Bootstrapping the Blockchain, with Applications to Consensus and Fast PKI Setup

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

[1] School of Informatics, University of Edinburgh, Edinburgh, UK
akiayias@inf.ed.ac.uk, giorgos.pan@ed.ac.uk
[2] Department of Informatics and Telecommunications, University of Athens, Athens, Greece
nikos.leonardos@gmail.com
[3] Department of Computer Science and Engineering, Texas A&M University, College Station, USA
garay@cse.tamu.edu

Abstract. The Bitcoin backbone protocol (Eurocrypt 2015) extracts basic properties of Bitcoin's underlying *blockchain* data structure, such as "common prefix" and "chain quality," and shows how fundamental applications including consensus and a robust public transaction ledger can be built on top of them. The underlying assumptions are "proofs of work" (POWs), adversarial hashing power strictly less than 1/2 *and* no adversarial pre-computation—or, alternatively, the existence of an unpredictable "genesis" block.

In this paper we first show how to remove the latter assumption, presenting a "bootstrapped" Bitcoin-like blockchain protocol relying on POWs that builds genesis blocks "from scratch" in the presence of adversarial pre-computation. Importantly, the round complexity of the genesis block generation process is *independent* of the number of participants.

Next, we consider applications of our construction, including a PKI generation protocol and a consensus protocol without trusted setup assuming an honest majority (in terms of computational power). Previous results in the same setting (unauthenticated parties, no trusted setup, POWs) required a round complexity linear in the number of participants.

1 Introduction

As the first decentralized cryptocurrency, Bitcoin [33] has ignited much excitment, not only for its novel realization of a central bank-free financial instrument, but also as an alternative approach to classical distributed computing problems,

A. Kiayias—Research partly supported by ERC project CODAMODA, No. 259152, and Horizon 2020 project PANORAMIX, No. 653497.

M. Abdalla and R. Dahab (Eds.): PKC 2018, LNCS 10770, pp. 465–495, 2018.
https://doi.org/10.1007/978-3-319-76581-5_16

such as reaching agreement distributedly in the presence of misbehaving parties. Formally capturing such reach has been the intent of several recent works, notably [21], where the core of the Bitcoin protocol, called the Bitcoin *backbone*, is extracted and analyzed. The analysis includes the formulation of fundamental properties of its underlying *blockchain* data structure, which parties ("miners") maintain and try to extend by generating "proofs of work" (POW, aka "cryptographic puzzle" [3,16,24,37])[1], called *common prefix* and *chain quality*. It is then shown in [21] how applications such as consensus (aka Byzantine agreement) [31,36] and a robust public transaction ledger (i.e., Bitcoin) can be built "on top" of such properties, assuming that the hashing power of an adversary controlling a fraction of the parties is strictly less than 1/2.

Importantly, those properties hold assuming that all parties—honest and adversarial—"wake up" and start computing at the same time, or, alternatively, that they compute on a common random string only made available at the exact time when the protocol execution is to begin (see further discussion under related work below). Indeed, the coinbase parameter in Bitcoin's "genesis" block, hardcoded into the software, contains text from *The Times* 03/Jan/2009 issue [5], arguably unpredictable.

While satisfactory in some cases, such a trusted setup/behavioral assumption might be unrealistic in other POW-based systems where details may have been released a lot earlier than the actual time when the system starts to run. A case in point is Ethereum, which was discussed for over a year before the system officially kicked off. That's from a practical point of view. At a foundational level, one would in addition like to understand what kind of cryptographic primitives can be realized without any trusted setup assumption and based on POWs, and whether that is in particular the case for the Bitcoin backbone functionality and its enabling properties mentioned above.

The former question was recently considered by Andrychowicz and Dziembowski [1], who, building on previous suggestions by Aspnes *et al.* [2] of using POWs as an identity-assignment tool and constructions by Fitzi *et al.* [12,19] showing how to morph "graded" consistency into global consistency, showed how to create a consistent PKI using POWs and no other trusted setup, which can then be used to run secure computation protocols (e.g., [23,38]) and realize any cryptographic functionality assuming an honest majority among parties. While this in principle addresses the foundational concerns, it leaves open the questions of doing it in *scalable* way—i.e., with round complexity independent of the number of parties, and in the context of blockchain protocols in particular, designing one that is *provably* secure without a trusted setup.

Our contributions. In this paper we answer the above questions. First, we present a Bitcoin-like protocol that neither assumes a simultaneous start nor the existence of an unpredictable genesis block, and has round complexity

[1] In Bitcoin, solving a proof of work essentially amounts to brute-forcing a hash inequality based on SHA-256.

essentially independent of the number of participants[2]. Effectively, the protocol, starting "from scratch," enables the coexistence of multiple genesis blocks with blockchains stemming from them, eventually enabling the players to converge to a single blockchain. This takes place despite the adversary being allowed (polynomial in the security parameter) pre-computation time. We work in the same model as [21] and we assume a 1/2 bound on adversarial hashing power. We call this protocol the *bootstrapped* (Bitcoin) backbone protocol. A pictorial overview of the protocol's phases, preceded by a period of potential precomputation by the corrupt players, is given in Fig. 1.

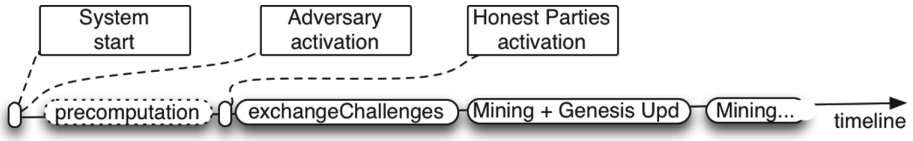


Fig. 1. Timeline and phases of the bootstrapped Bitcoin backbone protocol.

Second, we present applications of our bootstrapped construction, starting with its original one: a distributed ledger, i.e., a public and permanent summary of all transactions that honest parties can agree on as well as add their own, despite the potentially disruptive behavior of parties harnessing less than 1/2 of the hashing power. This entails proving that the ledger's required security properties (Persistence and Liveness—cf. [21]) hold in a genesis block-less setting.

Next, we consider the problem of setting up a PKI in our unauthenticated network setting *from scratch*. As mentioned above, the idea of using POWs as an identity-assignment tool was put forth by Aspnes *et al.* [2]. Here we build on this idea as well as on the "2-for-1 POWs" technique from [21] to use our bootstrapped protocol to assign identities to parties. The assignment relation will possibly assign more than one identity to the same party, while guaranteeing that the majority of them is assigned to honest parties. Such an identity infrastructure/"pseudonymous PKI" has numerous applications, including the bootstrapping of a proof-of-stake protocol [28,30], and the election of honest-majority "subcommittees," which would enable the application of traditional Byzantine fault-tolerant techniques for ledger creation and maintenance (cf. [7]) to permissionless (as opposed to permissioned) networks.

Finally, applying the 2-for-1 POWs technique we can also solve the consensus (aka Byzantine agreement) problem [31,36] probabilistically and from scratch, even if the adversary has almost the same hashing power as the honest parties[3],

[2] "Essentially" because even though there will be a dependency of the round complexity of the setup phase on the probability of computing POWs, which in turn depends on the number of parties, this dependency can be made small enough so as to be considered a constant. See Remark 3.

[3] Thus marking a contrast with the $\frac{2}{3}$ lower bound for consensus on the number of honest parties in the traditional network setting with no setup [6].

and with round complexity independent of the number of parties. Indeed, all our protocols have round complexity linear in the security parameter, and enjoy simultaneous termination. We conclude with an additional modification to the protocol that reduces (by a factor of n) the protocol's communication costs.

Related work. Nakamoto [32] proposed Bitcoin, the first decentralized currency system based on POWs while relaxing the anonymity property of a digital currency to mere pseudonymity. This work was followed by a multitude of other related proposals including Litecoin, Primecoin [29], and Zerocash [4], and further analysis improvements (e.g., [17,18]), to mention a few.

As mentioned above, we work in a model that generalizes the model put forth by Garay *et al.* [21], who abstracted out and formalized the core of the Bitcoin protocol—the Bitcoin *backbone*. As presented in [21], however, the protocol considers as valid any chain that extends the empty chain, which is not going to work in our model. Indeed, if the adversary is allowed polynomial-time pre-computation, he can prepare a very long, private chain; then, by revealing blocks of this chain at the rate that honest players compute new blocks, he can break security. As also mentioned above, to overcome this problem one can assume that at the time honest parties start the computation, they have access to a fresh common random string (a "genesis" block). Then, if we consider as valid only the chains that extend this block, all results proved in [21] follow, since the probability that the adversary can use blocks mined before honest players "woke up" is negligible in the security parameter. In this paper we show how to establish such genesis block directly, and in a number of rounds essentially independent of the number of participants.

To our knowledge, the idea of using POWs to distributedly agree on something (specifically, a PKI) in an unauthenticated setting with no trusted setup was first put forth by Aspnes *et al.* [2], who suggested to use them as an identity-assignment tool as a way to combat *Sybil* attacks [14], and 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 in [2] that a standard authenticated broadcast protocol (specifically, the one by Dolev and Strong [13]) could be run. Such protocols, however, would require that the PKI be *consistent*, details of which are not laid out in [2].

They are in [1], where Andrychowicz and Dziembowski address the more general goal of secure computation in this setting based on POWs, as mentioned earlier; the POWs are used to build a "graded" PKI, where keys have "ranks." The graded PKI is an instance of a "graded agreement," or "partial consistency" problem [12,19,20], where honest parties do not disagree "by much," according to some metric. In [19], Fitzi calls this the *b-set-neighboring* problem ("proxcast" in [12]), with b the number of possible "grades," and shows how to achieve global consistency by running the b-set-neighboring protocol multiple times. In [1], the

fact is used that an unreliable broadcast is available among honest parties to achieve the same—global consistency on a PKI, where the number of identities each party gets is proportional to its hashing power, as suggested in [2].

