The Bitcoin Backbone Protocol: Analysis and Applications

Juan Garay[1], Aggelos Kiayias[2](✉), and Nikos Leonardos[3]

[1] Yahoo Labs, Sunnyvale, CA, USA
garay@yahoo-inc.com
[2] Department of Informatics and Telecommunications, University of Athens, Athens, Greece
aggelos@di.uoa.gr
[3] LIAFA, Université Paris Diderot–Paris 7, Paris, France
nikos.leonardos@gmail.com

Abstract. Bitcoin is the first and most popular decentralized cryptocurrency to date. In this work, we extract and analyze the core of the Bitcoin protocol, which we term the Bitcoin *backbone*, and prove two of its fundamental properties which we call *common prefix* and *chain quality* in the static setting where the number of players remains fixed. Our proofs hinge on appropriate and novel assumptions on the "hashing power" of the adversary relative to network synchronicity; we show our results to be tight under high synchronization.

Next, we propose and analyze applications that can be built "on top" of the backbone protocol, specifically focusing on Byzantine agreement (BA) and on the notion of a public transaction ledger. Regarding BA, we observe that Nakamoto's suggestion falls short of solving it, and present a simple alternative which works assuming that the adversary's hashing power is bounded by 1/3. The public transaction ledger captures the essence of Bitcoin's operation as a cryptocurrency, in the sense that it guarantees the liveness and persistence of committed transactions. Based on this notion we describe and analyze the Bitcoin system as well as a more elaborate BA protocol, proving them secure assuming high network synchronicity and that the adversary's hashing power is strictly less than 1/2, while the adversarial bound needed for security decreases as the network desynchronizes.

1 Introduction

Bitcoin, introduced in [29], is a decentralized payment system that is based on maintaining a public transaction ledger in a distributed manner. The ledger is maintained by anonymous participants ("players") called *miners*, executing

A. Kiayias and N. Leonardos—Research supported by ERC project CODAMODA.
N. Leonardos—Work completed while at the National and Kapodistrian University of Athens.
The full version of this paper can be found at the Cryptology ePrint Archive [22].

International Association for Cryptologic Research 2015
E. Oswald and M. Fischlin (Eds.): EUROCRYPT 2015, Part II, LNCS 9057, pp. 281–310, 2015.
DOI: 10.1007/978-3-662-46803-6_10

a protocol that maintains and extends a distributed data structure called the *blockchain*. The protocol requires from miners to solve a "proof of work" (POW, aka "cryptographic puzzle" — see, e.g., [4,16,24,38]), which essentially amounts to brute-forcing a hash inequality based on SHA-256, in order to generate new blocks for the blockchain. The blocks that comprise the blockchain contain sets of transactions that are generated at will by owners of bitcoins, who issue transactions that credit any entity of their choice who accepts payments in bitcoin. Payers broadcast transactions and miners include the transactions they receive into the blocks they generate. Miners are rewarded for maintaining the blockchain by receiving bitcoins; it is in this manner bitcoins are created and distributed among the miners who are the first recipients of newly minted bitcoins.

An important concern in Bitcoin (or any e-payment system for that matter) is the prevention of *double-spending* attacks. Specifically, in the context of Bitcoin, a double-spending attack can occur when the attacker initially credits an account, receives service or goods by the account holder, but then manages to reorganize the transaction ledger so that the transaction that credits the account holder is reverted. In this way, the attacker keeps her bitcoin while receiving services and thus she is able to spend it again somewhere else.

In [29], Nakamoto provides an initial set of arguments of why the Bitcoin system will prevent double-spending attacks. Specifically, he argues that if a payee waits for the transaction that gives her credit to advance into the blockchain a number of k blocks, then the probability that an attacker can build an alternative blockchain that "reorganizes" the public blockchain (which contains the credit transaction) drops exponentially with k. Nakamoto argues this by modeling the attacker and the set of honest players as two competing actors performing a random walk moving toward a single direction with probabilistic steps. He demonstrates that the k blocks the payee waits are enough to ensure a negligible (in k) probability of the attacker catching up with the honest players.

Nevertheless, the above analysis can be easily seen to be oversimplified: in particular, it does not account for the fact that in Bitcoin's decentralized setting the attacker may attempt to introduce disagreement between the honest miners, thus splitting their hashing power on different POW instances. Nakamoto himself appeared to recognize the relevance of agreement in the context of Bitcoin, arguing in a forum post [30] that actually "Bitcoin's basic concept" of building and exchanging a blockchain is capable of solving Byzantine agreement (BA) [27,36] in the presence of an actively malicious adversary.[1] However a thorough analysis establishing the exact security properties of the Bitcoin system has yet to appear.

[1] In [30] Nakamoto refers to the problem as "Byzantine Generals," which is often used to refer to the single-source version of the problem, while in fact he is referring to the case where every party has an input value (Byzantine agreement). In the cryptographic setting, the problems are not equivalent in terms of the number of tolerated misbehaving parties t ($t < n$ vs. $t < n/2$, respectively).

Our Results. In this paper we extract, formally describe, and analyze the core of the Bitcoin protocol. We call this protocol the *Bitcoin backbone*, as we describe it in a way that is versatile and extensible and can be used to solve other problems as well — not just the problem of maintaining a public transaction ledger. The Bitcoin backbone protocol is executed by players that build a blockchain following the Bitcoin source code [31] and allows a set of players to maintain a blockchain in a distributed fashion. The protocol is parameterized by three external functions $V(\cdot), I(\cdot), R(\cdot)$ which we call the *input validation predicate*, the *input contribution function*, and the *chain reading function*, respectively. At a high level, $V(\cdot)$ determines the proper structure of the information that is stored into the blockchain, $I(\cdot)$ specifies how the contents of the blocks are formed by the players, and $R(\cdot)$ determines how a blockchain is supposed to be interpreted in the context of the application. Note that the structure, contents, and interpretation of the blockchain are not important for the description of the backbone protocol and are left to be specified by the three external functions above, which are application-specific (we provide examples of these functions in Section 5).

We analyze the Bitcoin backbone protocol in a static setting when the participants operate in a synchronous communication network (more details below and in Section 2) in the presence of an adversary that controls a subset of the players. We assume that the protocol is executed by a fixed number n of players; note, however, that this number is not necessarily known to the protocol participants. The players themselves *cannot* authenticate each other and therefore there is no way to know the source of a message; we capture this by allowing the adversary to "spoof" the source address of any message that is delivered. We assume that messages are eventually delivered and all parties in the network are able to synchronize in the course of a "round." The notion of round is not important for the description of the backbone protocol (which can also be executed in a loose and asynchronous fashion in the same way that Bitcoin works), however, it is important in terms of Bitcoin's inherent computational assumption regarding the players' ability to produce POWs.

Specifically, we assume that in a single round, all parties involved are allowed the same number of queries to a cryptographic hash function, as well as to communicate with the other participants. The hash function is modeled as a random oracle [6]. For simplicity we assume a "flat model," where all parties have the same quota of hashing queries per round, say q; the non-flat model where parties have differing hashing power capabilities can be easily captured by clustering the flat-model parties into larger virtual entities that are comprised by more than one flat-model player. In fact "mining pools" in Bitcoin can be thought of such aggregations of flat-model players. The adversary itself represents such pool as it controls $t < n$ players; for this reason, the adversary's quota per round is $t \cdot q$ hashing queries. Note that in this setting, the fact $t < n/2$ directly corresponds to the adversary controlling strictly less than half of the system's total "hashing power" that all players collectively harness, thus, we will use terms such as "honest majority" and "(1/2)-bounded adversary" interchangeably.

In our analysis of the Bitcoin backbone protocol we formalize and prove two fundamental properties it possesses. The properties are quantified by three parameters γ, β and f; γ and β roughly correspond to the collective hashing power per round of the honest players and the adversary, respectively, while f represents the expected number of POWs that may be found per round by the Bitcoin network participants as a whole.

- The *common prefix property.* We prove that if $\gamma > \lambda\beta$ for some $\lambda \in [1, \infty)$ that satisfies $\lambda^2 - f\lambda + 1 \geq 0$, then the blockchains maintained by the honest players will possess a large common prefix. More specifically, if two honest parties "prune" (i.e., cut off) k blocks from the end of their local chains, the probability that the resulting pruned chains will not be mutual prefixes of each other drops exponentially in k (see Definition 2 for the precise formulation). Provided that f is very close to 0 this enables us to choose λ very close to 1 and thus establish the common prefix property as long as an honest majority of participants in the flat-model setting is guaranteed (equivalently, when the adversary controls strictly less than 50% of the hashing power). On the other hand, when the network "desynchronizes" and f gets closer to 1, achieving a common prefix requires $\lambda \to \phi$, where ϕ is the golden ratio, which in turn suggests much stricter bounds on the adversarial behavior (in fact, the upper bound on the adversary for our analysis approaches 0).
- The *chain-quality property.* We prove that if $\gamma > \lambda\beta$, for some $\lambda \in [1, \infty)$, then the ratio of blocks in the chain of any honest player that are contributed by honest players is at least $(1 - \frac{1}{\lambda})$. Again observe that if λ is close to 1, we obtain that the blockchain maintained by honest players is guaranteed to have few, but still some, blocks contributed by honest players; a higher λ would be necessary to guarantee bigger percentages of blocks contributed by honest players in the blockchain. We also observe that this result is basically tight, i.e., that the adversary is capable of following a strategy (that deviates from the strategy of honest players) that enables the introduction of that many blocks in the blockchain, under a favorable (for the adversary) assumption on the propagation of adversarial blocks in the network.

While the above two security properties may seem rather abstract since they refer to properties of the data structure that is maintained distributively by the parties, we demonstrate that they are in fact quite powerful and show that the Bitcoin backbone protocol armed with the above properties can be used as a basis for solving other problems, including the problem of distributively maintaining a "robust" public transaction ledger. In Figure 1 we show how the two properties imply the properties of the applications that are explained below.

