution is typical. The fourth inequality follows by the size of r. 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 s rounds of a typical execution a chain has size at most $O(s \cdot \beta(t'_A + nt'_{\mathcal{H}}) \cdot \lambda)$ bits. □

5 Comparison with the Random-Oracle Analysis

In this section, and as a sanity check, we compare the results we got from our black-box analysis to those from the random oracle analysis of [30]. To do this, we first show that the SoW scheme used in the Bitcoin protocol (call it BSOW) is secure in the random oracle model according to our definitions. Then, based on the security parameters we obtain for BSOW, we show that the security guarantees we get from our analysis of the Bitcoin backbone protocol are similar to those proved in [30, 44].

In a nutshell, Bitcoin's Sign 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, while 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 signer. To abstract the randomization of the signing 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 signature in our SoW syntax since it is produced by the signing process and is necessary for verification. Similarly, the counter is also part of the signature produced by the signing process. BSOW, 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 SoWs at the cost of one. This would be problematic for our MU-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.

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

Theorem 36. *Let $\mathcal{F}$ be a computationally unpredictable function family. Then, for $h \in [2^\lambda - 1], \sigma \in (0,1)$* BSOW *is*

- *correct;*
- *$O(\lambda)$-verifiable;*
- *$(t_{sign}, 1 - \left(\frac{h}{2^\lambda}\right)^{t_{sign}})$-successful;*
- *run-time independent;*
- *$((1+\sigma)(1-\frac{h}{2^\lambda}), \epsilon(h,t,q_{\mathcal{S}}))$-MU-TCMA secure with tampering function class $\mathcal{F}$;*

  and $\epsilon(h,t,q_{\mathcal{S}}) = \begin{cases} 1-(1-p_h)^t + \mathsf{negl}(\lambda), & \text{when } \lceil \beta t \rceil \leq 1 \\ e^{-\frac{\beta(h) t \sigma^2}{6}} + \mathsf{negl}(\lambda), & \text{otherwise} \end{cases}$

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.

Correct. By the collision resistance of G, it follows that $|\{G(vk, \sigma_1, msg) | \sigma_1 \in \{0,1\}^\lambda\}|$ is greater than λ with overwhelming probability in λ. Hence, the probability that the Sign algorithm cannot find any signature for the given parameters is upper bounded by the probability that $\lambda \cdot 2^\lambda$ different queries to

BSOW: Bitcoin's SoW implementation

- $\mathsf{PPub}(1^\lambda)$:
  1. return 1^λ

- $\mathsf{KeyGen}(pp)$:
  1. $vk \leftarrow U_\lambda$;
  2. return vk

- $\mathsf{Sign}(pp, vk, msg, h)$:
  1. while(true) {
  2. $\sigma_1 \leftarrow \mathcal{U}_\lambda$;
  3. $dig \leftarrow G(vk, \sigma_1, msg)$;
  4. for $\sigma_2 = 0^\lambda|_2$ to $1^\lambda|_2$ do {
  5. if $(H(vk, dig, \sigma_2) < 2^\lambda - h)$
  6. return $\sigma_1||\sigma_2$; } }

- $\mathsf{Verify}(pp, vk, msg, h, \sigma)$:
  1. return $(H(vk, G(vk, \sigma_1, msg), \sigma_2) \stackrel{?}{<} 2^\lambda - h)$, where $\sigma = \sigma_1||\sigma_2$.

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

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 correctness property follows.