The protocol in [1], however, suffers from a total running time that depends on the number of parties, because of two factors: (1) the way in which it uses POWs, and (2) the use of the Dolev-Strong authenticated broadcast protocol (run multiple times in parallel based on the graded PKI), which takes a linear number of rounds. Regarding (1), and in more detail, in order to assign exactly one key per party, a low variance POW scheme is used. This implies that the time needed by an honest party to mine a POW is going to be proportional to the ratio of the adversarial hashing power to the hashing power of the weakest honest party. Otherwise, the "rushing" adversary would be able to compute more identities in the additional time she has due to the latency of the communication infrastructure.[4] Regarding (2), we note that potentially an expected-constant-round protocol could be used instead of Dolev-Strong, although the parallel composition of n instances would require more involved techniques [11].

Furthermore, having a PKI allows parties to generate an unpredictable beacon (in the random oracle model), which is then suggested in [1] as a genesis block-generation method for a new cryptocurrency. Yet, no formal treatment of the security of the resulting blockchain protocol is presented, and—as already mentioned—the round complexity of the suggested genesis block generation procedure is linear in the number of participants, both in contrast to our work.

As in [1], Katz *et al.* [26] also consider achieving pseudonymous broadcast and secure computation from POWs ("cryptographic puzzles") and the existence of digital signatures without prior PKI setup, but under the assumption of an existing unpredictable beacon. Finally, Pass *et al.* [35] consider a partially synchronous model of communication where parties are not guaranteed to receive messages at the end of each round but rather after a specified delay Δ (cf. [15]), and show that the backbone protocol can be proven secure in this setting. In principle, our results about the bootstrapped backbone protocol can be extended to their setting as shown in [22].

Organization of the paper. The rest of the paper is organized as follows. In Sect. 2 we describe the network and adversarial model, introduce some basic blockchain notation, and enumerate the various security properties. In Sect. 3 we present the bootstrapped Bitcoin backbone protocol and its analysis. Applications are presented in Sect. 4: a robust public transaction ledger, and PKI generation and consensus without trusted setup and with round complexity independent of the number of parties. Due to space limitations, some of the proofs and further details are presented in the full version of the paper.

[4] On the flip side, the benefit of the approach in [1] is that when all honest parties have the *same* hashing power, a PKI that maps each party to exactly one identity and preserves an honest majority on the keys can be achieved. However, in today's environments where even small devices (e.g., mobile phones, smart watches) have powerful CPUs with different clock frequencies, this assumption is arguably weak.

2 Model and Definitions

We describe our protocols in a model that extends the synchronous communication network model presented in [21] for the analysis of the Bitcoin backbone protocol (which in turn is based on Canetti's formulation of "real world" execution for multi-party cryptographic protocols [8,9]). As in [21], the protocol execution proceeds in rounds with inputs provided by an environment program denoted by $\mathcal{Z}$ to parties that execute the protocol.

Next we provide a high level overview of the model, focusing on the differences that are intrinsic to our setting where the adversary has a precomputation advantage. The adversarial model in the network is actively malicious following the standard cryptographic approach. The adversary is *rushing*, meaning that in any given round it gets to see all honest players's messages before deciding its strategy. Message delivery is provided by a "diffusion" mechanism that is guaranteed to deliver all messages, without however preserving their order and allowing the adversary to arbitrarily inject its own messages. Importantly, the honest parties are not guaranteed to have the same view of the messages delivered in each round, except for the fact that all honest messages from the previous round are delivered. Furthermore, the adversary is allowed to change the source information on every message (i.e., communication is not authenticated). In the protocol description, we will use Diffuse as the message transmission command to capture the "send-to-all" functionality that is available in our setting.[5] Note that, as in [21], an adversarial sender may abuse Diffuse and attempt to confuse honest parties by sending and delivering inconsistent messages to them.

In contrast to [21], where all parties (the honest ones and the ones controlled by the adversary), are activated for the first time in the execution of the protocol in the same round[6], in our model the environment will choose the round at which all the honest parties will become active; the corrupted parties, on the other hand, are activated in the first round. Once honest parties become active they will remain active until the end of the execution. In each round, after the honest parties become active, the environment activates each one by providing input to the party and receives the party's output when it terminates. When activated, 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 a Diffuse message that is guaranteed to be delivered to all parties at the beginning of the next round.

In more detail, we model the execution in the following manner. We employ the parameterized system of ITM's from [9] (2013 version) that is comprised of an initial ITM $\mathcal{Z}$, called the environment, and C, a control function that is specified below. We remark that our control function C is suitably restricted compared to that of [9,10] to take into account restrictions in the order of execution that are relevant to our setting.

[5] In [21] the command name Broadcast is used for this functionality, which we sometimes also will use informally.

[6] After their first-time activation, the environment keeps activating parties in every round (cf. [8]).

The execution is defined with respect to a protocol Π, a set of parties $P_1, \ldots, P_n$ and an adversary $\mathcal{A}$. The adversary is allowed to corrupt parties adaptively up to a number of $t < n$. The protocol Π has access to two resources or "ideal functionalities," the random oracle, and the diffusion channel. Initially, the environment may pass input to either the adversary $\mathcal{A}$ or spawn an instance running the protocol Π which will be restricted to be assigned to the lexicographically smallest honest party (such restrictions are imposed by the control function [9]). After a party P_i is activated, the environment is restricted to activate the lexicographically next honest party, except in the case when no such party is left, in which case the next program to be activated is the adversary $\mathcal{A}$; subsequently, the round-robin execution order between the honest parties will be repeated.

Whenever a party is activated the control function allows for q queries to be made to the random oracle while in the case of an activation of $\mathcal{A}$ a number of $t \cdot q$ queries are allowed where t is the number of corrupted parties. Honest parties are also allowed to annotate their queries to the random oracle for verification purposes, in which case an unlimited amount of queries is permitted (that still counts towards the overall running time of the system execution). Note that the adversary is not permitted to take advantage of this feature of the execution. With foresight, this asymmetry will be necessary, since otherwise it would be trivial for the adversary to break the properties of our protocols by simply "jamming" the incoming communication tape of the honest parties with messages whose verification would deplete their access quota to the random oracle per activation. Furthermore, for each party a single invocation to the diffusion channel is permitted. The diffusion channel maintains the list of messages diffused by each party, and permits the adversary $\mathcal{A}$ to perform a "fetch" operation so that it obtains the messages that were sent. When the adversary $\mathcal{A}$ is activated, the adversary will interact with the diffusion channel, preparing the messages to be delivered to the parties and performing a fetch operation. This write and fetch mode of operation with the communication channel enables the channel to enforce synchrony among the parties running the protocol (cf. [25]).

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. Following the resource-bounded computation model of [9], it holds that the total length of the execution is bounded by a polynomial in the security parameter κ and the length of the auxiliary string $|z|$, provided that the environment is *locally bounded* by a polynomial (cf. Proposition 3 in [9]). Note that the above execution model captures adversarial precomputation since it permits the environment to activate the adversary an arbitrary number of times (bounded by a polynomial in the security parameter κ of course) before the round-robin execution of the honest parties commences.

We note that the above modeling obviates the need for a strict upper bound on the number of messages that may be transmitted by the adversary in each activation (as imposed by [1]). In our setting, honest parties, at the discretion of the environment, will be given sufficient time to process all the messages delivered via the diffusion channel including all messages that are injected by the adversary.

The concatenation of the view of all parties ever activated in the execution, say, $P_1, \ldots, P_n$, is denoted by $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}$. As in [21], 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.

In our correctness and security statements we will be concerned with *properties* of protocols Π running in the above setting (as opposed to simulation-based notions of security). Such properties will be defined as predicates over the random variable $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, z)$ by quantifying over all locally polynomial-bounded adversaries $\mathcal{A}$ and environments $\mathcal{Z}$ (in the sense of [9]). 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 is selected from $\{1, \ldots, \kappa\}$. (Note that, in practice, one may choose k to be much smaller than κ, e.g., $k = 6$).

2.1 Blockchain Notation

Next, we introduce some basic blockchain notation, following [21]. 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

$$(H(ctr, G(s, x)) < D) \wedge (ctr \leq q),$$

where H, G are cryptographic hash functions (e.g., SHA-256) modelled as random oracles. 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 $\text{head}(\mathcal{C}_{\mathsf{new}}) = B$. 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" of 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 $\mathcal{C}_1 \preceq \mathcal{C}_2$.

2.2 Basic Security Properties of the Blockchain

We are going to show that the blockchain data structure built by our protocol satisfies a number of basic properties, as formulated in [21,27]. At a high level, the first property, called *common prefix*, has to do with the existence, as well as persistence in time, of a common prefix of blocks among the chains of honest players [21]. Here we will consider a stronger variant of the property, presented in [27,34], which allows for the black-box proof of application-level properties (such as the *persistence* of transactions entered in a public transaction ledger built on top of the Bitcoin backbone—cf. Sect. 4).

Definition 1 ((Strong) Common Prefix Property). *The* strong common prefix property Q_{cp} *with parameter* $k \in \mathbb{N}$ *states that the chains* $\mathcal{C}_1, \mathcal{C}_2$ *reported by two, not necessarily distinct honest parties* P_1, P_2*, at rounds* r_1, r_2*, with* $r_1 \leq r_2$*, satisfy* $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$*.*

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

Definition 2 (Chain Quality Property). *The* chain quality property Q_{cq} *with parameters* $\mu \in \mathbb{R}$ *and* $k, k_0 \in \mathbb{N}$ *states that for any honest party* P *with chain* $\mathcal{C}$ *in* $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$*, it holds that for any* k *consecutive blocks of* $\mathcal{C}$*, excluding the first* k_0 *blocks, the ratio of adversarial blocks is at most* μ*.*

Further, in the derivations in [21] an important lemma was established relating to the rate at which the chains of honest players were increasing as the Bitcoin backbone protocol was run. This was explicitly considered in [27] as a property under the name *chain growth.* Similarly to the variant of the common prefix property above, this property along with chain quality were shown sufficient for the black-box proof of application-level properties (in this case, transaction ledger *liveness*; see Sect. 4).