Byzantine agreement for (1/3)*-bounded adversaries.* As a first application, we show how a randomized BA protocol can be built on top of the Bitcoin backbone protocol more or less directly, and based solely on the POW assumption. We instantiate the $V(\cdot), I(\cdot), R(\cdot)$ functions so that parties form blockchains and act according to the following rules: each party i attempts to insert its own input $v_i \in \{0, 1\}$ into the blockchain; a blockchain is valid only if blocks contain elements in $\{0, 1\}$; the protocol terminates when the blockchain has reached

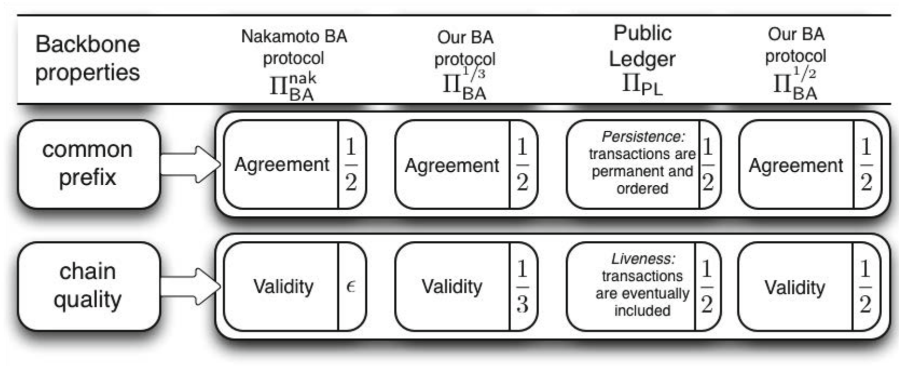
Backbone properties
Nakamoto BA protocol Π_{BA}^{nak}
Our BA protocol $\Pi_{BA}^{1/3}$
Public Ledger Π_{PL}
Our BA protocol $\Pi_{BA}^{1/2}$
common prefix
Agreement $\frac{1}{2}$
Agreement $\frac{1}{2}$
Persistence: transactions are permanent and ordered $\frac{1}{2}$
Agreement $\frac{1}{2}$
chain quality
Validity ϵ
Validity $\frac{1}{3}$
Liveness: transactions are eventually included $\frac{1}{2}$
Validity $\frac{1}{2}$

Fig. 1. An overview of the backbone protocol's applications: Nakamoto's BA protocol $\Pi_{\mathsf{BA}}^{\mathsf{nak}}$, our BA protocols $\Pi_{\mathsf{BA}}^{1/3}$ and $\Pi_{\mathsf{BA}}^{1/2}$, and the public ledger protocol Π_{PL}. All properties must be satisfied with overwhelming probability. In each box we state the name of the property as well as the maximum ratio of the adversarial hashing power that we can prove the protocol withstands (based on the corresponding backbone property). The value ϵ stands for a negligible quantity.

a sufficient length; and, the blockchain is read by the honest parties by pruning k elements from its end and returning the majority bit appearing in the resulting blockchain's prefix. We show how the common prefix property and the chain-quality property of the backbone protocol ensure Agreement and Validity (BA's basic properties; see Section 2) with high probability, thus turning the Bitcoin backbone protocol into a probabilistic BA protocol.

Observe that for the above protocol to work the chain-quality property should ensure that a majority of blocks in the blockchain originate from the honest players (otherwise Validity is lost). Our chain quality property enables this with overwhelming probability assuming the adversarial power is bounded by 1/3. This approach is different from Nakamoto's proposal [30] for BA, which, as we also show, only guarantees Validity with overwhelming probability if the adversary has a negligible amount of hashing power. On the positive side, we stress that Nakamoto's protocol fails gracefully when the adversarial power gets close to 50% as Validity can be shown with constant probability (but not overwhelming).

Public transaction ledgers and BA for honest majority. Next, we focus on how a "robust public transaction ledger" can be built on top of the Bitcoin backbone. We instantiate the $V(\cdot), I(\cdot), R(\cdot)$ functions so that parties form blockchains and act according to the following rules: each party (which in this context is called a "miner") receives a set S of transactions on its input tape and attempts to insert those in its blockchain, omitting any transactions in S that are already included in it. (A Bitcoin transaction is, for example, a statement of the type "account A credits account B a z number of bitcoins," which is signed using the secret key that corresponds to account A's Bitcoin address; each account has a unique Bitcoin address.) Reading a blockchain, on the other hand, amounts to returning the total sequence of transactions that is contained in the blockchain of the miner (and note that miners may disagree about the chain they report).

We show how the common prefix property and the chain-quality property ensure two properties needed by the ledger, which we call *Persistence* and *Liveness*, assuming an honest majority and arbitrary adversarial behavior. Persistence states that once a transaction goes more than k blocks "deep" into the blockchain of one honest player, then it will be included in *every honest player*'s blockchain with overwhelming probability, and it will be assigned a permanent position in the ledger. On the other hand, Liveness says that all transactions originating from honest account holders will eventually end up at a depth more than k blocks in an honest player's blockchain, and hence the adversary cannot perform a selective denial of service attack against honest account holders. For both properties to hold we require an honest majority (i.e., that the adversary's hashing power is strictly less than 50%) assuming high network synchronicity (i.e., that the expected number of POW solutions per round satisfies[2] $f \to 0$). If this is violated, Persistence requires stricter bounds on adversarial hashing power in order to be preserved following the bounds of the common prefix property.

In the context of Bitcoin, our analysis implies that the Bitcoin backbone provides an operational transaction ledger under the assumptions: (i) the adversary controls less than half of the total hashing power, and (ii) the network synchronizes much faster relative to the POW solution rate, (iii) digital signatures cannot be forged. On the other hand, when the network desynchronizes our results cannot support that the ledger is maintained by assuming an honest majority. This negative result is consistent with the experimental analysis provided by Decker and Wattenhoffer [15], who predicted a drop below 50% in the required adversarial bound for any setting when information propagation is problematic. Our result also provides some justification for the "slow" rate of 10-minute increments used in Bitcoin block generation. Specifically, information propagation in the Bitcoin network is on the order of seconds[3] so the ratio (essentially f) of this time window over the average 10-minute period is reasonably close to "small" and thus transaction persistence can be shown for roughly an honest majority. On the other hand, cryptocurrencies including Litecoin, Primecoin and others, reacting to the demand to offer faster transaction processing, opted for a faster response rate (some as small as 1 minute), which results in more precarious situations, e.g., $f > 0.1$, which is far from being "negligible" and thus cannot support our analysis that a common prefix would be guaranteed by merely assuming an honest majority. We finally note that the Persistence and Liveness properties we put forth and prove should not be interpreted as proofs that all Bitcoin's objectives are met. In particular, they do not guarantee that miners are properly incentivized to carry out the backbone protocol, and they can only offer guarantees in a setting of an *honest majority* amongst a fixed number of players as opposed to a setting where there is an ever changing pop-

[2] Note that we use the notation $f \to 0$ to mean that "f is close to 0" since f will be a constant in our analysis.

[3] See, for example, http://bitcoinstats.com/network/propagation/.

ulation of parties acting rationally; see related work below as well as Section 6 for further discussion.

Finally, we present a BA protocol assuming an honest majority, by suitably exploiting the properties of the robust transaction ledger above. The protocol substitutes Bitcoin's transactions with a type of transactions that are themselves based on POWs, and hence uses POWs in two distinct ways: for the maintenance of the ledger and for the generation of the transactions. We show that the ledger's Persistence implies Agreement, and that Liveness implies Validity, because assuming the ledger is maintained for long enough, a majority of transactions originating from the honest parties will be included (despite the fact that honest parties may control a minority of blocks in the blockchain). The protocol requires special care in the way it employs POWs since the adversary should be incapable of "shifting" work between the two POW tasks that it faces in each round. To solve this problem, we introduce a special strategy for POW-based protocol composition which we call "2-for-1 POWs."

Related Work. Realizing a digital currency with a centralized entity but while achieving strong privacy was proposed early on by Chaum in [13]. A number of other works improved various aspects of this concept, however the approach remained centralized. Nakamoto [29] proposed the first decentralized currency system based on POWs while relaxing the anonymity property of the payment system to mere pseudonymity. This work was followed by a multitude of other related proposals including Litecoin[4], Primecoin [26], and Zerocash [8], to mention a few. Our analysis of the Bitcoin backbone covers all these works as well, since they are based on exactly the same protocol.

It is interesting to juxtapose our positive results to the results of Eyal and Sirer [17], who introduce an attack strategy called "selfish mining" that shows how the number of blocks contributed to the blockchain by an adversary can exceed the percentage of the hashing power the adversary possesses. Their results are consistent and complementary to ours. The crux of the issue is (in our terminology) in terms of the chain-quality property, as its formulation is quite permissive: in particular we show that if the adversary controls a suitably bounded amount of hashing power, then it is also suitably bounded in terms of the number of blocks it has managed to insert in the blockchain that honest players maintain. Specifically, recall that we prove that if the hashing power of the adversary satisfies $\beta < \frac{1}{\lambda}\gamma$ (where γ roughly corresponds to the hashing power of the honest players), then the adversary may control at most a $\frac{1}{\lambda}$ percentage of the blocks in the chain. For instance, if the adversary controls up to 1/3 of the hashing power (i.e., $\lambda = 2$), then it will provably control less than 50% of the blocks in the honest players' blockchain. As it can be easily seen, this does not guarantee that the rate of a party's hashing power translates to an equal rate of rewards (recall that in Bitcoin the rewards are linearly proportional to the number of blocks that a party contributes in the chain). We define as *ideal chain quality* the property that for any coalition of parties (following any mining strategy) the percentage of blocks in the blockchain is exactly proportional to their collective

[4] http://www.litecoin.com.

hashing power. The chain quality property that we prove is not ideal and the results of [17] show that in fact there is a strategy that magnifies the percentage of a malicious coalition. Still, their mining attack does much worse than our bound. To close the gap, in the full version of the paper [22] we sketch a simple selfish mining strategy that matches our upper bound and hence our chain quality result is tight in our model[5] assuming the number of honest parties is large.

