Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Algorand: A secure and efficient distributed ledger ☆

Jing Chen [a,*], Silvio Micali [b]

[a] Computer Science Department, Stony Brook University, Stony Brook, NY 11794, USA
[b] CSAIL, MIT, Cambridge, MA 02139, USA

ARTICLE INFO

Article history:
Received 15 February 2018
Received in revised form 26 January 2019
Accepted 4 February 2019
Available online xxxx

Keywords:
Public ledger
Blockchain
Byzantine agreement
Distributed computation
Cryptographic self-selection
Permissionless system

ABSTRACT

A *distributed ledger* is a tamperproof sequence of data that can be publicly accessed and augmented by everyone, without being maintained by a centralized party. Distributed ledgers stand to revolutionize the way a modern society operates. They can secure all kinds of traditional transactions, such as payments, asset transfers and titles, in the exact order in which the transactions occur; and enable totally new transactions, such as cryptocurrencies and smart contracts. They can remove intermediaries and usher in a new paradigm for trust. As currently implemented, however, distributed ledgers scale poorly and cannot achieve their enormous potential.

In this paper we propose *Algorand*, an alternative, secure and efficient distributed ledger. Algorand is permissionless and works in a highly asynchronous environment. Unlike prior implementations of distributed ledgers based on "proof of work," Algorand dispenses with "miners" and requires only a negligible amount of computation. Moreover, its transaction history "forks" only with negligible probability: that is, Algorand guarantees the finality of a transaction the moment the transaction enters the ledger.

1. Introduction

Money is becoming increasingly virtual. It has been estimated that about 80% of United States dollars today only exist as ledger entries [9]. In an ideal world, in which we could count on a universally trusted central entity, immune to all possible cyber attacks, money could be solely electronic. Unfortunately, we do not live in such a world. Accordingly, decentralized cryptocurrencies such as Bitcoin [36] and "smart contract" systems such as Ethereum [17] have been proposed. At the heart of these systems is a shared *ledger* that reliably records a sequence of transactions, as varied as payments and contracts, in a tamperproof way. The technology of choice to guarantee such tamperproofness is the *blockchain*. Blockchains are behind applications such as cryptocurrencies (most prominently, Bitcoin [36]), financial applications (e.g., [17]), and the Internet of Things (e.g., [42]). Several techniques to manage blockchain-based ledgers have been proposed: *proof of work* [36], *proof of stake* [41], *practical Byzantine fault-tolerance* [5], or some combinations of them.

☆ This paper is based on early versions of the arXiv paper by the second author [32], a paper itself based on [24]. Algorand's technologies are the object of the following patent applications: US62/117,138 US62/120,916 US62/142,318 US62/218,817 US62/314,601 PCT/US2016/018300 US62/326,865 62/331,654 US62/333,340 US62/343,369 US62/344,667 US62/346,775 US62/351,011 US62/653,482 US62/352,195 US62/363,970 US62/369,447 US62/378,753 US62/383,299 US62/394,091 US62/400,361 US62/403,403 US62/410,721 US62/416,959 US62/422,883 US62/455,444 US62/458,746 US62/459,652 US62/460,928 US62/465,931.

* Corresponding author.
E-mail addresses: jingchen@cs.stonybrook.edu (J. Chen), silvio@csail.mit.edu (S. Micali).

https://doi.org/10.1016/j.tcs.2019.02.001
0304-3975/

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Currently, however, ledgers can be inefficient to manage. For example, Bitcoin's proof-of-work approach (based on the original concept of [15]) requires a vast amount of computation, is wasteful and scales poorly. In addition, it *de facto* concentrates power in few hands.

We therefore wish to put forward a new method to implement a distributed ledger that offers the convenience and efficiency of a centralized system run by a trusted and inviolable authority, without the inefficiencies and weaknesses of current decentralized implementations. We call our approach *Algorand*, because we crucially rely on algorithmic randomness for its efficiency.

For concreteness, in this paper we focus on Algorand as a payment system. Essentially, each user is identified with his public key. The system starts with an initial set of users, each owning a given amount of money. The initial status is assumed to be common knowledge. At any time, each public key can make a payment: that is, transfer all or part of the money it currently owns to another public key, by means of a digital signature. Roughly said, the goal of the system is to organize all payments into an ordered sequence of *blocks*, $B^1, B^2, \ldots$, each containing a set of payments, so as to guarantee the following three properties in absence of any centralized authority: (P1) each block quickly becomes universally known; (P2) all payments in each block B^r are valid, relative to the amount of money each payer owns according to the initial status and the payments in preceding blocks; and (P3) each valid payment quickly appears in a block.

Our model. In our system we assume the availability of a digital signature scheme secure in the sense of [23], such that each message has a *unique* valid signature (even when a public key is adversarially chosen); see [34] for an example. We also assume the availability of a random oracle [22]—or a hash function H modeled as a random oracle, as done in Bitcoin.