Definition 3 (Chain Growth Property). *The chain growth property* Q_{cg} *with parameters* $\tau \in \mathcal{R}$ *(the "chain speed" coefficient) and* $s, r_0 \in \mathbb{N}$ *states that for any round* $r > r_0$*, where honest party* P *has chain* $\mathcal{C}_1$ *at round* r *and chain* $\mathcal{C}_2$ *at round* $r + s$ *in* $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$*, it holds that* $|\mathcal{C}_2| - |\mathcal{C}_1| \geq \tau \cdot s$*.*

3 The Bootstrapped Backbone Protocol

We begin this section by presenting the "bootstrapped" Bitcoin backbone protocol, followed by its security analysis. In a nutshell, the protocol is a generalization of the protocol in [21], which is enhanced in two ways: (1) an initial challenge-exchange phase, in which parties contribute random values, towards the establishment of an unpredictable genesis block, despite the precomputation efforts of corrupt players, and (2) a ranking process and chain-validation predicate that, in addition to its basic function (checking the validity of a chain's content), enables the identification of "fresh" candidate genesis blocks. The ranking process yields

a graded list of genesis blocks and is inspired by the "key ranking" protocol in [1], where it is used to produce a "graded" PKI, as mentioned in Sect. 1.

Before describing the bootstrapped backbone protocol in detail, we highlight its unique features.

- *No trusted setup and individual genesis block mining.* Parties start without any prior coordination and enter an initial challenge-exchange phase, where they will exchange random values that will be used to construct "freshness" proofs for candidate genesis blocks. The parties will run the initial challenge-exchange phase for a small number of rounds, and subsequently will try to mine their own genesis blocks individually. Once they mine or accept a genesis block from the network they will engage in mining further blocks and exchanging blockchains as in Bitcoin's blockchain protocol. On occasion they might switch to a chain with a different genesis block. Nevertheless, as we will show, quite soon they will stabilize in a common prefix and a single genesis block.
- *Freshness of genesis block impacts chains' total weight.* Chains rooted at a genesis block will incorporate its weight in their total valuation. Genesis blocks can be quite "heavy" compared to regular blocks and their total valuation will depend on how fresh they are. Their weight in general might be as much as a linear number of regular blocks in the security parameter. Furthermore, each regular block in a chain accounts for 3 units in terms of the total weight of the chain, something that, as we show, will be crucial to account for differences in terms of weight that are assigned to the same genesis block by different parties running the protocol (cf. Remark 1).
- *Personalized chain selection rule.* Given the co-existence of multiple genesis blocks, a ranking process is incorporated into the chain selection rule that, in addition to its basic function (checking the validity of a chain's content) and picking the longest chain, it now also takes into account the freshness degree of a genesis block from the perspective of each player running the protocol. The ranking process effectively yields a graded list of genesis blocks and is inspired by the "key ranking" protocol in [1], where it is used to produce a "graded" PKI (see further discussion below). The weight value for each genesis block will be thus proportional to its *perceived* "freshness" by each party running the protocol (the fresher the block the higher its weight). It follows that honest players use different chain selection procedures since each predicate is "keyed" with the random coins that were contributed by each player in the challenge-exchange phase (and thus guaranteed to be fresh from the player's perspective). This has the side effect that the same genesis block might be weighed differently by different parties. Despite these differences, we show that eventually all parties accept the same chains as valid and hence will unify their chain selection rule in the course of the protocol.
- *Robustness is achieved after an initial period of protocol stabilization.* All our modifications integrate seamlessly with the Bitcoin backbone protocol [21], and we are able to show that our blockchain protocol is a robust transaction ledger, in the sense of satisfying the properties of persistence and liveness.

Nevertheless, contrary to [21], the properties are satisfied only after an initial period of rounds where persistence is uncertain and liveness might be slower; this is the period where the parties still stabilize the genesis block and they might be more susceptible to attacks. Despite this, a ledger built on top of our blockchain will be available immediately after the challenges exchange phase. Furthermore, once the stabilization period is over the robust transaction ledger behavior is guaranteed with overwhelming probability (in the length of the security parameter).

3.1 Protocol Description

The bootstrapped Bitcoin backbone protocol is executed by an arbitrary number of parties over an unauthenticated network (cf. Sect. 2). For concreteness, we assume that the number of parties running the protocol is n; however, parties need not be aware of this number when they execute the protocol. Communication over the network is achieved by utilizing a send-to-all DIFFUSE functionality that is available to all parties (and may be abused by the adversary in the sense of delivering different messages to different parties). After an initial ("challenge") phase, each party is to maintain a data structure called 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.

The protocol description intentionally avoids 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. In the protocol description, these actions are abstracted by the external functions $V(\cdot), I(\cdot), R(\cdot)$ which are specified by the application that runs "on top" of the backbone protocol.

The protocol is specified as Algorithm 1. At a high level, the protocol first executes a challenge-exchange phase for $l+1$ rounds (l will be determined later), followed by the basic backbone functions, i.e., mining and broadcasting blocks; a crucial difference here with respect to the original backbone protocol is that the chain validation process must also verify candidate genesis blocks, which in turn requires updating the validation function as the protocol proceeds. (This, however, only happens in the next l rounds after the challenge phase.) The protocol's supporting algorithms are specified next.

The challenge-exchange phase. In order to generate an unpredictable genesis block, players first execute a "challenge-exchange" phase, where they broadcast, for a given number of rounds $(l + 1)$, randomly generated challenges that depend on the challenges received in the previous rounds. The property that is assured is that an honest player's k-round challenge, $1 \leq k \leq l$, depends on the $(k - 1)$-round challenges of all honest players. This dependence is made explicit through the random oracle. The code of the challenge-exchange phase is shown in Algorithm 2.

Algorithm 1. The *bootstrapped backbone* protocol, parameterized by the *input contribution function* $I(\cdot)$, the *chain reading function* $R(\cdot)$, and parameter l.

$\mathcal{C} \leftarrow \varepsilon$
$st \leftarrow \varepsilon$
$round \leftarrow 1$ ▷ Global variable *round*
$Gen \leftarrow \emptyset$ ▷ Set of candidate genesis blocks
$Rank \leftarrow \langle \epsilon \rangle$
$(\mathbf{c}, \mathbf{A}, c) \leftarrow \mathsf{exchangeChallenges}(1^\kappa)$
while TRUE **do**
  $k \leftarrow round - l - 2$
  $M_{Gen} \leftarrow \{(\langle s', x', ctr' \rangle, \langle A'_{l+1}, \ldots, A'_{l+1-k} \rangle)\}$ from RECEIVE()
  $M_{Chain} \leftarrow$ chains $\mathcal{C}'$ found in RECEIVE()
  $(Gen, Rank) \leftarrow \mathsf{updateValidate}(\mathbf{c}, \mathbf{A}, M_{Gen}, Gen, Rank)$
  $\tilde{\mathcal{C}} \leftarrow \mathsf{maxvalid}(\mathcal{C}, M_{Chain}, Gen, Rank)$
  $\langle st, x \rangle \leftarrow I(st, \tilde{\mathcal{C}}, round, \text{INPUT}(), \text{RECEIVE}())$
  $\mathcal{C}_{\mathsf{new}} \leftarrow \mathsf{pow}(x, \tilde{\mathcal{C}}, c)$
  if $\mathcal{C} \neq \mathcal{C}_{\mathsf{new}}$ **then**
    if $\mathcal{C} = \epsilon$ **then** ▷ New genesis block has been produced
      DIFFUSE($(\mathcal{C}_{\mathsf{new}}, \langle A_{l+1}, \ldots, A_{l+1-(k+1)} \rangle)$)
    $\mathcal{C} \leftarrow \mathcal{C}_{\mathsf{new}}$
    DIFFUSE($\mathcal{C}$)
  $round \leftarrow round + 1$
  if INPUT() contains READ **then**
    write $R(\mathbf{x}_{\mathcal{C}})$ to OUTPUT()

Validation predicate update. In the original backbone protocol [21], the chain validation function (called validate—see below) performs a validation of the structural properties of a given chain $\mathcal{C}$, and remains unchanged throughout the protocol. In our case, however, where there is no initial fresh common random string, the function plays the additional role of checking for valid genesis blocks, and players have to update their validation predicate as the protocol advances (for the first l rounds after the challenge phase).

Indeed, using the challenges distributed in the challenge-exchange phase of the protocol, players are able to identify fresh candidate genesis blocks that have been shared during that phase and are accompanied by a valid proof. In addition, the valid genesis blocks are ranked with a negative dependence on the round they were received. In order to help other players to also identify the same genesis blocks, players broadcast the valid genesis blocks they have accepted together with the additional information needed by the other players for verification. The validation predicate update function is shown in Algorithm 3. Recall that *Gen* is the set of candidate genesis blocks.