Byzantine agreement (BA, aka distributed consensus) [27,36] considers a set of n parties connected by reliable and authenticated pair-wise communication links and with possible conflicting initial inputs that wish to agree on a common output in the presence of the disruptive (even malicious) behavior of some of them. The problem has received a considerable amount of attention under various models. In this paper we are interested in randomized solutions to the problem (e.g., [7,11,18,20,25,37])[6] as in the particular setting we are in, deterministic BA algorithms are not possible. In more detail, we consider BA in the *anonymous synchronous setting*, i.e., when processors do not have identifiers and cannot correlate messages to their sources, even across rounds. This model for BA was considered by Okun, who classified it as "anonymous model without port awareness," and proved the aforementioned impossibility result, that deterministic algorithms are impossible for even a single failure [33,34]. In addition, Okun showed that probabilistic BA is feasible by suitably adapting Ben-Or's protocol [7] for the standard, non-anonymous setting (cf. [34])[7]; the protocol, however, takes exponentially many rounds. It turns out that by additionally assuming that the parties are "port-aware" (i.e., they can correlate messages to sources across rounds), deterministic protocols are possible and some more efficient solutions were proposed in [35].

The anonymous synchronous setting was also considered by Aspnes *et al.* [2] who pointed to the potential usefulness of proofs of work (e.g., [4,16,24,38]) as an identity assignment tool, in such a way that the number of identities assigned to the honest and adversarial parties can be made proportional to their aggregate computational power, respectively. For example, by assuming that the adversary's computational power is less than 50%, one of the algorithms in [2] results in a number of adversarial identities less than half of that obtained by the honest parties. By running this procedure in a pre-processing stage, it is then suggested that a standard authenticated BA protocol could be run. Such

[5] Our model allows the unfavorable event of adversarial messages winning all head-to-head races in terms of delivery with honestly generated messages in any given round.

[6] We remark that, in contrast to the approach used in typical randomized solutions to the problem, where achieving BA is reduced to (the construction of) a shared random coin, the probabilistic aspect here stems from the parties' likelihood of being able to provide proofs of work. In addition, as our analysis relies on the random oracle model [6], we are interested in computational/cryptographic solutions to the problem.

[7] Hence, BA in this setting shares a similar profile with BA in the asynchronous setting [19].

protocols, however, would require the establishment of a consistent PKI (as well as of digital signatures), details of which are not laid out in [2].

In contrast, and as mentioned above, building on our analysis of the Bitcoin backbone protocol, we propose two BA protocols solely based on POWs that operate in $O(k)$ rounds with error probability $e^{-\Omega(k)}$. The protocols solve BA with overwhelming probability under the assumption that the adversary controls less than 1/3 and 1/2 of the computational power, respectively.

The connection between Bitcoin and probabilistic BA was also considered by Miller and LaViola in [28] where they take a different approach compared to ours, by not formalizing how Bitcoin works, but rather only focusing on Nakamoto's suggestion for BA [30] as a standalone protocol. As we observe here, and also recognized in [28], Nakamoto's protocol does not quite solve BA since it does not satisfy Validity with overwhelming probability. The exact repercussions of this fact are left open in [28], while with our analysis, we provide explicit answers regarding the transaction ledger's actual properties and the level of security that the backbone realization can offer.

Finally, related to the anonymous setting, the feasibility of secure computation without authenticated links was considered by Barak *et al.* in [5] in a more extreme model where all messages sent by the parties are controlled by the adversary and can be tampered with and modified (i.e., not only source addresses can be "spoofed," but also messages' contents can be altered and messages may not be delivered). It is shown in [5] that it is possible to limit the adversary so that all he can do is to partition the network into disjoint sets, where in each set the computation is secure, and also independent of the computation in the other sets. Evidently, in such system, one cannot hope to build a global ledger.

Organization of the Paper. The rest of the paper is organized as follows. In Section 2 we present our model within which we formally express the Bitcoin backbone protocol and prove its basic properties. The backbone protocol builds "blockchains" based on a cryptographic hash function; we introduce notation for this data structure as well as the backbone protocol itself in Section 3, followed by its analysis in Section 4. Section 5 is dedicated to applications. In the full version we analyze two simple POW-based BA protocols: Nakamoto's attempt to BA and our protocol tolerating 1/3 adversarial power. We present the robust public ledger application (Bitcoin's essential task — Section 5.1) and our BA protocol for 1/2 adversarial power (Section 5.2). Due to space limitations, some of the detailed constructions, various useful remarks, and proofs, can be found in the full version of this paper [22].

2 Model and Definitions

We describe our protocols in a standard multiparty synchronous communication setting (e.g., Canetti's formulation of "real world" execution [12]) with the relaxation that the underlying communication graph is not fully connected and messages are delivered through a "diffusion" mechanism that reflects Bitcoin's peer-to-peer structure. Our adversarial model in the network is "adaptive," meaning that the adversary is allowed to take control of parties on the fly, and "rushing," meaning that in any given round the adversary gets to see all honest players' messages before deciding his strategy, and, furthermore, also allows the adversary to change the source information on every message. Note that the adversary cannot change the contents of the messages nor prevent them from being delivered. Effectively, this parallels communication over TCP/IP in the Internet where messages between parties are delivered reliably, but nevertheless malicious parties may "spoof" the source of a message they transmit and make it appear as originating from an arbitrary party (including another honest party) in the view of the receiver. This aspect of the communication model, where processors cannot correlate messages to their sources, even across arounds, was considered by Okun [33], who classified it as "anonymous model without port awareness." In this setting we use Broadcast as the message transmission command that captures the "send-to-all" functionality allowed by our communication model. Note that an adversarial sender may abuse Broadcast and attempt to confuse honest parties by sending and delivering inconsistent messages to them.

The parties' inputs are provided by the environment $\mathcal{Z}$ which also receives the parties' outputs. Parties that receive no input from the environment remain inactive, in the sense that they will not act when their turn comes in each round. The environment may provide input to a party at any round and may also modify that input from round to round. We denote by Input() the input tape of each party.

In each round, parties are able to read their input tape Input() and communication tape Receive(), perform some computation that will be suitably restricted (see below) and issue[8] a Broadcast message that is guaranteed to be delivered to all parties in the beginning of the next round. As stated above the adversary can do multiple broadcasts per round and in fact deliver to each honest party a different message or even multiple messages.

The term $\{\text{VIEW}^{P}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa,z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ denotes the random variable ensemble describing the view of party P after the completion of an execution with environment $\mathcal{Z}$, running protocol Π, and adversary $\mathcal{A}$, on auxiliary input $z \in \{0,1\}^*$. We often drop the parameters κ and z and simply refer to the ensemble by $\text{VIEW}^{P}_{\Pi,\mathcal{A},\mathcal{Z}}$ if the meaning is clear from the context. If n parties $P_1,\ldots,P_n$ execute Π, the concatenation of the view of all parties $\langle\text{VIEW}^{P_i}_{\Pi,\mathcal{A},\mathcal{Z}}\rangle_{i=1,\ldots,n}$ is denoted by $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}$. With foresight, we note that, in contrast to the standard setting where parties are aware of the number of parties executing the protocol, we are interested in protocols Π that do not make explicit use of the number of parties n or their identities. Further, note that because of the unauthenticated nature of the communication model the parties may never be certain about the number of participants in a protocol execution. Nonetheless note that the number of parties is fixed during the course of the protocol execution.

[8] For simplicity, we assume that the broadcast operation is atomic and hence the corruption of a party may not happen while the operation is taking place (cf. [21,23]).

In order to capture the parties' limited ability to produce POWs, we assume that all parties may have access to an oracle $H(\cdot)$ and allowed to perform a number of queries q *per round*, where q is a function of the security parameter κ; we refer to such parties as *q-bounded*. Note that this is a "flat-model" interpretation of the parties' computation power, where all parties are assumed equal. In the real world, different honest parties may have different "hashing power;" nevertheless, our flat-model does not sacrifice generality since one can imagine that real honest parties are simply clusters of some arbitrary number of honest flat-model parties. The adversary $\mathcal{A}$ is allowed to perform $t \cdot q$ queries per round, where $t \leq n$ is the number of corrupted parties. The environment $\mathcal{Z}$, on the other hand, is not permitted any queries to $H(\cdot)$. The rationale for this is that we would like to bound the "CPU power" [29] of the adversary to be proportionate to the number of parties it controls while making it infeasible for them to be aided by external sources or by transferring the hashing power potentially invested in concurrent or previous protocol executions. It follows that in our analysis we will focus on the "standalone" setting, where a single protocol instance is executed in isolation.

We refer to the above restrictions on the environment, the parties and the adversary as the *q-bounded synchronous setting.* The view of the parties participating in the protocol will be denoted by $\text{VIEW}^{P,H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, z)$ and the concatenation of all parties' views by $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, z)$.

In our theorems we will be concerned with *properties* of protocols Π in the q-bounded synchronous setting. Such properties will be defined as predicates over the random variable $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, z)$ by quantifying over all possible adversaries $\mathcal{A}$ and environments $\mathcal{Z}$. Note that all our protocols will only satisfy properties with a small probability of error in κ as well as in a parameter k that can be freely selected in $\{1, \ldots, \kappa\}$. The probability space is determined by the oracle $H(\cdot)$ as well as any random choices made by the protocol itself (if any). Further details about the model are given in [22].

Byzantine Agreement. As a simple illustration of the formulation above we define the properties of a Byzantine agreement (BA) protocol.

Definition 1. *A protocol Π solves BA in the q-bounded synchronous setting provided it satisfies the following two properties:*

- Agreement. *There is a round after which all honest parties return the same output if queried by the environment.*
- Validity. *The output returned by an honest party P equals the input of some party P' that is honest at the round P's output is produced.*

We note that in our protocols, the participants are capable of detecting agreement and furthermore they can also detect whether other parties detect agreement, thus *termination* can be easily achieved by all honest parties. The formulation of Validity above is intended to capture security/correctness against adaptive adversaries. The notion (specifically, the requirement that the output value be one of the honest parties' inputs) has also been called "Strong Validity" [32], but the distinction is only important in the case of non-binary inputs.

In either case, it is known that in the synchronous cryptographic setting the problem has a solution if and only if $n > |V|t$, where V is the input/decision domain [20]. Our POW-based protocols work for both versions of the problem.