MU-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 $\mathsf{Exp}^{\text{MU-TCMA}}_{\mathcal{A}',\mathcal{F}}$ (Figure 1) with almost the same time complexity and probability that $\mathcal{A}$ wins, without using the signing oracle $\mathcal{S}$. $\mathcal{A}'$ is going to run $\mathcal{A}$ internally, and all calls made by $\mathcal{A}$ to $\mathcal{S}$ are going to be simulated, i.e., assuming $\mathcal{A}$ queries $\mathcal{S}$ with values (f, msg), $\mathcal{A}'$ will respond with some number t' sampled from the time distribution of $\mathcal{S}$ (t' can be efficiently sampled from a geometric distribution, since queries are i.i.d Bernoulli trials) and some random signature $\sigma = (\sigma_1, \sigma_2)$, where $\sigma_2 < t'$. $\mathcal{A}'$ 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 $f(\Sigma, vk)||G(f(\Sigma, vk), \sigma_1, msg)||\sigma'_2$, where $\sigma'_2 = \sigma_2$, then $\mathcal{A}'$ will respond with the same value he responded on the initial query to $\mathcal{S}$, otherwise if $\sigma'_2 < \sigma_2$, he responds by $2^\lambda - h + (y \mod h)$, where y is the output of the real RO in this query. Note, that since σ_1 is chosen at random and RO is unpredictable, the probability that $\mathcal{A}$ has queried the RO with a string of this format before querying $\mathcal{S}$ is negligible. Hence, the view of $\mathcal{A}$ in both experiments is computationally indistinguishable, and he will output ℓ valid SoWs 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 SoW related to the queries asked to (the simulated) oracle $\mathcal{S}$, and thus it does not correspond to a winning output for $\mathcal{A}'$, i.e., there exists a query $\langle (f, msg), (\sigma_1, \sigma_2) \rangle$ on $\mathcal{S}$ and a SoW on the output of the form $\langle f', msg', (\sigma'_1, \sigma'_2) \rangle$ such that $G(f(\Sigma, vk), \sigma_1, msg) = G(f'(\Sigma, vk), \sigma'_1, msg')$ and either $msg \neq msg'$ or $\sigma_1 \neq \sigma'_1$. This implies that the adversary has found a collision in G, which only happens with negligible probability in λ. Hence, $\mathcal{A}'$ will win $\mathsf{Exp}^{\text{MU-TCMA}}_{\mathcal{A}',\mathcal{F}}$ 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 $q_{\mathcal{S}}$ i.e. $\mathcal{A}'$ can simulate the t steps taken by $\mathcal{A}_2$ in time $t' = t \cdot (1 + \log(q_{\mathcal{S}}))$; he has to maintain a heap of the queries made to $\mathcal{S}$ and search it each time the RO is queried. Note, that $\mathcal{A}'$ queries the real RO at most t times.

Let A_ℓ be the event where $\mathcal{A}'$ asks t queries the RO after receiving vk, 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_ℓ. For any $\sigma \in (0, 1)$, since $\ell = \lceil \beta t \rceil \geq (1 + \sigma) p_h t$, it follows that:

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

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

$$\Pr[A_1] = \Pr[X \geq 1] = 1 - (1 - p_h)^t$$

Let B be the event where $\mathcal{A}'_2$ outputs f, m, σ such that $f \in \mathcal{F}$ and there exists a query made to the random oracle by $\mathcal{A}'_1$ of the form $f(\Sigma, vk)||x||\sigma$. 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 $\mathcal{A}'_2(1^\lambda, vk, st)$ until it halts and possibly outputs a number of SoWs. 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 SoW 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 SoWs 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, note that if A_ℓ, B, C do not occur, it is implied that $\mathcal{A}'$ will lose in the MU-TCMA experiment. Thus:

$$\begin{aligned}
\Pr[\mathsf{Exp}^{\text{MU-TCMA}}_{\mathcal{A},\mathcal{F}}(1^\lambda, h, \ell) = 1] &= \Pr[\mathsf{Exp}^{\text{MU-TCMA}}_{\mathcal{A}',\mathcal{F}}(1^\lambda, h, \ell) = 1] \\
&\leq \Pr[A_\ell \vee B \vee C] \\
&\leq \Pr[A_\ell] + \Pr[B] + \Pr[C] \\
&\leq \Pr[A_\ell] + negl(\lambda) \\
&\leq \begin{cases} 1 - (1-p_h)^t + \mathsf{negl}(\lambda), \text{ when } \lceil \beta t \rceil \leq 1 \\ e^{-\frac{(1+\sigma)p_h t \sigma^2}{6}} + \mathsf{negl}(\lambda), \text{ otherwise} \end{cases}
\end{aligned}$$

where we have used the union bound for the third inequality.

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.

Successful. Let E be the event that in an execution of the Sign function no collisions occur. By the collision resistance property of H and G, it holds that $\Pr[\neg E] \leq negl(\lambda)$. For any $t \in \mathbb{N}$, $pp \in PP$, $vk \in K$, $msg \in M$ and $h \in \mathbb{N}$ it follows that:

$$\Pr\left[\mathsf{Steps}_{\mathsf{Sign}}(pp, vk, msg, h) < t\right] \geq 1 - (1-p_h)^t - negl(\lambda)$$