Chain validation. A chain is considered valid if in addition to the checks performed by the basic backbone protocol regarding the chain's structural

Algorithm 2. The *challenge-exchange* function. Note that variable *round* is global, and originally set to 1.

function exchangeChallenges(1^κ)
  $c_1 \xleftarrow{R} \{0,1\}^\kappa$
  DIFFUSE(c_1)
  $round \leftarrow round + 1$
  while $round \leq l + 1$ **do**
    $A_{round} \leftarrow \kappa$-bit messages found in RECEIVE()
    $r_{round} \xleftarrow{R} \{0,1\}^\kappa$
    $A_{round} \leftarrow A_{round} || r_{round}$
    $c_{round} \leftarrow H(A_{round})$ ▷ Compute challenge
    DIFFUSE(c_{round})
    $round \leftarrow round + 1$
  return $(\langle c_1, \ldots c_l \rangle, \langle A_2, \ldots A_{l+1} \rangle, c_{l+1})$

Algorithm 3. The *validation predicate update* function.

function updateValidate($\mathbf{c}, \mathbf{A}, M_{Gen}, Gen, Rank$)
  $k \leftarrow round - l - 2$
  if $k \geq l$ **then**
    return $Gen, Rank$ ▷ No updates after round $2l + 2$
  for each $(\langle s', x', ctr' \rangle, \langle A'_{l+1}, \ldots, A'_{l+1-k} \rangle)$ in M_{Gen} **do**
    if validblock$_q^D(\langle s, x, ctr \rangle) \wedge \langle s, x, ctr \rangle \notin Gen$ **then**
      $flag \leftarrow (H(A'_{l+1}) = s) \wedge (c_{l-k} \in A'_{l+1-k})$
      for $i = l + 1 - k$ to l **do**
        if $H(A'_i) \notin A'_{i+1}$ **then**
          $flag \leftarrow$ FALSE
      if $flag =$ TRUE **then**
        $Gen \leftarrow Gen \cup \langle s, x, ctr \rangle$
        $Rank[\langle s, x, ctr \rangle] \leftarrow l - k$
        DIFFUSE($\langle s, x, ctr \rangle, \langle A'_{l+1}, \ldots, A'_{l+1-k}, A_{l-k} \rangle$) ▷ Augment A' sequence with own A value.
  return $Gen, Rank$

properties, its genesis block is in the *Gen* list, which is updated by the updateValidate function (Algorithm 3). The chain validation function is shown in Algorithm 4.

Chain selection. The objective of the next algorithm in Algorithm 1, called maxvalid, is to find the "best possible" chain. The accepted genesis blocks have different *weights* depending on when a player received them. It is possible that the same genesis block is received by honest players in two different rounds

(as we show later, those rounds have to be consecutive). In order to take into account the "slack" introduced by the different views honest players may have regarding the same block, as well as the different weights different blocks may have, we let the *weight of a chain* $\mathcal{C}$ be equal to the *weight of its genesis block plus three times its length minus one*. The chain selection function is shown in Algorithm 5.

Algorithm 4. The *chain validation predicate*, parameterized by q, D, the hash functions $G(\cdot), H(\cdot)$, and the *content validation predicate* $V(\cdot)$. The input is $\mathcal{C}$.

function validate($\mathcal{C}$, Gen)
  $b \leftarrow V(\mathbf{x}_{\mathcal{C}}) \wedge (C \neq \varepsilon) \wedge (\text{tail}(\mathcal{C}) \in Gen)$
  if $b = \text{True}$ **then**
    $\langle s, x, ctr \rangle \leftarrow \text{head}(\mathcal{C})$
    $s' \leftarrow H(ctr, G(s, x))$
    repeat
      $\langle s, x, ctr \rangle \leftarrow \text{head}(\mathcal{C})$
      if validblock$_q^D(\langle s, x, ctr \rangle) \wedge (H(ctr, G(s, x)) = s')$ **then**
        $(s', \mathcal{C}) \leftarrow (s, \mathcal{C}^{\lceil 1})$ ▷ Retain hash value and remove the head from $\mathcal{C}$
      else
        $b \leftarrow \text{False}$
    until $(\mathcal{C} = \varepsilon) \vee (b = \text{False})$
  return b

Algorithm 5. The function that finds the "best" chain. The input is a set of chains and the list of genesis blocks.

function maxvalid($\mathcal{C}_1, \ldots, \mathcal{C}_k$, Gen)
  $temp \leftarrow \varepsilon$
  $maxweight \leftarrow 0$
  for $i = 1$ to k **do**
    if validate($\mathcal{C}_i, Gen$) **then**
      $weight \leftarrow Rank(\text{tail}(\mathcal{C}_i)) + 3(|\mathcal{C}_i| - 1)$
      if $maxweight < weight$ **then**
        $maxweight \leftarrow weight$
        $temp \leftarrow \mathcal{C}_i$
  return $temp$

The proof-of-work function. Finally, we need to modify the proof-of-work function in [21], so that when a genesis block is mined, the challenge computed in the last round of the challenge-exchange phase will be included in the block. This, in addition to the proof of genesis information sent in the backbone protocol,

is required so that other honest players accept this block as valid and rank it accordingly. The code is presented in Algorithm 6.

Algorithm 6. The *proof of work* function, parameterized by q, D and hash functions $H(\cdot), G(\cdot)$. The input is $(x, \mathcal{C}, c)$.

```
1:  function pow(x, C, c)
2:      if C = ε then
3:          s ← c                                   ▷ c is required to prove freshness
4:      else
5:          ⟨s', x', ctr'⟩ ← head(C)
6:          s ← H(ctr', G(s', x'))
7:      ctr ← 1
8:      B ← ε
9:      h ← G(s, x)
10:     while (ctr ≤ q) do
11:         if (H(ctr, h) < D) then                 ▷ Proof of work found
12:             B ← ⟨s, x, ctr⟩
13:             break
14:         ctr ← ctr + 1
15:     return CB                                   ▷ Extend chain
```

Figure 2 presents the overall structure (phases and corresponding rounds) of the bootstrapped backbone protocol. Next, we turn to its analysis.

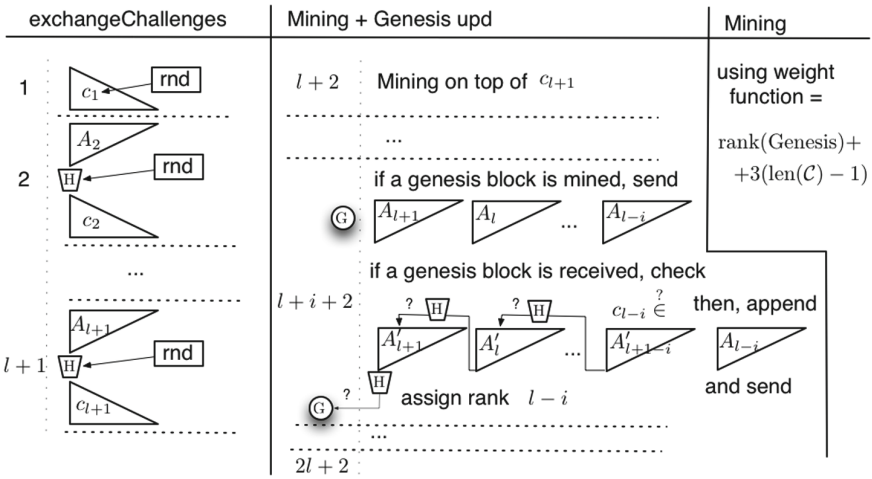
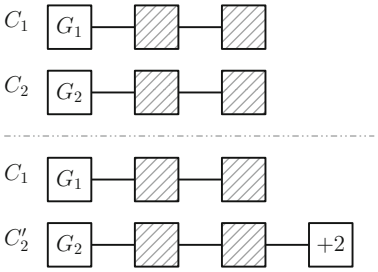


Fig. 2. The different phases of the bootstrapped backbone protocol.

Remark 1. To understand some of our design choices we briefly give some examples of simpler protocols that don't work. For the first example, assume that we only have one round of challenge exchange i.e. l equal to 1. With some non-negligible probability, the adversary can send one block to half of the honest players and another block to the other half. By splitting the honest players in two groups such that no one in the first group will choose the chain of the second and vice versa, agreement becomes impossible. Moreover, l must be large enough so that at least one honest party computes a genesis block with overwhelming probability. Otherwise the adversary can choose to remain silent and no genesis block will be mined with non-negligible probability.

For the second example assume that blocks weigh less than 3 units, as in the original protocol. Also, assume that somehow the problem of the first example was avoided and honest parties only adopted chains with genesis blocks that everyone had in their genesis block list. In this case, uniquely successful rounds would not imply agreement on a single chain (see Fig. 3), as the adversary would have been able to take advantage of the different views that honest players have regarding the weight of genesis blocks. However, if we set the block weight to 3, this event becomes impossible and makes the analysis a lot easier.



	P_1	P_2
G_1	6	5
G_2	4	5
C_1	10	9
C_2	8	9
C'_2	10	11

Fig. 3. An example where blocks weigh 2 units. In the table the weights of the respective chains are depicted. Initially player P_1 has adopted chain $\mathcal{C}_1$ and player P_2 chain $\mathcal{C}_2$. Then a uniquely successful round happens and $\mathcal{C}_2$ is extended to $\mathcal{C}'_2$. Notice that, P_1 will not adopt $\mathcal{C}'_2$ since it has the same weight as C_1. If the new block weighted 3 units, all players would have adopted chain $\mathcal{C}'_2$.

3.2 Analysis of the Bootstrapped Backbone Protocol

First, some additional definitions that will become handy in the analysis. We saw in the previous section that genesis blocks are assigned weights, and, further, that a single genesis block may have different weights for different parties depending on when they received it. We extend this notion to chains of blocks.

Definition 4. *Let $w_P(B)$ be the weight that P assigned to genesis block B. We define the weight of a chain $\mathcal{C}$ with genesis block B (with respect to party P) to be:*

$$w_P(\mathcal{C}) = w_P(B) + 3(|\mathcal{C}| - 1).$$

If block B was not received by P until round $2l+1$, or if $\mathcal{C} = \epsilon$, then $w_P(\mathcal{C}) = -1$.

In [21], all parties assign the same weight to the same chain, i.e., the length of the chain; thus, for all parties P_i, P_j we have that $w_{P_i}(\mathcal{C}) = w_{P_j}(\mathcal{C})$. In contrast, in our case the genesis block of each chain may have different weight for different parties, akin to some bounded amount of "noise" that is party-dependent being added to the chain weights. We are going to show that if the amount of noise is at most 1, then by letting each new block weigh 3 units our protocol satisfies the chain growth, common prefix and chain quality properties.