3 The Bitcoin Backbone Protocol

We start by introducing blockchain notation. Let $G(\cdot), H(\cdot)$ be cryptographic hash functions with output in $\{0,1\}^\kappa$. A *block* is any triple of the form $B = \langle s, x, ctr \rangle$ where $s \in \{0,1\}^\kappa, x \in \{0,1\}^*, ctr \in \mathbb{N}$ are such that satisfy predicate $\mathsf{validblock}_q^D(B)$ defined as $(H(ctr, G(s,x)) < D) \wedge (ctr \leq q)$.

The parameter $D \in \mathbb{N}$ is also called the block's *difficulty level*. The parameter $q \in \mathbb{N}$ is a bound that in the Bitcoin implementation determines the size of the register *ctr*; in our treatment we allow this to be arbitrary, and use it to denote the maximum allowed number of hash queries in a round. We do this for convenience and our analysis applies in a straightforward manner to the case that *ctr* is restricted to the range $0 \leq ctr < 2^{32}$ and q is independent of *ctr*.

A *blockchain*, or simply a *chain* is a sequence of *blocks*. The rightmost block is the *head* of the chain, denoted $\text{head}(\mathcal{C})$. Note that the empty string ε is also a chain; by convention we set $\text{head}(\varepsilon) = \varepsilon$. A chain $\mathcal{C}$ with $\text{head}(\mathcal{C}) = \langle s', x', ctr' \rangle$ can be extended to a longer chain by appending a valid block $B = \langle s, x, ctr \rangle$ that satisfies $s = H(ctr', G(s', x'))$. In case $\mathcal{C} = \varepsilon$, by convention any valid block of the form $\langle s, x, ctr \rangle$ may extend it. In either case we have an extended chain $\mathcal{C}_{\mathsf{new}} = \mathcal{C}B$ that satisfies $\text{head}(\mathcal{C}_{\mathsf{new}}) = B$.

The *length* of a chain $\text{len}(\mathcal{C})$ is its number of blocks. Given a chain $\mathcal{C}$ that has length $\text{len}(\mathcal{C}) = n > 0$ we can define a vector $\mathbf{x}_{\mathcal{C}} = \langle x_1, \ldots, x_n \rangle$ that contains all the x-values that are stored in the chain such that x_i is the value of the i-th block.

Consider a chain $\mathcal{C}$ of length m and any nonnegative integer k. We denote by $\mathcal{C}^{\lceil k}$ the chain resulting from the "pruning" the k rightmost blocks. Note that for $k \geq \text{len}(\mathcal{C})$, $\mathcal{C}^{\lceil k} = \varepsilon$. If $\mathcal{C}_1$ is a prefix of $\mathcal{C}_2$ we write $\mathcal{C}_1 \preceq \mathcal{C}_2$.

We note that Bitcoin uses chains of variable difficulty, i.e., the value D may change across different blocks within the same chain according to some rule that is determined by the x values stored in the chain[9]. This is done to account for the fact that the number of parties (and hence the total hashing power of the system) is variable from round to round (as opposed to the unknown but fixed number of parties n we assume). See Section 6 for further discussion. We are now ready to describe the protocol.

3.1 The Backbone Protocol

The Bitcoin backbone protocol is executed by an arbitrary number of parties over an unauthenticated network. For concreteness, we assume that the number of parties running the protocol is n; however, parties need not be aware of

[9] In Bitcoin every 2016 blocks the difficulty is recalibrated according to the timestamps stored in the blocks so that the block generation rate remains at approximately 10 minutes per block.

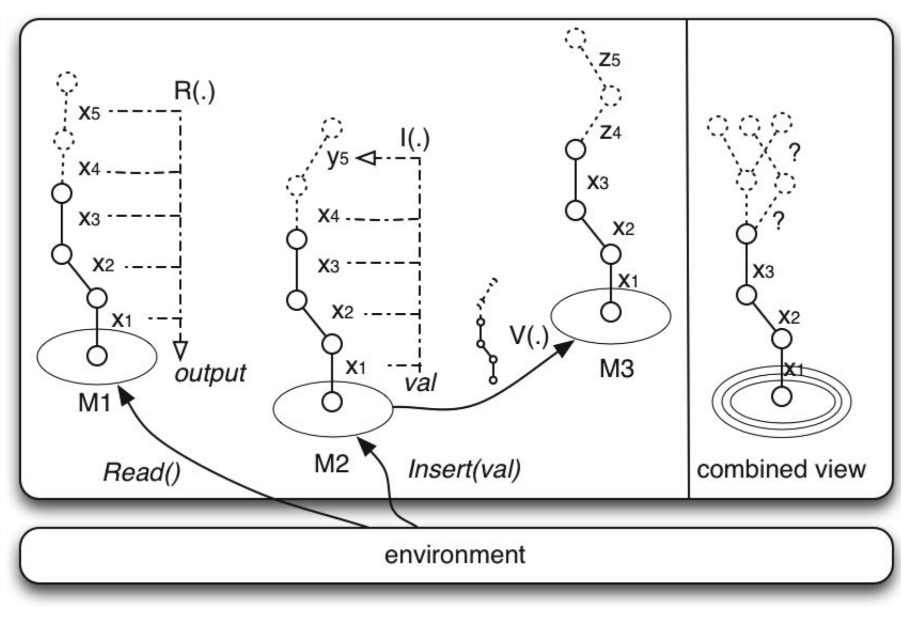

Fig. 2. Overview of the basic operation of the Bitcoin backbone protocol. Miner M_1 receives from the environment a READ instruction that results in the application of the $R(\cdot)$ function on the contents of its chain which are equal to the vector $\langle x_1, x_2, x_3, x_4, x_5 \rangle$. Miner M_2 receives from the environment an INSERT instruction and uses the function $I(\cdot)$ to determine the value y_5 that it subsequently successfully inserts in its local block chain by solving a proof of work; this results in a broadcast of the newly extended chain. Finally miner M_3 receives the newly extended chain and validates it both structurally as well as using the input validation predicate $V(\cdot)$. M_3 will adopt this chain if M_3 deems it better than its local chain as specified by the backbone protocol. Note that the joint view of M_1, M_2, M_3 is inconsistent but there is agreement on the prefix $\langle x_1, x_2, x_3 \rangle$.

this number when they execute the protocol. As mentioned in Section 2, communication over the network is achieved by utilizing a send-to-all BROADCAST functionality that is available to all parties (and maybe abused by the adversary in the sense of delivering different messages to different parties). Each party maintains a blockchain, as defined above. Each party's chain may be different, but, as we will prove, under certain well-defined conditions, the chains of honest parties will share a large common prefix. (Figure 2 depicts the local view of each party as well as the shared portion of their chains.)

In the protocol description we intentionally avoid specifying the type of values that parties try to insert in the chain, the type of chain validation they perform (beyond checking for its structural properties with respect to the hash functions $G(\cdot), H(\cdot)$), and the way they interpret the chain. These functions are handled by the external functions $V(\cdot), I(\cdot), R(\cdot)$ which are specified by the application that runs "on top" of the backbone protocol.

Chain Validation. The first algorithm, called validate performs a validation of the structural properties of a given chain $\mathcal{C}$. It is given as input the values q and D, as well as a hash function $H(\cdot)$. It is parameterized by a predicate $V(\cdot)$, called the *input validation predicate.* For each block of the chain, the algorithm checks that the proof of work is properly solved, that the counter *ctr* does not exceed q and that the hash of the previous block is properly included in the block. It further collects all the inputs from the chain's blocks and assembles them into a vector $\mathbf{x}_\mathcal{C}$. If all blocks verify and $V(\mathbf{x}_\mathcal{C})$ is true then the chain is valid; otherwise it is rejected. Note that we purposely leave the predicate $V(\cdot)$ undetermined.

Algorithm 1 The *chain validation predicate*, parameterized by q, D, the hash functions $G(\cdot), H(\cdot)$, and the *input validation predicate* $V(\cdot)$. The input is $\mathcal{C}$.

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

Chain Comparison. The objective of the second algorithm, called maxvalid, is to find the "best possible" chain when given a set of chains. The algorithm is straightforward and is parameterized by a $\max(\cdot)$ function that applies some ordering in the space of chains. The most important aspect is the chains' length, in which case $\max(\mathcal{C}_1, \mathcal{C}_2)$ will return the longest of the two. In case $\text{len}(\mathcal{C}_1) = \text{len}(\mathcal{C}_2)$, some other characteristic can be used to break the tie. In our case, $\max(\cdot, \cdot)$ will always return the first operand[10]; alternatively, other options exist, such as lexicographic order or picking a chain at random. The analysis we will perform will essentially be independent of the tie-breaking rule[11].

[10] Note that the way we deploy maxvalid, amounts to parties always giving preference to their local chain as opposed to any incoming chain. This is consistent with current Bitcoin operation; however, some debate about alternate tie-breaking rules has ensued in Bitcoin forums, e.g., see [14].

[11] It is worth to point out that the behavior of maxvalid$(\cdot)$ is associated with some stability aspects of the backbone protocol and currently there are proposals to modify

Algorithm 2 The function that finds the "best" chain, parameterized by function $\max(\cdot)$. The input is $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$.

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

Algorithm 3 The *proof of work* function, parameterized by q, D and hash functions $H(\cdot), G(\cdot)$. The input is $(x, \mathcal{C})$.

```
1: function pow($x, \mathcal{C}$)
2:     if $\mathcal{C} = \varepsilon$ then                    ▷ Determine proof of work instance
3:         $s \leftarrow 0$
4:     else
5:         $\langle s', x', ctr' \rangle \leftarrow \mathrm{head}(\mathcal{C})$
6:         $s \leftarrow H(ctr', G(s', x'))$
7:     end if
8:     $ctr \leftarrow 1$
9:     $B \leftarrow \varepsilon$
10:    $h \leftarrow G(s, x)$
11:    while ($ctr \leq q$) do
12:        if ($H(ctr, h) < D$) then                     ▷ Proof of work succeeded
13:            $B \leftarrow \langle s, x, ctr \rangle$
14:            break
15:        end if
16:        $ctr \leftarrow ctr + 1$
17:    end while
18:    $\mathcal{C} \leftarrow \mathcal{C}B$                                   ▷ Extend chain
19:    return $\mathcal{C}$
20: end function
```

Proof of Work. The third algorithm, called pow, is the main "workhorse" of the backbone protocol. It takes as input a chain and attempts to extend it via solving a proof of work. This algorithm is parameterized by two hash functions $H(\cdot), G(\cdot)$ (which in our analysis will be modeled as random oracles)[12], as well

it (e.g., by randomizing it — cf. [17]). It is an interesting question whether any improvement in our results can be achieved by randomizing the maxvalid operation.