Independence. Let $\{Y_i\}_{i \in I}$ be the same as $\{X_i\}_{i \in I} = \{\mathsf{Steps}_{\mathsf{Sign}}(pp, vk_i, m_i, h_i)\}_{i \in I}$ with the only difference that the random oracle is replaced with a random function, i.e., every time Sign 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 σ_1 sampled are different among all the invocations of Sign, and that no collisions occurs in G. Note, for polynomially big I, this event happens with overwhelming probability in λ. Moreover, conditioned E, it holds that $\Pr[\{X_i\}_{i\in I} = z|E]$ is equal to $\Pr[\{Y_i\}_{i\in I} = z|E]$, for any z, 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 number of steps of Sign, 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 MU-TCMA property of $\mathtt{BSOW}$ is negligible in $\beta(h)\cdot t$ and the scheme is runtime independent, we can use Theorem 36 and obtain meaningful bounds for the γ, f quantities introduced in the previous subsection. 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. Replacing with the parameters proved for $\mathtt{BPOW}$, γ and f are equal to:

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

Both of these quantities appear in [30] and are well approximated by our results. Hence, $\Pi_{\mathsf{PL}}^{\mathtt{BSOW}}$ implements a robust transaction ledger with overwhelming probability in λ and with bounds comparable to those in [30], and achieves consensus when the honest parties have the honest majority of the computational power.

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*, volume 10677 of *Lecture Notes in Computer Science*, pages 493–526. Springer, 2017.

[4] M. Andrychowicz and S. Dziembowski. Distributed cryptography based on the proofs of work. Cryptology ePrint Archive, Report 2014/796, 2014.

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

[6] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.

[7] 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*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.

[8] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Proofs of work from worst-case assumptions. Cryptology ePrint Archive, Report 2018/559, 2018. `https://eprint.iacr.org/2018/559`.

[9] 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.

[10] 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.

[11] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93*, pages 62–73, 1993.

[12] 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.

[13] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In R. L. Probert, N. A. Lynch, and N. Santoro, editors, *PODC*, pages 27–30. ACM, 1983.

[14] I. Bentov, P. Hub'avcek, T. Moran, and A. Nadler. Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies. *IACR Cryptology ePrint Archive*, 2017:300, 2017.

[15] 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.

[16] 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.

[17] M. Borderding. Levels of authentication in distributed agreement. In Ö. Babaoglu and K. Marzullo, editors, *Distributed Algorithms, 10th International Workshop, WDAG '96, Bologna, Italy, October 9-11, 1996, Proceedings*, volume 1151 of *Lecture Notes in Computer Science*, pages 40–55. Springer, 1996.

[18] B. B. Brumley and N. Tuveri. Remote timing attacks are still practical. Cryptology ePrint Archive, Report 2011/232, 2011. `https://eprint.iacr.org/2011/232`.

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

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

[21] B. B. B. F. Dan Boneh, Joseph Bonneau. Verifiable delay functions. Cryptology ePrint Archive, Report 2018/601, 2018. `https://eprint.iacr.org/2018/601`.

[22] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[23] 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.

[24] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[25] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

[26] P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[27] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[28] M. Fitzi. *Generalized communication and security models in Byzantine agreement*. PhD thesis, ETH Zurich, 2002.

[29] J. Garay and A. Kiayias. Sok: A consensus taxonomy in the blockchain era. Cryptology ePrint Archive, Report 2018/754, 2018.

[30] J. A. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology - EUROCRYPT 2015*, 2015.

[31] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO*, pages 291–323. Springer, 2017.

[32] 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.

[33] J. A. Garay, A. Kiayias, and G. Panagiotakos. Blockchain and consensus from proofs of work without random oracles. Cryptology ePrint Archive, Report 2017/775, 2017. `https://eprint.iacr.org/2017/775`.

[34] 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.

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

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

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

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

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

[40] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. Inclusive block chain protocols. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 528–547, 2015.

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

[42] M. Okun. Agreement among unacquainted byzantine generals. In P. Fraigniaud, editor, *DISC*, volume 3724 of *Lecture Notes in Computer Science*, pages 499–500. Springer, 2005.

[43] M. Okun. Distributed computing among unacquainted processors in the presence of byzantine distributed computing among unacquainted processors in the presence of byzantine failures. Ph.D. Thesis Hebrew University of Jerusalem, 2005.

[44] 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 , Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.

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

[46] B. Pfitzmann and M. Waidner. Unconditional byzantine agreement for any number of faulty processors. In A. Finkel and M. Jantzen, editors, *STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, February 13-15, 1992, Proceedings*, volume 577 of *Lecture Notes in Computer Science*, pages 339–350. Springer, 1992.

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

[48] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, Dec. 1990.

[49] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.

[50] Y. Sompolinsky and A. Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881, 2013. `http://eprint.iacr.org/`.