Definition 5. *Regarding chains and their weight:*

- *Define $h_{\mathcal{C}} = \max_P\{w_P(\mathcal{C})\}$ and $\ell_{\mathcal{C}} = \min_P\{w_P(\mathcal{C})\}$.*
- *Let $\mathcal{C}(B)$ denote the truncation of chain $\mathcal{C}$ after its block B.*
- *For a block B of a chain $\mathcal{C}$, define $h_{\mathcal{C}}(B) = h_{\mathcal{C}(B)}$ and similarly for $\ell_{\mathcal{C}}(B)$. (Sometimes we will abuse notation and write $\ell(B)$ instead of $\ell_{\mathcal{C}}(B)$. As long as no collision happens $\ell(B)$ is well defined. The same holds for $h(B)$.)*
- *For chains $\mathcal{C}_1$ and $\mathcal{C}_2$, define $\mathcal{C}_1 \cap \mathcal{C}_2$ to be the chain formed by their common prefix.*

The following are important concepts introduced in [21], which we are also going to use in our analysis:

Definition 6. *A round is called:*

- successful *if at least one honest party computes a solution;*
- uniquely successful *if exactly one honest party computes a solution.*

Definition 7. *In an execution blocks are called:*

- honest*, if mined by an honest party,*
- adversarial*, if mined by the adversary, and*
- u.s. blocks*, if mined in a uniquely successful round by an honest player.*

Recall that our model is "flat" in terms of computational power in the sense that all honest parties are assumed to have the same computational power while the adversary has computational power proportional to the number of players that it controls. The total number of parties is n and the adversary is assumed to control up to t of them (honest parties do not know any of these parameters). Obtaining a new block is achieved by finding a hash value that is smaller than the difficulty parameter D. Thus, the success probability that a single hash query produces a solution is $p = \frac{D}{2^\kappa}$, where κ is the length of the hash. The total hashing power of the honest players is $\alpha = pq(n-t)$, the hashing power of the adversary is $\beta = pqt$, and the total hashing power is $f = \alpha + \beta$. Moreover, in [21], a lower bound on the probability that a round is uniquely successful was established; denoted by γ and equal to $\alpha - \alpha^2$. Notice that γ is also a bound for the probability of a round being just successful.

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

computed a POW at round j, and let $Y_j = 1$ iff j was a *uniquely successful round*, i.e., exactly one honest party computed a POW at round j. With respect to a set of rounds S, let $Z(S)$ denote the number of POWs obtained by the adversary during the rounds in S (i.e., in $qt|S|$ queries). Also, let $X(S) = \sum_{j \in S} X_j$ and define $Y(S)$ similarly. Note that $\gamma|S| \leq \mathbb{E}[Y(S)] \leq \mathbb{E}[X(S)] \leq \alpha|S|$ and $\mathbb{E}[Z(S)] = \beta|S|$.

Lemma 1. *If $|S| = k$ and $\gamma \geq (1+\delta)\beta$ for some $\delta \in (0,1)$, then*

$$\Pr[Y(S) > (1 + \frac{5\delta}{9})Z(S)] > 1 - e^{-\Omega(\delta^2 k)}.$$

Proof. By the Chernoff bound we have that:

$$\Pr[Y(S) \leq (1 - \frac{\delta}{8})\mathbb{E}[Y(S)]] \leq e^{-\frac{\delta^2 \gamma k}{128}} \text{ and } Pr[Z(S) \geq (1 - \frac{\delta}{9})\mathbb{E}[Z(S)] \leq e^{-\frac{\delta^2 \beta k}{243}}.$$

Suppose none of the above events happens. Then, from the union bound, we get that with probability $1 - e^{-(2\min(\frac{\beta}{243}, \frac{\gamma}{128})\delta^2 k - \ln(2))}$ it holds that

$$Y(S) > (1 - \frac{\delta}{8})\gamma k \geq (1 - \frac{\delta}{8})(1+\delta)\beta k \geq (1 + \frac{5\delta}{9})(1 + \frac{\delta}{9})\beta k > (1 + \frac{5\delta}{9})Z(S).$$

□

Remark 2. For ease of exposition, in our analysis we will assume that there are no collisions; that is, for any two different queries to the random oracle, always a different response is returned. This would generally be a problem since for example it would break independence of X_i, X_j, for $i \neq j$, and we would not be able to apply the Chernoff bound in the previous lemma. However, since the probability of a collision happening, as well as all other events we consider, is at most $e^{-\Omega(\kappa)}$, we can always use the union bound to include the event of no collision occurring to our other assumptions. In addition, we assume that no two queries to the oracle are the same, as formalized by the Input Entropy condition in [21].

Properties of the genesis block generation process. We now establish a number of properties of the genesis block generation process.

Lemma 2 (Graded Consistency). *If any honest party P_i accepts genesis block B with rank $w_{P_i}(B) > 1$, then all honest parties accept B with rank at least $w_{P_i}(B) - 1$.*

Proof. Let $w_{P_i}(B) = k > 1$. Since P_i accepted B with rank k at some round r, he must have received a message of the form $(B, E_{l+1}, \ldots, E_{k+1})$, where

- B is a valid block that contains $H(E_{l+1})$;
- E_{k+1} contains c_k and for $k+2 < j \leq l+1$, E_j contains $H(E_{j-1})$; and
- c_k is the challenge computed by P_i at round k.

Since $k > 0$, according to Algorithm 3, P_i is going to broadcast $(B, E_{l+1}, \ldots, E_{k+1}, A_k)$, where $H(A_k) = c_k$ is contained in E_{k+1} and A_k contains all the messages received by P_i at round k. All honest-party challenges of round $k-1$ were received in this round; therefore, all honest parties have accepted or will accept block B by the next round and the lemma follows. □

Lemma 3 (Validity). *Genesis blocks computed by honest parties before round $2l+2$, will be accepted by all honest parties in the next round.*

Proof. Suppose honest party P_i mined genesis block B at round m. According to Algorithm 1, B contains the challenge he has computed at the last round of the challenge-exchange phase. In addition, when the party broadcasts it, it includes the message sets $A_{l+1}, \ldots, A_r$, where A_j contains the messages received by P_i at round j and $r = 2l + 2 - m$. Since P_i is honest, the following hold:

- B is a valid block that contains $H(A_{l+1})$;
- for $r + 1 < j \leq l + 1$, A_j contains $H(A_{j-1})$;
- if c_r is the challenge sent by some honest party at round r, then c_r is contained in A_{r+1}; and
- all honest parties are going to receive the message.

Thus, all honest parties are going to accept B at round $m + 1$ and the lemma follows. □

Lemma 4 (Freshness). *Let $r \leq l + 2$. Every block computed before round r cannot be part of some chain with genesis block B, where $w_P(B) \geq r - 1$ for some honest party P, with overwhelming probability in the security parameter κ.*

Weak chain growth. We now turn our attention to the weight of chains and prove a *weak* chain-growth property. In the original Bitcoin backbone protocol [21], it was proved that chains grow at least at the rate of successful rounds, independently of the adversary's behavior. Here, at least initially, the chains of honest parties grow in a "weak" manner, in the sense that the adversary is able to slow down this growth by using his own blocks. Later on, we will show that after some specific round our protocol also achieves optimal chain growth.

Lemma 5. *Let round r such that $l + 2 \leq r < 2l + 2$, and suppose that at round r an honest party, say, P_1 has a chain $\mathcal{C}$ such that $w_{P_1}(\mathcal{C}) = d$. Then, by round s, where $r \leq s < 2l + 2$, every honest party P will have received a chain $\mathcal{C}'$ of weight at least $w_P(\mathcal{C}') = d - 2 + 3\sum_{i=r}^{s-1} Y_i - \sum_{i=r}^{s-1} Z_i$.*

Proof (*Sketch*). Note that every time a uniquely successful round happens, the minimum weight over all parties' chains will increase by 2. Moreover, if the adversary has not diffused any block in the same round the minimum weight increases by 3. By applying this result iteratively, the lemma follows. We refer the reader to the full version of the paper for the full proof. □

Universal chain validity. A novelty of our construction is that the same genesis block may have different weight for different parties. Unfortunately, it could be the case that due to the adversary's influence, a genesis block is valid for one party but invalid for another. This could lead to disagreement, in the sense that some honest parties may adopt a chain that others don't because it is not valid for them. We will show that with overwhelming probability such an event cannot occur for our protocol; as such, chain validity is a "universal" property; if some honest party accepts a chain $\mathcal{C}$ as valid, then $\mathcal{C}$ will also be valid for all other parties.

Notice, that in order to prove the following lemma we need l to be greater than a value that depends on $1/\gamma$, i.e. the expected time it takes for honest parties to mine a block, and the security parameter κ (see also Remark 1). Intuitively l should be large enough so that (i) honest parties mine at least one block at this time interval, and (ii) any adversarial chain that is based on a genesis block broadcast at the end of the bootstraping phase will never be adopted by honest parties (because such genesis block will have too small weight in comparison).

Lemma 6. *Suppose that for some* $\delta \in (0,1), 3(1+\delta)f < 1, l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, *and* $\gamma \geq (1+\delta)\beta$, *and that at round* r *an honest party* P *has chain* $\mathcal{C}$. *Then* $\mathcal{C}$ *will also be valid for all other parties from this round on with probability* $1-e^{-\Omega(\delta^2 k)}$.

The complete version of the weak chain growth lemma follows from the argument we've made above.