[12] In reality the same hash function (SHA-256) instantiates both G and H; however, it is notationally more convenient to consider them as distinct.

as two positive integers q, D; q represents the number of times the algorithm is going to attempt to brute-force the hash function inequality that determines the POW instance, and D determines the "difficulty" of the POW. The algorithm works as follows. Given a chain $\mathcal{C}$ and a value x to be inserted in the chain, it hashes these values to obtain h and initializes a counter ctr. Subsequently, it increments ctr and checks to see whether $H(ctr, h) \leq D$; if a suitable ctr is found then the algorithm succeeds in solving the POW and extends chain $\mathcal{C}$ by one block inserting x as well as ctr (which serves as the POW). If no suitable ctr is found, the algorithm simply returns the chain unaltered. (See Algorithm 3.)

The Backbone Protocol. Given the three algorithms above, we are now ready to describe the Bitcoin backbone protocol. This is the protocol that is executed by the miners and which is assumed to run "indefinitely" (our security analysis will apply when the total running time is polynomial in κ). It is parameterized by two functions, the input contribution function $I(\cdot)$ and the chain reading function $R(\cdot)$, which is applied to the values stored in the chain.

Each miner maintains a local chain $\mathcal{C}$, attempting to extend it by invoking the POW algorithm pow described above. Prior to updating the chain, the miner checks its communication tape RECEIVE() to see whether a "better" chain has been received. This is done using the $\mathsf{maxvalid}$ function, depending on which the local chain is substituted.

The value that the miner attempts to insert in the chain is determined by function $I(\cdot)$. The input to $I(\cdot)$ is the state st, the current chain $\mathcal{C}$, the contents of the miner's input tape INPUT() (recall that they can be written by the environment $\mathcal{Z}$ at the beginning of any round) and communication tape RECEIVE(), as well as the current round number *round*. The protocol expects two types of entries in the input tape, READ and (INSERT, *value*); other inputs are ignored.

We purposely leave the functions $I(\cdot), R(\cdot)$ undetermined in the description of the backbone protocol, as their specifics will vary according to the application. One may choose, for example, $I(\cdot)$ to be as simple as copying the contents of the INSERT input symbols from $\mathsf{Input}()$ into x and keeping $st = \epsilon$, or performing a complex operation parsing $\mathcal{C}$ and maintaining old inputs in st. We provide explicit examples of $I(\cdot)$ and $R(\cdot)$ in Section 5. When the input x is determined, the protocol attempts to insert it into the chain $\mathcal{C}$ by invoking pow. In case the local chain $\mathcal{C}$ is modified during the above steps, the protocol transmits ("broadcasts") the new chain to the other parties. Finally, in case a READ symbol is present in the communication tape, the protocol applies function $R(\cdot)$ to its current chain and writes the result onto the output tape OUTPUT(). This way, the round ends and a new round begins, continuing indefinitely.

3.2 (Desired) Properties of the Backbone Protocol

We next define the two main properties of the backbone protocol that we will prove. The first property is called the *common prefix property* and is parameterized by a value $k \in \mathbb{N}$. It considers an arbitrary environment and adversary in the q-bounded setting, and it holds as long as any two honest parties' chains are different only in its most recent k blocks.

Algorithm 4 The Bitcoin backbone protocol, parameterized by the *input contribution function* $I(\cdot)$ and the *chain reading function* $R(\cdot)$.

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

Definition 2 (Common Prefix Property). *The common prefix property Q_{cp} with parameter $k \in \mathbb{N}$ states that for any pair of honest players P_1, P_2 maintaining the chains $\mathcal{C}_1, \mathcal{C}_2$ in* $\textsc{view}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, z)$*, it holds that*

$$\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2 \text{ and } \mathcal{C}_2^{\lceil k} \preceq \mathcal{C}_1.$$

The second property, which we call the *chain quality property*, aims at expressing the number of honest-player contributions that are contained in a sufficiently long and continuous part of an honest player's chain. Specifically, for parameters $k \in \mathbb{N}$ and $\mu \in (0, 1)$, the rate of adversarial input contributions in a continuous part of an honest party's chain is bounded by μ. This is intended to capture that at any moment that an honest player looks at a sufficiently long part of its blockchain, that part will be of sufficient "quality," i.e., the number of adversarial blocks present in that portion of the chain will be suitably bounded.

Definition 3 (Chain Quality Property). *The chain quality property Q_{cq} with parameters $\mu \in \mathbb{R}$ and $\ell \in \mathbb{N}$ states that for any honest party P with chain $\mathcal{C}$ in* $\textsc{view}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, z)$*, it holds that for any ℓ consecutive blocks of $\mathcal{C}$ the ratio of adversarial blocks is at most μ.*

It is easy to see that any set of, say, h honest parties, obtain as many blocks as their proportion of the total hashing power, i.e., h/n. We say that a protocol Π satisfies *ideal chain quality* if this is the case for adversarial parties as well, i.e., $\mu = t/n$ with respect to those parties. The ideal chain quality is not achieved by the Bitcoin backbone protocol (see [22]).

4 Analysis of the Bitcoin Backbone

We now proceed to the analysis of the protocol presented in the previous section. Let $\{0,1\}^\kappa$ be the range of $H(\cdot)$. Each party tries to provide a POW by issuing queries to $H(\cdot)$, which succeed with probability $p = D/2^\kappa$, where D is the difficulty level. By the properties of the random oracle $H(\cdot)$, any collection of queries will be treated as a collection of independent Bernoulli trials with success probability p. In order to support this we will assume that the function $I(\cdot)$ (which determines the input of the players that is to be inserted in the blockchain) ensures (at least with overwhelming probability) that the inputs are unique. There are two simple ways to enforce this: either have $I(\cdot)$ add a sufficiently long random nonce to x, or, in case parties have unique identities, it may be parameterized by it and introduce it as part of x. In either case, this value will be ignored by the other functions $V(\cdot), R(\cdot)$ as it need not be useful in the application. It is easy to see that if a κ-long nonce is used the output will be unique except for probability at most $\overline{q}^2 \cdot 2^{-\kappa}$ where $\overline{q}$ is the total number of queries submitted to the random oracle; we will ignore this small term in our analysis.

Definitions and Preliminary Lemmas. Recall that n is the number of parties, t of which can be corrupted by the adversary. We introduce the following parameters for notational convenience:

$$\alpha = pq(n-t), \quad \beta = pqt, \quad \gamma = \alpha - \alpha^2, \quad f = \alpha + \beta.$$

The first parameter, α, reflects the hashing power of the honest parties. It is an upper bound on the expected number of solutions that the honest parties compute in one round. Similarly, β, is the expected number of solutions that the corrupted parties compute in one round. Notice the asymmetry that while the honest parties will not compute more than one solution per round, a corrupted party may use all its q queries and potentially compute more than one solution. The parameter γ will serve as a lower bound on the following two probabilities. The first one is that at least one honest party computes a solution in a round:

$$1 - (1-p)^{q(n-t)} \geq 1 - e^{-\alpha} \geq \gamma;$$

we will call such round a *successful round.* The second one is the probability that exactly one honest party does so; we will call such round a *uniquely successful* round. We lower bound the probability of such a round by the probability that out of $q(n-t)$ coin tosses exactly one comes up heads. Thus, the probability is at least:

$$(n-t)qp(1-p)^{q(n-t)-1} \geq \alpha(1-\alpha+p) \geq \gamma.$$

The ratio $\alpha/\beta = (n-t)/t$ will be of interest for the analysis. When α is small (as it will be when f is small), then $\gamma \approx \alpha$ and we will be justified to concentrate on the ratio γ/β. To understand how well γ estimates the probability of a uniquely successful round, call it γ', we observe the following upper bound:

$$\gamma' = (n-t)(1-(1-p)^q)(1-p)^{q(n-t-1)} \leq (n-t)pqe^{-\alpha+pq}$$
$$\leq \alpha(1-\alpha+pq+(\alpha-pq)^2/2) = \alpha - \alpha^2(1-\tfrac{1}{n-t}) + \tfrac{\alpha^3}{2}(1-\tfrac{1}{n-t})^2,$$

where we use Bernoulli's inequality (see [22]). From this it follows that $\gamma' \leq \alpha - \alpha^2 + \alpha^3/2 + O(1/(n-t))$.

The following definition will be crucial in the analysis of the common-prefix property.

Definition 4 (Uniform rounds). *We call a round* uniform *if, at that round, every honest party invokes the* $\mathsf{pow}(\cdot)$ *algorithm with a chain of the same length (i.e.,* $\mathrm{len}(\tilde{\mathcal{C}})$ *at line 7 of Algorithm 4 is the same for all honest parties).*

We will call a query of a party *successful* if it submits a pair (ctr, h) such that $H(ctr, h) \leq D$. Without loss of generality, let $P_1, \ldots, P_t$ be the set of corrupted parties (knowledge of this set will not be used in any argument). For each round i, $j \in [q]$, and $k \in [t]$, we define Boolean random variables X_i and $Z_{ijk} \in \{0,1\}$ as follows. If at round i an honest party obtains a POW, then $X_i = 1$, otherwise $X_i = 0$. Regarding the adversary, if at round i, the j-th query of the k-th corrupted party is successful, then $Z_{ijk} = 1$, otherwise $Z_{ijk} = 0$. Further, if $X_i = 1$, we call i a *successful round*. If a round is uniform (Def. 4) and uniquely successful, we say it is a *uniquely successful uniform round*.

Next, we will prove two preliminary lemmas that will be helpful in our analysis. The first one states that, at any round, the length of any honest party's chain will be at least as large as the number of successful rounds. As a consequence, the chain of honest parties will grow at least at the rate of successful rounds. The second lemma is a simple application of Chernoff bounds and states that, with high probability, the honest parties will have, at any round, at least λ as many successful rounds as the adversary has. The usefulness of this lemma will be in showing that honest parties will be building a blockchain at a rate the adversary will find it hard to overcome.

Lemma 1. *Suppose that at round r the chain of an honest party is of length ℓ. Then, after round $s \geq r$, the chain of any honest party will have length at least $\ell + \sum_{i=r}^{s} X_i$.*

Lemma 2. *Assume $\gamma \geq (1+\delta)\lambda\beta$ for some $\delta \in (0,1)$ and $\lambda \geq 1$. The probability that during s rounds the number of successful rounds exceeds by a factor $(1+\frac{\delta}{2})\lambda$ the number of solutions computed by the adversary is at least $1 - e^{-\Omega(\delta^2 s)}$.*

We are now ready for the treatment of the protocol's properties outlined in Section 3.2.

The Common-Prefix Property. This property is established in Theorem 1, whose main argument is in turn given in Lemma 4. We start with a lemma leading to that argument. The lemma will be used to argue that uniform rounds favor the honest parties. Informally, the idea is that a uniquely successful uniform round forces an adversary trying to make honest parties' chains "diverge" to produce POWs. In the second lemma we take advantage of this, to show that if the adversary has appropriately bounded computational power, then there will be enough uniquely successful uniform rounds to prevent him from mounting a successful attack on the common-prefix property.

Lemma 3. *Consider a uniquely successful uniform round where the honest parties have chains of length $\ell - 1$. Then, in any subsequent round, there can be at most one chain $\mathcal{C}$ where the ℓ-th block was contributed by an honest party.*

Note that in order for the common-prefix property to be violated at round r, at least two honest parties should have chains $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ or $\mathcal{C}_2^{\lceil k} \not\preceq \mathcal{C}_1$. Therefore, the existence of many blocks computed at uniform rounds forces the adversary to provide as many blocks of its own. We need to show that, with high probability the adversary will fail to collect as many solutions by round r.

We say that two chains *diverge* at a given round, if the last block of their common prefix was computed before that round.

Our main lemma below asserts the following. Suppose the protocol is halted at round r and two honest parties have distinct chains $\mathcal{C}_1$ and $\mathcal{C}_2$. Then, for s large enough, the probability that $\mathcal{C}_1$ and $\mathcal{C}_2$ diverge at round $r - s$ is negligible. The idea of the proof is to upper bound the number of (valid) broadcasts that the adversary can perform during these last s rounds. Note that they are in the order of βs in expectation. The crucial observation here is that if at a given round the adversary is silent, then a uniform round follows. Therefore we expect about $(1 - \beta)s$ uniform rounds, and consequently $\gamma(1 - \beta)s$ uniquely-successful uniform rounds. Recalling Lemma 3, the adversary needs to collect $\gamma(1 - \beta)s$ POWs. Thus, in the lemma's condition we choose the relation between β and γ suitably so that the adversary is incapable of accomplishing this task, except with probability exponentially decreasing in s.

Lemma 4. *Assume $f < 1$ and $\gamma \geq (1+\delta)\lambda\beta$, for some real $\delta \in (0, 1)$ and $\lambda \geq 1$ such that $\lambda^2 - f\lambda - 1 \geq 0$. Suppose $\mathcal{C}_1$ and $\mathcal{C}_2$ are the chains of two honest parties at round r. Then, for any $s \leq r$, the probability that $\mathcal{C}_1$ and $\mathcal{C}_2$ diverge at round $r - s$ is at most $e^{-\Omega(\delta^3 s)}$.*

The above lemma is almost what we need, except that it refers to number of rounds instead of number of blocks. In order to obtain the common-prefix property we should use the properties of the blockchains of the parties themselves as the sole measure of divergence. The next theorem establishes the connection.

Theorem 1. *Assume $f < 1$ and $\gamma \geq (1 + \delta)\lambda\beta$, for some real $\delta \in (0, 1)$ and $\lambda \geq 1$ such that $\lambda^2 - f\lambda - 1 \geq 0$. Let $\mathcal{S}$ be the set of the chains of the honest parties at a given round of the backbone protocol. Then the probability that $\mathcal{S}$ does not satisfy the common-prefix property with parameter k is at most $e^{-\Omega(\delta^3 k)}$.*

The Chain-Quality Property. We now turn to the chain-quality property (Definition 3), which the theorem below establishes for a suitable bound on the number of blocks introduced by the adversary. In [22] we argue that the theorem is tight via a "selfish mining"-type strategy.

Theorem 2. *Assume $f < 1$ and $\gamma \geq (1+\delta)\lambda\beta$ for some $\delta \in (0,1)$. Suppose $\mathcal{C}$ belongs to an honest party and consider any ℓ consecutive blocks of $\mathcal{C}$. The probability that the adversary has contributed more than $(1-\frac{\delta}{3})\frac{1}{\lambda}\ell$ of these blocks is less than $e^{-\Omega(\delta^2\ell)}$.*

From the above theorem, it follows immediately that the chain quality is satisfied with parameter $\mu = \frac{1}{\lambda}$ for any segment length ℓ and probability that drops exponentially in ℓ.

5 Applications

We now show how the Bitcoin backbone protocol armed with the above properties can be used as a basis for solving other problems. We start with Byzantine agreement, as suggested by Nakamoto in a forum post [30][13]. Having defined the backbone protocol, this attempt is quite straightforward, and, at a high level, amounts to parties inserting their input values into blocks, and the validation predicate requiring that all valid chains contain the same input value, after running the protocol for a given number of rounds (alternatively, chains acquiring a certain length). However, we observe that Nakamoto's suggestion falls short of satisfying Definition 1, as Validity cannot be guaranteed with high probability, and present an alternative approach solving BA with an error that decreases exponentially in the length of the chain. Besides a change in the decision function, the most important difference is that parties build chains where they insist on inserting their local inputs, which results in a chain that contains inputs contributed by all parties. As long as the majority of blocks are contributed by honest parties one can derive Validity; this happens however provided that the adversary's power is bounded by 1/3. Due to space limitations, these protocols are presented in [22].

Next, we focus on how a "robust public transaction ledger," Bitcoin's essential task, can be built on top of the backbone protocol, followed by a more elaborate POW-based BA protocol assuming an honest majority. (Recall Figure 1.)

5.1 Robust Public Transaction Ledgers

A *public transaction ledger* is defined with respect to a set of valid ledgers $\mathcal{L}$ and a set of valid transactions $\mathcal{T}$, each one possessing an efficient membership test. A ledger $\mathbf{x} \in \mathcal{L}$ is a vector of sequences of transactions $\mathrm{tx} \in \mathcal{T}$. Each transaction tx may be associated with one or more *accounts*, denoted $a_1, a_2, \ldots$ (Here we

[13] We note that Nakamoto's description is quite informal. Here we make the most plausible interpretation of it in our framework.

will be treating transactions and accounts rather abstractly; see [22] for more concrete, Bitcoin-like notions.)

The backbone protocol parties, called *miners* in the context of this section, process sequences of transactions of the form $x = \mathrm{tx}_1 \dots \mathrm{tx}_e$ that are supposed to be incorporated into their local chain $\mathcal{C}$. The input inserted at each block of the chain $\mathcal{C}$ is the sequence x of transactions. Thus, a ledger is a vector of transaction sequences $\langle x_1, \dots, x_m \rangle$, and a chain $\mathcal{C}$ of length m contains the ledger $\mathbf{x}_{\mathcal{C}} = \langle x_1, \dots, x_m \rangle$ if the input of the j-th block in $\mathcal{C}$ is x_j.

The description and properties of the ledger protocol will be expressed relative to an oracle Txgen which will control a set of accounts by creating them and issuing transactions on their behalf. In an execution of the backbone protocol, the environment $\mathcal{Z}$ as well as the miners will have access to Txgen. Specifically, Txgen is a stateful oracle that responds to two types of queries (which we purposely only describe at a high level):

- $\mathsf{GenAccount}(1^\kappa)$: It generates an account a.
- $\mathsf{IssueTrans}(1^\kappa, \tilde{\mathrm{tx}})$: It returns a transaction tx provided that $\tilde{\mathrm{tx}}$ is some suitably formed string, or $\perp$.

We also consider a symmetric relation on $\mathcal{T}$, denoted by $C(\cdot,\cdot)$, which indicates when two transactions $\mathrm{tx}_1, \mathrm{tx}_2$ are conflicting. Valid ledgers $\mathbf{x} \in \mathcal{L}$ can never contain two conflicting transactions. We call oracle Txgen *unambiguous* if it holds that for all PPT $\mathcal{A}$, the probability that $\mathcal{A}^{\mathsf{Txgen}}$ produces a transaction tx' such that $C(\mathrm{tx}', \mathrm{tx}) = 1$, for tx issued by Txgen, is negligible in κ.

Finally, a transaction tx is called *neutral* if $C(\mathrm{tx}, \mathrm{tx}') = 0$ for any other transaction tx'. The presence of neutral transactions in the ledger can be helpful for a variety of purposes, as we will see next and in the BA protocol that we build on top of the ledger. For convenience we will assume that a single random nonce $\rho \in \{0,1\}^\kappa$ is also a valid transaction. Nonces will be neutral transactions and may be included in the ledger for the sole purpose of ensuring independence between the POW instances solved by the honest parties.

Next, we determine the three functions $V(\cdot), I(\cdot), R(\cdot)$ that will turn the backbone protocol into Π_{PL}, a protocol realizing a public transaction ledger.

We now introduce two essential properties for a protocol maintaning a public transaction ledger: (i) *Persistence* and (ii) *Liveness*. In a nutshell, Persistence states that once an honest player reports a transaction "deep enough" in the ledger, then all other honest players will report it indefinitely whenever they are asked, and at exactly the same position in the ledger (essentially, this means that all honest players agree on all the transactions that took place and in what order). In a more concrete Bitcoin-like setting (see [22]), Persistence is essential to ensure that credits are final and that they happened at a certain "time" in the system's timeline (which is implicitly defined by the ledger itself).