Corollary 1. *Suppose that for some* $\delta \in (0,1)$, $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, *and* $\gamma \geq (1+\delta)\beta$. *Let round* r *such that* $r \geq l+2$, *and suppose that at round* r *an honest party, say,* P_1 *has a chain* $\mathcal{C}$ *such that* $w_{P_1}(\mathcal{C}) = d$. *Then, by round* s, *where* $r \leq s$, *every honest party* P *will have received a chain* $\mathcal{C}'$ *of weight at least* $w_P(\mathcal{C}') = d - 2 + 3\sum_{i=r}^{s-1} Y_i - \sum_{i=r}^{s-1} Z_i$ *with probability at most* $1 - e^{-\Omega(\delta^2 k)}$.

Remark 3. Note further that the dependency of γ on n does not undermine the scalability of the round complexity of our protocol. This claim is argued on the basis that the difficulty level D can be set proportional to $1/n$, so that γ can be treated as a constant and then l is in essence independent of n (note that both parameters would be polynomials in κ).

A bound on adversarially precomputed blocks. The honest parties begin mining right after the challenge-exchange phase. Note that it does not help the adversary to precompute blocks before the challenge-exchange phase, except for the small probability of the event that some of his blocks happen to extend future blocks. We have shown that the adversary cannot create a private chain that honest parties will adopt if he starts mining at the first round of the challenge-exchange phase. It is though possible to start mining *after* the first round in order to gain some advantage over the honest parties. The following lemma provides a bound on the number of blocks mined during the challenge-exchange phase with sufficient weight so that they can be later used by the adversary.

Lemma 7 (Precomputed blocks). *Assume* $3(1+\delta)f < 1$ *and* $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, *for some* $\delta \in (0,1)$. *Let* R *be the set that contains any adversarial block* B *mined before round* $l+2$, *where* $h(B) > l - 1 - (1-\delta)\delta^2 k$. *Then* $\Pr[|R| > \frac{5\delta}{9}k\beta] \leq e^{-\Omega(\delta^4 k)}$.

Proof (Sketch). We first show that the adversary cannot take advantage of blocks which belong to chains whose genesis block was computed early on in the challenge exchange phase. Hence, with overwhelming probability she can only use blocks computed near the end of the challenge exchange phase; remember that the weight of a genesis block is small if it is mined early in the challenge exchange phase. By applying appropriate Chernoff bounds the result follows. We refer the reader to the full version of the paper for the full proof. □

We are now ready to prove the security properties listed in Sect. 2.2.

Common Prefix. Every time a uniquely successful round happens all honest players converge to one chain, unless the adversary broadcasts some new block. This turns out to be a very important fact and a consequence of it is described in the next lemma.

Lemma 8. *Suppose block* B *in chain* $\mathcal{C}$ *is a u.s. block and consider a chain* $\mathcal{C}'$ *such that* $B \notin \mathcal{C}'$. *If* $\ell_{C'} \geq \ell_C(B) - 1$ *then there exists a unique adversarial block* B' *such that* $\ell_{\mathcal{C}'}(B') \in [\ell_{\mathcal{C}}(B) - 1, \ell_{\mathcal{C}}(B) + 1]$. *Moreover, if* B *is not a genesis block, then* B' *will also not be a genesis block.*

Proof. Assume block B was mined at some round r. If B is not a genesis block, then for any honest block B'' mined before round r it should hold that $\ell(B'') \leq \ell(B) - 2$. Otherwise, at round r no honest party would choose the parent of B to mine new blocks. If B is a genesis block, then no other honest party has mined a block in some previous round. On the other hand, for any honest block B'' mined after round r it must hold that $\ell(B'') \geq \ell(B) - 1 + 3 = \ell(B) + 2$, since honest parties will only extend chains of length at least $\ell(B) - 1$ after this round. Thus, if a block with weight in the given interval exists, it must be adversarial.

For the sake of contradiction, suppose B is not a genesis block while B' is a genesis block and let B'' be the parent of B. Then $h_{\mathcal{C}}(B'') < \ell_{\mathcal{C}'}(B')$ since $h_{\mathcal{C}}(B'') \leq \ell_{\mathcal{C}}(B) - 2$. This implies than every honest party received B' before block B''. But then, no honest party would mine on the parent of B, because he would have lower weight than B', which leads to a contradiction. Hence, the lemma follows. □

We use Lemma 8 in order to show that the existence of a fork implies that the adversary has mined blocks proportional in number to the time the fork started.

Theorem 1. *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0,1)$. *Let* S *be the set of the chains of the honest parties from round* $2l+2$ *and onwards of the bootstrapped backbone protocol. Then the probability that* S *does not satisfy the strong common-prefix property with parameter* k *is at most* $e^{-\Omega(\delta^4 k)}$.

Proof (Sketch). We are going to use Lemma 8 to match u.s. blocks to adversarial ones. Function ℓ will help us show that the matched blocks are distinct; every pair of matched blocks is very close with respect to the ℓ function, while pairs of u.s. blocks can be very far under specific conditions. Initially, we construct such a matching whenever we have a fork between two chains $\mathcal{C}_1$, $\mathcal{C}_2$, either by matching adversarial blocks alternatively to each of the chains or by matching consecutive u.s. blocks on the same chain to consecutive adversarial blocks in the other chain.

Using this result, we prove that if a "deep" enough fork exists, the adversary must have mined more blocks than his hashing power allows, which leads to a contradiction. In more detail, the initial fork implies the existence of some honestly mined block B that is part of the common prefix of the two chains. Starting from B we construct a matching of all u.s. blocks mined after it, by picking "deeper" and "deeper" forks and repeatedly applying the matching procedure. Using the upper bound on precomputed blocks established in Lemma 7, we can show that the adversary is able to mine a sufficient number of blocks only with negligible probability. Hence, the theorem follows. □

Chain Growth. We proved that after round $2l + 1$ the strong common-prefix property is satisfied. This implies that all players share a common genesis block after this round. The next lemma shows that this is sufficient in order to get chain growth at the same level as in the original Backbone protocol.

Lemma 9. *Suppose that at round r an honest party P_1 has a chain $\mathcal{C}$ of weight $w_{P_1}(\mathcal{C}) = d$ and all honest parties after round $r - 1$ adopt chains that share the same genesis block B. Then, by round $s \geq r$, every honest party P will have received a chain $\mathcal{C}'$ of weight at least $w_P(\mathcal{C}') = d - 1 + 3\sum_{i=r}^{s-1} X_i$.*

Proof. Since all parties adopt chains with the same genesis block after round $r - 1$, and P_1 has adopted a chain $\mathcal{C}$ of weight d, there are two cases: either (1) $\ell_{\mathcal{C}} = d - 1$ and any chain that honest parties adopt after round $r - 1$ has a weight that is congruent to d or $d - 1$ modulo 3, or (2) $\ell_{\mathcal{C}} = d$ and the weight is congruent to d or $d + 1$ modulo 3. This observation is implied from the fact that each extra block adds 3 units of weight to the chain and B can only have two different weights under the views of honest parties.

It is sufficient to study only one of the two cases so w.l.o.g. suppose that the weight of the chains is congruent to d or $d - 1$ modulo 3. The proof is by induction on $s - r \geq 0$. For the basis ($s = r$), observe that if at round r P_1 has a chain $\mathcal{C}$ of weight $w_{P_1}(\mathcal{C}) = d$, then he broadcast $\mathcal{C}$ at an earlier round (than r). It follows that every honest party P will receive $\mathcal{C}$ by round r and $w_P(\mathcal{C}) \geq d-1$.

For the inductive step, note that by the inductive hypothesis every honest party P has received a chain $\mathcal{C}'$ of weight at least $w_P(\mathcal{C}') = d' = d-1+3\sum_{i=r}^{s-2} X_i$ by round $s - 1$. When $X_{s-1} = 0$ the statement follows directly, so assume $X_{s-1} = 1$. Observe that every honest party queried the oracle with a chain of weight at least d' at round $s - 1$. It follows that every honest party P successful at round $s - 1$ broadcast a chain $\mathcal{C}'$ of weight at least $w_P(\mathcal{C}') = d' + 3$. For every

other party P' it holds that $w_{P'}(\mathcal{C}') \geq d' + 2 \geq d - 1 + 3\sum_{i=r}^{s-1} X_i - 1$. However, no chain that an honest party adopts can have length $d' + 2$, because $d' + 2$ is congruent to $d - 2$ modulo 3. Thus all honest parties adopt chains that have length at least $d' + 3$ and the lemma follows. □

It can be easily shown that Lemma 9 implies the chain growth property after round $2l + 1$.

Theorem 2. *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0,1)$. *The bootstrapped Bitcoin protocol satisfies the chain growth property for* $r_0 = 2l+2$ *with speed coefficient* $(1-\delta)\gamma$ *and probability at least* $1-e^{-\Omega(\delta^4 s)}$.

Chain Quality. We first observe a consequence of Theorem 1.

Lemma 10. *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0,1)$. *From round* $2l+2$ *and onwards of the bootstrapped backbone protocol, the probability that the adversary has a chain which is more than* k *blocks longer than the chain of some honest party is at most* $e^{-\Omega(\delta^4 k)}$.

Proof. Given any execution and an adversary that at a round r has a chain $\mathcal{C}$ which is k blocks longer than the chain $\mathcal{C}'$ of an honest party P, we can define an adversary such that at round $r+1$ the common-prefix property does not hold for parameter k. The adversary simply sends $\mathcal{C}$ to $P' \neq P$ at round r. □

Theorem 3. *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0, 1/2)$. *Suppose* $\mathcal{C}$ *belongs to an honest party and consider any* k *consecutive blocks of* $\mathcal{C}$ *computed after round* $2l+2$ *of the bootstrapped backbone protocol. The probability that the adversary has contributed more than* $(1+\frac{\delta}{2})\frac{\beta}{\gamma} \cdot k \leq (1-\frac{\delta}{3})k$ *of these blocks is less than* $e^{-\Omega(\delta^5 k)}$.

Corollary 2. *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0, 1/2)$. *The bootstrapped Bitcoin protocol satisfies the chain-quality property with parameters* $\mu = (1+\frac{\delta}{2})\frac{\beta}{\gamma}$, $k_0 = 2f(1+\delta)(l+1)$, *and* k, *with probability at least* $1 - e^{\Omega(\delta^5 k)}$.