Note that Persistence is useful but not enough to ensure that the ledger makes progress, i.e., that transactions are eventually inserted in a chain. This is captured by the Liveness property, which states that as long as a transaction comes from an honest account holder and is provided by the environment to all honest players, then it will be inserted into the honest players' ledgers,

Input validation predicate $V(\cdot)$	$V(\langle x_1,\ldots,x_m\rangle)$ is true if and only if the vector $\langle x_1,\ldots,x_m\rangle$ is a valid ledger, i.e., $\langle x_1,\ldots,x_m\rangle \in \mathcal{L}$.
Chain reading function $R(\cdot)$	If $V(\langle x_1,\ldots,x_m\rangle) =$ True, the value $R(\mathbf{x}_\mathcal{C})$ is equal to $\langle x_1,\ldots,x_m\rangle$; undefined otherwise.
Input contribution function $I(\cdot)$	$I(st, \mathcal{C}, round, \textsc{Input}())$ operates as follows: if the input tape contains $(\textsc{Insert}, v)$, it parses v as a sequence of transactions and retains the largest subsequence $x' \preceq v$ that is valid with respect to $\mathbf{x}_\mathcal{C}$ (and whose transactions are not already included in $\mathbf{x}_\mathcal{C}$). Finally, $x = \mathrm{tx}_0 x'$ where tx_0 is a neutral random nonce transaction.

Fig. 3. The public transaction ledger protocol Π_{PL}, built on the Bitcoin backbone

assuming the environment keeps providing it as an input for a sufficient number of rounds[14].

Definition 5. *A protocol* Π *implements a* robust public transaction ledger *in the q-bounded synchronous setting if it satisfies the following two properties:*

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

We prove the two properties separately, starting with Persistence. The proof is based on the common prefix property of the backbone protocol (recall Definition 2 and Theorem 1).

Lemma 5 (Persistence). *Suppose* $f < 1$ *and* $\gamma \geq (1+\delta)\lambda\beta$*, for some real* $\delta \in (0,1)$ *and* $\lambda \geq 1$ *such that* $\lambda^2 - f\lambda - 1 \geq 0$*. Protocol* Π_{PL} *satisfies Persistence with probability* $1 - e^{-\Omega(\delta^3 k)}$*, where k is the depth parameter.*

We next prove Liveness, which is based on the chain-quality property (recall Definition 3 and Theorem 2).

[14] Observe that here we take the view that new transactions are available to all honest players and the way they are propagated is handled by the environment that feeds the backbone protocol. While this makes sense in the honest/malicious cryptographic model, it has been challenged in a model where all players are rational [3]. Analysis of the backbone protocol in a setting where transaction propagation is governed by rational players is beyond the scope of our paper.

Lemma 6 (Liveness). *Assume $f < 1$ and $\gamma \geq (1+\delta)\lambda\beta$, for some $\delta \in (0,1)$, $\lambda \in [1,\infty)$ and let $k \in \mathbb{N}$. Further, assume oracle* Txgen *is unambiguous. Then protocol Π_{PL} satisfies Liveness with wait time $u = 2k/(1-\delta)\gamma$ and depth parameter k with probability at least $1 - e^{-\Omega(\delta^2 k)}$.*

In [22] we show how to instantiate the public transaction ledger for Bitcoin, by defining the specific sets of accounts, transactions and valid ledgers.

5.2 Byzantine Agreement for Honest Majority

We now use the public transaction ledger formulation to achieve POW-based BA for an honest majority by properly instantiating the notion of a transaction, thus improving on the simple BA protocol tolerating a (1/3)-bounded adversary presented in the full version [22].

Here we consider a set of valid ledgers $\mathcal{L}$ that contain sequences of transactions of the form $\langle nonce, v, ctr\rangle$, and satisfy the predicate:

$$(H_1(ctr, G(nonce, v)) < D) \wedge (ctr \leq q), \tag{1}$$

where $H_1(\cdot), G(\cdot)$ are two hash functions as in the definition of the backbone protocol, and $v \in \{0,1\}$ is a party's input. (Recall that D is the difficulty level and q determines how many calls to $H_1(\cdot)$ a party is allowed to make per round.) To distinguish the oracles, in this section we will use $H_0(\cdot)$ to refer to the oracle used in the backbone protocol.

For the ledger we consider in this section, there will be no accounts and all transactions will be neutral — i.e., the conflict predicate $C(\cdot,\cdot)$ will be false for all pairs of transactions.

We first provide a high level description of the BA protocol assuming parties have q queries per round to each oracle $H_0(\cdot), H_1(\cdot)$. We then show how to use a single oracle $H(\cdot)$ to achieve the combined functionality of both of them while only using q queries per round.

At a high level, the protocol, $\Pi_{\mathsf{BA}}^{1/2}$, works as follows:

- *Operation:* In each round, parties run two protocols in parallel. The first protocol is protocol Π_{PL} (Fig. 3), which maintains the transaction ledger and requires q queries to the oracle $H_0(\cdot)$. The second process is a "transaction production" protocol Π_{tx}, which continuously generates transactions satisfying predicate (1).[15] The protocol makes q queries to the $H_1(\cdot)$ oracle.
- *Termination:* When the ledger reaches $2k$ blocks, a party prunes the last k blocks, collects all the unique POW transactions that are present in the ledger and returns the majority bit from the bits occuring in these transactions (note that uniqueness takes also the *nonce* of each transaction into account).

As described, protocol $\Pi_{\mathsf{BA}}^{1/2}$ does not conform to the q-bounded setting since parties require q queries to oracle $H_0(\cdot)$ and q queries to oracle $H_1(\cdot)$ to perform the computation of a single round (the setting imposes a bound of q queries to

[15] See [22] for detailed specification.

a single oracle for all parties). Note that a naïve simulation of $H_0(\cdot), H_1(\cdot)$ by a single oracle $H(\cdot)$ in the $(2q)$-bounded setting (e.g., by setting $H_b(x) = H(b, x)$) would violate the restriction imposed on each oracle individually, since nothing would prevent the adversary, for example, from querying $H_0(\cdot)$ $2q$ times. Next, we show how we can combine the two protocols into a single protocol that utilizes at most q queries to a single random oracle in a way that the adversary will remain q-bounded for each oracle. This transformation, explained below, completes the description of $\Pi_{\mathsf{BA}}^{1/2}$.

2-for-1 POWs. We now tackle the problem of how to turn a protocol operation that uses two separate POW subprocedures involving two distinct and independent oracles $H_0(\cdot), H_1(\cdot)$ into a protocol that utilizes a single oracle $H(\cdot)$ for a total number of q queries per round. Our transformation is general and works for any pair of protocols that utilize $H_0(\cdot), H_1(\cdot)$, provided that certain conditions are met (which are satisfied by protocol $\Pi_{\mathsf{BA}}^{1/2}$ above). In more detail, we consider two protocols Π_0, Π_1 that utilize a POW step as shown in Algorithm 5 in Figure 4.

Algorithm 5 POW-based protocol fragment of Π_b, $b \in \{0,1\}$ parameterized by q, D and hash functions $H_b(\cdot), G(\cdot)$, $b \in \{0,1\}$. The value w_b is determined from the protocol's context.

```
 1: ...                              ▷ Value $w_b$ is determined
 2: $ctr \leftarrow 1$
 3: $B \leftarrow \varepsilon$
 4: $h_b \leftarrow G(w_b)$
 5: while $(ctr \leq q)$ do
 6:     if $(H(ctr, h_b) < D)$ then
 7:         $B_b \leftarrow \langle w_b, ctr \rangle$
 8:         break
 9:     end if
10:     $ctr \leftarrow ctr + 1$
11: end while
12: ...                         ▷ The POW $B$ is exploited here
```

Algorithm 6 The *double proof of work* function, parameterized by q, D and hash functions $H(\cdot), G(\cdot)$ that substitutes steps 2-11 of two POW-based protocols.

```
 1: function double-pow$(w_0, w_1)$
 2:     $B_0, B_1 \leftarrow \varepsilon$
 3:     $ctr \leftarrow 1$
 4:     $h \leftarrow \langle G(w_0), G(w_1) \rangle$
 5:     while $(ctr \leq q)$ do
 6:         $u \leftarrow H(ctr, h)$
 7:         if $(u < D) \wedge (B_0 = \varepsilon)$ then
 8:             $B_0 \leftarrow \langle w_0, ctr, G(w_1) \rangle$
 9:         end if
10:         if $([u]^{\mathsf{R}} < D) \wedge (B_1 = \varepsilon)$ then
11:             $B_1 \leftarrow \langle w_1, ctr, G(w_0) \rangle$
12:         end if
13:         $ctr \leftarrow ctr + 1$
14:     end while
15:     return $\langle B_0, B_1 \rangle$
16: end function
```

Fig. 4. The 2-for-1 POW transformation

In order to achieve composition of the two protocols Π_0, Π_1 in the q-bounded setting with access to a single oracle $H(\cdot)$, we will substitute steps 2-11 in both protocols with a call to a new function, double-pow, defined below. First, observe that in Π_b, $b \in \{0,1\}$, the POW steps 2-11 operate with input w_b and produce output in B_b if the POW succeeds. The probability of obtaining a solution is $D \cdot 2^{-\kappa}$.

The modification consists in changing the structure of the POWs from pairs of the form (w, ctr) to triples of the form $(w, ctr, label)$, where $label$ is a κ-bit string that is neutral from the point of view of the proof. This will further require the modification of the verification step for POWs in both protocols Π_0, Π_1 in the following manner.

- Any verification step in Π_0 of a POW $\langle w_0, ctr\rangle$ which is of the form $H(ctr, G(w_0)) < D$, will now operate with a POW of the form $\langle w_0, ctr, label\rangle$ and will verify the relation
$$H(ctr, \langle G(w_0), label\rangle) < D.$$
- Similarly for Π_1: it will now verify the relation
$$[H(ctr, \langle label, G(w_1)\rangle)]^{\mathsf{R}} < D,$$
where $[a]^{\mathsf{R}}$ denotes the reverse of the bitstring a.

This parallel composition strategy in the form of function double-pow is shown in Algorithm 6. Either or both the solutions it returns, B_0, B_1, may be empty if no solution is found.

Protocol $\Pi_{\mathsf{BA}}^{1/2}$ will employ double-pow, which will substitute the individual POW operation of the two underlying protocols Π_0, Π_1 as defined in lines 2-11 of Algorithm 5. The correctness of the above composition strategy follows from the following simple observation.