Proof. Note that the next two events occur with probability at least $1 - e^{\Omega(\delta^2 l)}$, for any $\delta \in (0,1)$. The honest parties in the first $l+1$ rounds have computed at most $\alpha(1+\delta)(l+1)$ blocks. The adversary, who might have been mining also during the challenges phase, has computed at most $2\beta(1+\delta)(l+1)$. The statement then follows from Theorem 3, since $\alpha(1+\delta)(l+1) + 2\beta(1+\delta)(l+1) < 2f(1+\delta)(l+1)$. □

4 Applications of the Bootstrapped Backbone Protocol

In this section we present applications of our construction, starting with its primary/original one: a distributed ledger, i.e., a public and permanent summary

of all transactions that honest parties can agree on as well as add their own, despite the potentially disruptive behavior of parties harnessing less than 1/2 of the hashing power. This entails proving that the ledger's required security properties (Persistence and Liveness—cf. [21]) hold in a genesis block-less setting.

Next, we consider the problem of setting up a PKI in our unauthenticated network setting *from scratch*, i.e., without any trusted setup. As mentioned in Sect. 1, the idea of using POWs as an identity-assignment tool was put forth by Aspnes *et al.* [2]. Here we build on this idea as well as on the "2-for-1 POWs" technique from [21] to use our bootstrapped protocol to assign identities to parties. The assignment relation will possibly assign more than one identities to the same party, while guaranteeing that the majority of them is assigned to honest parties.

Finally, applying the 2-for-1 POWs technique we can also solve the consensus (aka Byzantine agreement) problem [31,36] without any trusted setup, even if the adversary has almost the same hashing power as the honest parties, and in a number of rounds independent of the number of parties. Indeed, all our protocols have round complexity linear in the security parameter, and enjoy simultaneous termination.

Compared to other works, most notably [1], our approach is different in the order in which it sets up a "bulletin board" and assigns identities to parties. We choose to first establish the former—i.e., the ledger—and then assign the identities; in contrast, in [1] identities are established first in a graded manner, and then using that infrastructure the parties can implement a broadcast channel.

We now turn to the applications in detail.

Robust public transaction ledger. 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*. Ledgers correspond to chains in the backbone protocol. In the protocol execution there also exists an oracle Txgen that generates valid transactions. Note, that it is possible for the adversary to create two transactions that are conflicting; valid ledgers must not contain conflicting transaction. We will assume that the oracle is unambiguous, i.e., that the adversary cannot create transactions that come in 'conflict' with the transactions generated by the oracle. A transaction is called *neutral* if there does not exist any transactions that comes in conflict with it.

In order to turn the backbone protocol into a protocol realizing a public transaction ledger suitable definitions were given for functions $V(\cdot), R(\cdot), I(\cdot)$ in [21]. Namely, $V(\langle x_1, \ldots, x_m)$ is true if its input is a valid ledger. Function $R(\mathcal{C})$ returns the contents of the chain if they constitute a valid ledger, otherwise it is undefined. Finally, $I(st, \mathcal{C}, round, \mathtt{INPUT}(), \mathtt{RECEIVE}())$ returns the largest subsequence of transactions in the input and receive tapes that constitute a valid ledger, with respect to the contents of the chain the party already has, together with a randomly generated neutral transaction. We denote the instantiation of our protocol with these functions by $\Pi_{\mathsf{PL}}^{\mathtt{Boot}}$. For more details we refer to [21].

Definition 8. *A protocol* Π *implements a* robust public transaction ledger *in the* q*-bounded synchronous setting without trusted setup if there is a round* r_0 *so that the following two properties are satisfied:*

- Persistence: *Parameterized by* $k \in \mathbb{N}$ *(the "depth" parameter), if in a certain round after* r_0 *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 after round* r_0*, then there exists an honest party who will report this transaction at a block more than* k *blocks from the end of the ledger.*

Chain quality, chain growth and the strong common prefix property were shown in [27] to be sufficient to implement such a ledger[7] in a black-box manner. Our protocol satisfies all these properties after a specific condition is met. Chain quality holds after the $2f(1+\delta)(l+1)$ block in the chain of any player, as Corollary 2 dictates, and common prefix and chain growth hold after round $2l+2$, according to Theorem 1. Finally, due to chain growth, after at most $(2(1+\delta)(1-\delta)f/\gamma+2)(l+1) \leq 14(l+1)$ rounds all necessary conditions will have been met with overwhelming probability.

Lemma 11 (Persistence). *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0, 1/2)$. *Then for all* $k \in \mathbb{N}$ *protocol* $\Pi_{\mathsf{PL}}^{\mathsf{Boot}}$ *satisfies Persistence after round* $2l+2$ *with probability* $1 - e^{-\Omega(\delta^5 k)}$*, where* k *is the depth parameter.*

Lemma 12 (Liveness). *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0, 1/2)$. *Further, assume oracle* Txgen *is unambiguous. Then for all* $k \in \mathbb{N}$ *protocol* $\Pi_{\mathsf{PL}}^{\mathsf{Boot}}$ *satisfies Liveness after round* $14(l+1)$ *with wait time* $u = \frac{3}{(1-\delta)\gamma} \cdot \max(k, \frac{1}{1-(1+\frac{\delta}{2})\frac{\beta}{\gamma}})$ *rounds and depth parameter* k *with probability at least* $1 - e^{-\Omega(\delta^5 k)}$.

Corollary 3. *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0, 1/2)$. *Then protocol* $\Pi_{\mathsf{PL}}^{\mathsf{Boot}}$ *implements a robust transaction ledger with parameter* $r_0 = 14(l+1)$.

Fast PKI setup. Next, we use the ledger to generate an honest majority PKI from scratch in a number of rounds that is linear in the security parameter. The first idea that we are going to use is that of a *2-for-1 POW* described in [21]. At a high level, the technique allows to do combined mining for two POW schemes in the price of one. In more detail, we can add additional information in the queries to the random oracle, and if the response to the query is less than some value

[7] A similar definitional approach was pursued in [34].

T_1, then we consider it a valid POW of type 1; if it is greater than some value T_2 we consider it as a valid POW of type 2. T_1 and T_2 should be appropriately chosen so that the events of success in either of these POWs are independent. The second POW is used to "mine" transactions, in the same way blocks are mined. This guarantees that the *number of transactions* is proportional to the hashing power of each player. By having parties broadcast their transactions on one hand, and making sure that at least one honest block that contains these transactions is in the chain of all honest parties due to liveness on the other hand, the protocol in [21] manages to achieve consensus assuming an honest-majority hashing power.

In our case, transactions will contain the public keys, and in this way we will obtain an honest-majority PKI. However, in contrast with [21], we cannot let parties start mining transactions from the beginning of the execution, since the adversary would have some additional precomputation time. Instead, we are going to wait for the public ledger to be established, and then use some of the blocks added by honest parties to guarantee that all transactions where mined recently enough. In more detail, any POW will be represented by a triple $\langle w, ctr, label\rangle$. The verification procedure for "block level" POWs ("block POWs" for short) will be of the form

$$H(ctr, \langle G(w), label\rangle) < T_1,$$

while the verification procedure for the "transaction level" POWs will be of the form

$$[H(ctr, \langle label, G(w)\rangle)]^R < T_2,$$

where $[a]^R$ denotes the reverse of the bitstring a. In w we are going to encode the information needed for each application. For example, in block POWs, w will contain the transactions related to this block as well as the hash of the previous block. Note that by making one hash query of the form $H(ctr, \langle G(w_0), G(w_1)\rangle)$ and only two comparisons, we will be mining POWs of both types at the same time. Moreover, if $\lceil \log(T_1)\rceil + \lceil \log(T_2)\rceil$ is less than κ, where κ is the size of the hash's output, then the events of succeeding in any of the two POWs are independent, since they depend on different bits of the hash which are sampled independently and uniformly at random by the random oracle.

Next, we describe our protocol $\Pi_{\mathsf{PL}}^{\mathtt{PKI}}$ for an honest party P. L_1, L_2 are constants such that $L_1 < L_2$.

- *Initialization.* P runs $\Pi_{\mathsf{PL}}^{\mathtt{Boot}}$, as described so far, until she receives a chain of length at least L_1. We choose L_1 so that it is guaranteed that all security properties hold, and about k new blocks have been inserted in the common-prefix of the chains of all honest players.
- *2-for-1 mining.* Let $\mathcal{C}$ be P's chain at the end of the initialization phase. From now on, she is going to do 2-for-1 POW mining, and include in her transaction POWs (i) the hash of the $(L_1 - k)$-th block of $\mathcal{C}$, and (ii) a randomly generated public key for which she has stored the corresponding secret key. Obviously, a new key must be generated every time she starts mining a new transaction.

Whenever P mines a new transaction, she diffuses it to the network, and whenever she receives one, she includes it in the transactions of the block she is mining.
The first time P receives a chain of length greater or equal to L_2, she runs the Key extraction procedure (below). The phase ends at round $\frac{L_2}{(1-\delta)\gamma}$, where P runs the Termination procedure.

- *Key extraction.* P extracts and stores a set of keys from her current chain according to the following rules: If chain $\mathcal{C}'$ is her chain at this round, she stores any public key which belongs to a transaction that (i) is in the first $L_2 - k$ blocks of $\mathcal{C}'$, and (ii) the hash of the block contained in the transaction matches the hash of the $(L_1 - k)$-th block in her chain.
- *Termination.* P outputs the keys from the key extraction phase and terminates.

Next, we prove that a consistent PKI with an honest majority is generated at the end of the execution of protocol $\Pi_{\mathsf{PL}}^{\mathtt{PKI}}$. Two properties are guaranteed: (1) honest parties output the same set of keys and (2) more than half of these keys have been generated by them. For the rest of this section let α_2, β_2, f_2 be the corresponding values of α, β, f for the difficulty level T_2, e.g. $f_2 = nq\frac{T_2}{2^\kappa}$. The full proof of the following theorem is provided in the full version of the paper.

Theorem 4. *Assume* $3(1+\delta)f < 1$, $l > \frac{(1-\delta)k/\gamma+3}{1-3(1+\delta)f}$, $\gamma \geq (1+\delta)\beta$, *for some real* $\delta \in (0, 1/2)$ *and* $\lceil \log(T_1) \rceil + \lceil \log(T_2) \rceil \leq \kappa$. *Then, for parameters* $L_1 = 14(l+1)(1+\delta)f + 2k$ *and* $L_2 = L_1 + 2k \cdot (1 + \frac{10}{\delta})\frac{(1+\delta)f}{(1-\delta)\gamma}$ *the following hold for protocol* $\Pi_{\mathsf{PL}}^{\mathtt{PKI}}$ *with probability* $1 - e^{-\Omega(\delta^5 k)}$*:*

- *All honest players output the same set of public keys, the size of which is*

$$k\frac{\alpha_2}{\gamma}\frac{20}{\delta} \leq N \leq 60k\frac{f_2}{\gamma}(1 + \frac{10}{\delta});$$

- *the majority of the keys are generated by honest parties; and*
- $\Pi_{\mathsf{PL}}^{\mathtt{PKI}}$ *has round complexity linear in* κ.

Proof (Sketch). First, note that the adversary can start precomputing transactions at most $2k/\gamma$ rounds before the honest parties. Otherwise, she will be unable to predict the hash of their chain as dictated by our protocol, since by the chain quality property the chain of each honest player will contain an honest block near the tail of the chain. Moreover, again by the chain quality and common prefix properties, the adversary will stop mining transactions at most $2k/\gamma$ rounds after the honest parties. After this round, she will be unable to insert her transactions deep enough in the chain for the honest parties to take them into account. Finally, by choosing an appropriate value for δ, we are sure that the number of keys mined by the honest parties is greater than the number of keys mined by the adversary. □

Remark 4. To better understand $\Pi_{\mathsf{PL}}^{\mathsf{PKI}}$ we compute different parameters of the system for the Bitcoin network parameters. Assume that $f = 2\%, \alpha = 1.33\%, \gamma = 1.31\%, \beta = 0.6\%, k = 10$, and $\delta = 0.25$. The choice of f approximately corresponds to a rate of one block per 10 min with a round duration of about 12 s; the adversary's hashing power is half of that of the honest parties. Then, $l \approx 623$, which corresponds in terms of rounds to about 2 h. Moreover, if we set f_2 to be equal to f/k we have that $80 < N < 600$. We note that the parameters of Bitcoin are quite conservative and that's why our runtime suffers. In principle, by carefully analyzing and re-engineering our protocol we can get tighter bounds; many of the design decisions we got here, were made to aid the readability of our work.

Remark 5. The probability that some honest party succeeds in mining at least one transaction is:

$$\Pr[\geq \text{ 1 key}] = 1 - \Pr[0 \text{ keys}] = 1 - (1 - \frac{T_2}{2^\kappa})^{q\frac{20k}{(1-\delta)\gamma\delta}} \geq 1 - e^{-\frac{T_2}{2^\kappa} \cdot q\frac{20k}{(1-\delta)\gamma\delta}}.$$

Hence, by setting $T_2 > \frac{\ln(\frac{1}{\epsilon})2^\kappa(1-\delta)\gamma\delta}{q \cdot 20k}$, each party will obtain at least one key with probability at least $1 - \epsilon$, for any $\epsilon \in (0, 1)$. Note here that T_2 and κ must be carefully chosen to retain the independence of the 2 POWs. In case this is not possible, the *2-for-1* mining phase may be extended.

Consensus and other applications. Next, we describe how $\Pi_{\mathsf{PL}}^{\mathsf{PKI}}$ can be used in other contexts. First, a direct application of our protocol is in the context of *proof of stake* protocols. In this type of protocols, blocks are mined by randomly selecting stake holders with probability proportional to their stake. A typical requirement for bootstrapping such protocols (e.g. [28,30]), is that in the initial state of the economy the majority of the coins is controlled by honest parties. By assigning one coin to each public key produced by our protocol, we can efficiently and securely bootstrap a proof of stake protocol.

A more general application of $\Pi_{\mathsf{PL}}^{\mathsf{PKI}}$ is in solving consensus (aka Byzantine agreement) [31,36], with no trusted setup, and in a number of rounds independent of the number of parties. If parties submit transactions containing their input instead of public keys, it follows that by taking the majority of their output they are going to achieve Byzantine agreement. That is, everyone will agree on the same value (the Agreement property), and if all honest parties have the same input v, they are all going to output v (Validity).

Finally, our protocol for the establishment of an honest-majority PKI enables the application of traditional Byzantine fault-tolerant techniques for ledger creation and maintenance based on "subcommittees" as opposed to mining (cf. [7]) to permissionless networks. Instead of having arbitrary membership authorities, these committees can be elected using our protocol with the guarantee of an honest majority. Note that by changing the difficulty of the transaction-level POW we can force the number of parties in the committee to be in a specific predefined interval.

Reducing the communication cost. While the round complexity of our protocol is independent of the number of parties, this does not hold for its communication cost, measured by the number of transmitted messages. The reason is that in the challenge-exchange phase, all parties have to diffuse their random challenges, thus increasing the communication cost of the protocol by an $O(n)$ factor. We can redesign the challenge-exchange phase so that the number of *different* messages diffused by honest parties is independent of their number, and only depends on the security parameter and the precomputation time available to the adversary.[8] We do this in the following way: instead of having all parties sent a random challenge in order to be sure that the genesis blocks that are later mined are fresh, we demand that each random challenge be accompanied by a POW. This way, all honest parties will be sure that at least one honest challenge is generated with high probability every $O(\kappa)$ rounds. Moreover, honest parties will only diffuse random challenges that are tied to a POW. Thus, the total number of different messages sent will be upper-bounded by the number of POWs that the adversary and the honest parties combined have generated. Also, again different honest parties will have received the same block with at most one round difference. By combining the above ideas, we can again create a graded-agreement-type procedure for the genesis blocks and in the same way achieve consensus. We defer further details to the full version of the paper.

References

1. Andrychowicz, M., Dziembowski, S.: PoW-based distributed cryptography with no trusted setup. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 379–399. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_19
2. Aspnes, J., Jackson, C., Krishnamurthy, A.: Exposing computationally-challenged Byzantine impostors. Technical report YALEU/DCS/TR-1332 (2005)
3. Back, A.: Hashcash (1997). http://www.cypherspace.org/hashcash
4. 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)
5. Bitcoinwiki: Genesis block. https://en.bitcoin.it/wiki/Genesis_block
6. Borderding, M.: Levels of authentication in distributed agreement. In: 10th International Workshop on Distributed Algorithms, WDAG 1996 (1996)
7. Cachin, C.: Architecture of the hyperledger blockchain fabric. In: Workshop on Distributed Cryptocurrencies and Consensus Ledgers (2016)
8. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
9. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. IACR Cryptology ePrint Archive 2000:67 (2000)
10. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS 2001, pp. 136–145. IEEE Computer Society (2001)

[8] Note, that each diffusion requires sending the same message at least $O(n)$ times.

11. Cohen, R., Coretti, S., Garay, J., Zikas, V.: Probabilistic termination and composability of cryptographic protocols. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 240–269. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_9
12. Considine, J., Fitzi, M., Franklin, M.K., Levin, L.A., Maurer, U.M., Metcalf, D.: Byzantine agreement given partial broadcast. J. Cryptol. **18**(3), 191–217 (2005)
13. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
14. Douceur, J.R.: The sybil attack. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45748-8_24
15. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
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). https://doi.org/10.1007/3-540-48071-4_10
17. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-NG: a scalable blockchain protocol. CoRR, abs/1510.02037 (2015)
18. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_28
19. Fitzi, M.: Generalized communication and security models in Byzantine agreement. Ph.D. thesis, ETH Zurich, Zürich, Switzerland (2003)
20. Garay, J.A., Katz, J., Koo, C., Ostrovsky, R.: Round complexity of authenticated broadcast with a dishonest majority. In: FOCS 2007, pp. 658–668 (2007)
21. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
22. Garay, J.A., Kiayias, A., Leonardos, N., Panagiotakos, G.: Bootstrapping the blockchain-directly. IACR Cryptology ePrint Archive 2016:991 (2016)
23. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC. ACM (1987)
24. Juels, A., Brainard, J.G.: Client puzzles: a cryptographic countermeasure against connection depletion attacks. In: NDSS. The Internet Society (1999)
25. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. IACR Cryptology ePrint Archive 2011:310 (2011)
26. Katz, J., Miller, A., Shi, E.: Pseudonymous secure computation from time-lock puzzles. IACR Cryptology ePrint Archive 2014:857 (2014)
27. Kiayias, A., Panagiotakos, G.: Speed-security tradeoffs in blockchain protocols. Technical report, IACR, Cryptology ePrint Archive (2015)
28. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
29. King, S.: Primecoin: cryptocurrency with prime number proof-of-work, July 2013. http://primecoin.io/bin/primecoin-paper.pdf
30. King, S., Nadal, S.: PPcoin: peer-to-peer crypto-currency with proof-of-stake. Self-published paper, 19 August 2012
31. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)

32. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). http://bitcoin.org/bitcoin.pdf
33. Nakamoto, S.: Bitcoin open source implementation of P2P currency, February 2009. http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source
34. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454 (2016)
35. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 643–673. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_22
36. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
37. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA (1996)
38. Yao, A.C.: Protocols for secure computations (extended abstract). In: FOCS, pp. 160–164. IEEE (1982)