Lemma 7. *Consider a uniform random variable U over the integers in $[0, 2^{\kappa})$ and an integer D such that $D = 2^t$ for some positive integer $t < \kappa/2$. Then, the events $(U < D)$ and $([U]^{\mathsf{R}} < D)$ are independent and they both occur with probability $D \cdot 2^{-\kappa}$.*

Theorem 3. *Assume $f < 1$ and $\gamma \geq (1+\delta)\lambda\beta$, for some real $\delta \in (0,1)$ and $\lambda \geq 1$ such that $\lambda^2 - f\lambda - 1 \geq 0$. Protocol $\Pi_{\mathsf{BA}}^{1/2}$ solves BA in $O(k)$ rounds with probability at least $1 - e^{-\Omega(\delta^3 k)}$.*

6 Summary and Directions for Future Work

In this paper we presented a formal treatment of the Bitcoin backbone, the protocol used at the core of Bitcoin's transaction ledger. We expressed and proved two properties of the backbone protocol — "common prefix" and "chain quality" — and showed how they can be used as foundations for designing Byzantine agreement and robust public transaction ledger protocols. Our results show that an honest majority among the (equally equipped) participants suffices, assuming the network synchronizes much faster than the proof of work rate ($f \to 0$ in our notation) and the proper inputs (e.g., transactions) are available to the honest majority[16], while the bound on the adversary for honest parties to reach agreement degenerates as f gets larger.

While these are encouraging results, we have demonstrated deviations that are of concern for the proper operation of Bitcoin. Importantly, we show that as the network ceases to synchronize fast enough compared to the proof-of-work rate (i.e., the worst-case time that takes honest players to "hear" each other becomes substantial compared to the time it takes to solve a proof of work), the honest majority property ceases to hold and the bound offered by our analysis that is required to obtain a robust transaction ledger approaches 0 as f approaches 1. Note that the effects of bad synchronization is in the maintenance of the common prefix property, which is the critical property for showing agreement.

A second important concern is regarding the chain quality property, where our results show that if an adversary controls a hashing power corresponding to β then the ratio of the blocks it can contribute to the blockchain is bounded but can be strictly bigger than β. When β gets close to 1/2, our bounds show that the honest players' contributions approach 0 in our security model.

The above caveats in the two basic properties of the backbone have repercussions on the Persistence and Liveness properties of the Bitcoin ledger. Firstly, they illustrate that fast information propagation amongst honest players is essential for transaction persistence. Secondly, they show that transaction liveness becomes more fragile as the adversarial power gets close to 1/2. Note that we achieve Liveness for any adversarial bound less than 1/2 but we do not assume any upper bound on the number of transactions that may be inserted in a block[17]; it is obvious that the fewer blocks the honest miners get into the blockchain the harder may be for a transaction to get through. Furthermore, the fact that chain quality demonstrably fails to preserve a one-to-one correspondence between a party's hashing power and the ratio of its contributions to the ledger point to the fact that Bitcoin's rewarding mechanism is not incentive compatible (cf. [17]). Assuming the hashing power of the honest parties γ exceeds the adversary's hashing power β by a factor λ, we show that the adversary's contributions to the ledger are bounded by $1/\lambda$ — a result we show to be tight in our rushing adversary model. In this way our results flesh out the incentive compatibility problems of the Bitcoin backbone, but (on a more positive note) they also point to the fact that honest hashing-power majority is sufficient to maintain the public ledger (under favorable network conditions), and hence suggest that the Bitcoin protocol can work as long as the majority of the miners *want it to work* (without taking into account the rationality of their decision).

The above observations apply to the setting where the number of participants is fixed. In the dynamic setting (where the number of parties running the protocol may change from round to round), given the flat model that we consider, the

[16] Our formalization is a way to formally express what perhaps was Nakamoto's intuition when he wrote about Bitcoin that "it takes advantage of the nature of information being easy to spread but hard to stifle" [31].

[17] In the current Bitcoin implementation there is an upper bound of 1MB for blocks, hence the number transactions per block is limited.

difficulty D of the blockchain may be calibrated according to the number of players n that are active in the system. If D is set by an omniscient trusted party then the analysis carries in a straightforward way but otherwise, if D is somehow calculated by the parties themselves, the adversary can try to exploit its calculation. Note that in this case the maxvalid function would need to take the difficulty's variability into account and thus choose the "most difficult" chain (as opposed to the longest). Comparing chains based on difficulty is simply done by computing the length of a chain by counting blocks proportionally to how difficult they are (for example, a block whose difficulty is two times larger than a given difficulty value would contribute twice as much in "length").

Interesting open questions include the security analysis of the Bitcoin backbone protocol in a rational setting as opposed to honest/malicious, in the dynamic setting where the parties themselves attempt to recalibrate the difficulty based on some metric (e.g., the time that has passed during the generation of a certain number of blocks), and in a concurrent/universal composition setting as opposed to standalone. Furthermore, the substitution of the random oracle assumption with a suitable computational assumption, as well as the development of backbone modifications that improve its characteristics in terms of common prefix and chain quality. In terms of the ledger application, transaction processing times (i.e., reducing the wait time parameter u in the Liveness property) is also an interesting question with implications to practice (since real world payment systems benefit greatly from fast transaction confirmation and verification). In all these cases, our work offers a formal foundation that allows analyzing the security properties of "tweaks" on the backbone protocol (such as the randomization rule of [17] or the "GHOST" rule in [39] used in ethereum[18]) towards meeting the above goals.

Another set of interesting directions include the development of other applications that may be built on top of the backbone protocol such as secure multiparty computation with properties such as fairness and guaranteed output delivery (current works in this direction, e.g., [1,9,10], assume an idealized version of the Bitcoin system).

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, Ł.: Secure multiparty computations on bitcoin. IEEE Security and Privacy (2014)
2. Aspnes, J., Jackson, C., Krishnamurthy, A.: Exposing computationally-challenged Byzantine impostors. Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science (July 2005)
3. Babaioff, M., Dobzinski, S., Oren, S., Zohar, A.: On bitcoin and red balloons. In: Faltings, B., Leyton-Brown, K., Ipeirotis, P. (eds.) EC, pp. 56–73. ACM (2012)
4. Back, A.: Hashcash (1997). http://www.cypherspace.org/hashcash
5. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure computation without authentication. J. Cryptology **24**(4), 720–760 (2011)

[18] https://www.ethereum.org/

6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS 1993, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3–5, pp. 62–73 (1993)
7. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In: Probert, R.L., Lynch, N.A., Santoro, N. (eds.) PODC, pp. 27–30. ACM (1983)
8. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. IACR Cryptology ePrint Archive **2014**, 349 (2014)
9. Bentov, I., Kumaresan, R.: How to Use Bitcoin to Design Fair Protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014)
10. Bentov, I., Kumaresan, R.: How to use bitcoin to incentivize correct computations. ACM CCS **2014**, (2014)
11. Berman, P., Garay, J.A.: Randomized distributed agreement revisited. In: Digest of Papers: FTCS-23, The Twenty-Third Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22–24, pp. 412–419. IEEE Computer Society (1993)
12. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptology **13**(1), 143–202 (2000)
13. Chaum, D.: Blind signatures for untraceable payments, pp. 199–203 (1982)
14. Cunicula. Why doesn't bitcoin use a tiebreaking rule when comparing chains of equal length? (2013) https://bitcointalk.org/index.php?topic=355644.0
15. Decker, C., Wattenhofer, R.: Information propagation in the bitcoin network. In: P2P, pp. 1–10. IEEE (2013)
16. Dwork, C., Naor, M.: Pricing via Processing or Combatting Junk Mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993)
17. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography (2014)
18. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous byzantine agreement. SIAM J. Comput. **26**(4), 873–933 (1997)
19. Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
20. Fitzi, M., Garay, J.A.: Efficient player-optimal protocols for strong and differential consensus. In: Borowsky, E., Rajsbaum, S. (eds.) PODC, pp. 211–220. ACM (2003)
21. Garay, J.A., Katz, J., Kumaresan, R., Zhou, H.: Adaptively secure broadcast, revisited. In: Gavoille, C., Fraigniaud, P., (eds.) Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6–8, pp. 179–186. ACM (2011)
22. Garay, J.A., Kiayias, A., Leonardos, N.: The Bitcoin Backbone Protocol: Analysis and Applications. IACR Cryptology ePrint Archive **2014**, 765 (2014)
23. Hirt, M., Zikas, V.: Adaptively Secure Broadcast. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 466–485. Springer, Heidelberg (2010)
24. Juels, A., Brainard, J.G.: Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: NDSS. The Internet Society (1999)
25. Katz, J., Koo, C.-Y.: On expected constant-round protocols for byzantine agreement. Journal of Computer and System Sciences **75**(2), 91–112 (2009)
26. King, S.: Primecoin: Cryptocurrency with prime number proof-of-work (July 2013). http://primecoin.io/bin/primecoin-paper.pdf

27. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
28. Miller, A., LaViola, J.J.: Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. University of Central Florida. Tech Report, CS-TR-14-01 (April 2014)
29. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. (2008) http://bitcoin.org/bitcoin.pdf
30. Nakamoto, S.: The proof-of-work chain is a solution to the byzantine generals' problem. The Cryptography Mailing List (November 2008). https://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html
31. Nakamoto, S.: Bitcoin open source implementation of p2p currency (February 2009). http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source
32. Neiger, G.: Distributed consensus revisited. Inf. Process. Lett. **49**(4), 195–201 (1994)
33. Okun, M.: Agreement Among Unacquainted Byzantine Generals. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 499–500. Springer, Heidelberg (2005)
34. Okun, M.: 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)
35. Okun, M., Barak, A.: Efficient algorithms for anonymous byzantine agreement. Theor. Comp. Sys. **42**(2), 222–238 (2008)
36. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
37. Rabin, M.O.: Randomized byzantine generals. In: FOCS, pp. 403–409. IEEE Computer Society (1983)
38. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA (1996)
39. Sompolinsky, Y., Zohar, A.: Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. IACR Cryptology ePrint Archive, 2013:881 (2013)