Our system is *permissionless*: any user can join at any time. The mechanism for a player to join Algorand is the same main mechanism for joining Bitcoin. Namely, a new user i joins the system when an already-existing user j makes a payment to i.[1]

The users have individual and asynchronous clocks, and all honest users' clocks have the same speed. Colloquially, "a minute is a minute for all of them." Users communicate by propagating messages (i.e., via peer-to-peer gossiping) [11].[2] We assume that each message m whose propagation is initiated by an honest user is received by (almost) all honest users within a fixed amount of time that solely depends on m's length.[3] Specifically, as our system envisages only two classes of messages—*control messages* which are several hundred bits long, and *blocks* which are (chosen to be) several mega-bytes long, we let λ and Λ be the time upper-bounds for the two classes, respectively. A similar communication model is considered by [37] for analyzing Bitcoin in asynchronous networks. Following [20], it is well known that consensus is impossible to achieve in a totally asynchronous network by any deterministic protocol, where messages may arrive after arbitrarily long delays, even with a fixed set of users and a single malicious user. Recently, [28] achieves consensus in the totally asynchronous model with fixed users, with expected $O(n^3)$ communication and polynomial computation, when the number of faulty users is less than $\frac{n}{10^6}$.

In this paper we consider a powerful but computationally bounded Adversary, which (1) instantaneously corrupts any user he wants, whenever he wants; (2) chooses the actions of all corrupted users; and (3) introduces new users into the system whenever he wants. At no time, however, can the corrupted users collectively own more than 1/3 of the total amount of money in the system. Also, the Adversary cannot forge signatures of honest users except with negligible probability. It is important to emphasize that the Adversary *fully determines* when each user receives a specific message, as long as it is within the corresponding time bound. Thus, messages may reach different users under different orders and at different times.

Algorand's high-level structure. Algorand is an asynchronous distributed protocol organized in *rounds*. Conceptually, rounds are non-overlapping time intervals for a single user, but different users' consecutive rounds may overlap due to asynchrony. Round r is devoted to construct the r-th block, B^r. Each user i starts his own round r the moment he is sure about block B^{r-1}.

At the highest level, round r starts by randomly selecting and publicizing (the identity of) a user, ℓ^r, the round *leader*. The leader constructs, digitally signs, and propagates a block B, which is his own candidate for the r-th block and includes a set of new and valid payments. Next, a small set of *selected verifiers*, SV^r, referred to as the *committee*, is randomly chosen and publicized. The size of the committee is such that, with overwhelming probability it has at least a 2/3 honest majority. The committee reaches *Byzantine agreement* [40,19,13,30] on the block B proposed by ℓ^r. Upon termination, each honest verifier locally outputs his own block, whose hash value he digitally signs and propagates. Block B^r is defined to be the block that has been signed by a given number of properly chosen verifiers. We now summarize the performance of our protocol.

[1] Indeed, we focus on Algorand as a payment system. In principle, any user may join the system by publicizing his public key. However, a key with 0 balance cannot make payment to others or participate in block-building, thus we ignore their existence.

[2] Essentially, when a message m is propagated, every user i receiving m for the first time randomly and independently selects a small number of active users, his "neighbors", to whom he forwards m, possibly until he receives their acknowledgments. The propagation terminates when no user receives m for the first time.

[3] Our protocol works if m is received by a sufficiently high fraction, say 95%, of the honest users, within a fixed amount of time. To simplify the discussions, we will not deal with this scenario in the analysis.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Theorem 1. (informally stated) *The following properties hold with overwhelming probability for each round $r \geq 0$:*

- *All honest users agree on the same block B^r, and all payments in B^r are valid.*
- *Let $h > 2/3$ be the fraction of money in the system collectively owned by honest users, and $p_h = h^2(1 + h - h^2)$. The leader ℓ^r is honest with probability at least p_h.*
- *When ℓ^r is honest, the block B^r is generated by ℓ^r and round r takes at most $4\lambda + \Lambda$ time.*
- *When ℓ^r is malicious, round r takes in expectation at most $(\frac{12}{p_h} + 8)\lambda + \Lambda$ time.*
- *All honest users become sure about B^r within time λ of each other.*

Indeed, different from all other existing blockchains, Algorand essentially never "forks". A user of Algorand can rely on a new block B^r as soon as his own round r terminates, and all honest users quickly learn about the agreement on B^r.

Forthcoming work. In this paper we present and analyze Algorand under widely-adopted network assumptions for distributed ledgers, where messages propagated by honest users reach all honest users within a bounded amount of time. In a forthcoming paper, we strengthen our protocol by allowing the Adversary to arbitrarily *partition* the network for an arbitrarily long time. Essentially, for a large interval of time, the Adversary is allowed to complete control the delivery of messages: fully determining which message is delivered to which user and at what time, without any bounded time guarantee.

In other forthcoming papers we will also discuss quite different concerns, such as how to properly distribute initial money so as to accelerate the adoption of the system, and incentive mechanisms. Note that traditional cryptocurrencies require incentives. For example, in Bitcoin, since miners have to consume and pay for a lot of electricity, incentives are necessary to reimburse them for their large expenses. By contrast, in Algorand, the required amount of computation is trivial, thus there is little to reimburse. This said, it is possible to introduce incentives in Algorand. However, in line with Algorand's mathematically rigorous approach, these incentives are engineered so as to be able to prove that they do not interfere with our stated liveness and soundness properties. (For instance, in the case of Bitcoin, the rise of mining pools can be attributed to a set of poorly engineered incentives.) It is thus to be expected that incentive engineering in Algorand is a separate project.

2. Main challenges and overview of algorand

Implementing the high-level structure of Algorand to achieve the desired properties of a blockchain requires meeting notable challenges. We now briefly discuss the challenges and our techniques.

From honest majority of users to honest majority of money. In Algorand a real person may own arbitrarily many public keys. To avoid Sybil attacks [14], each public key is selected to be a leader or a verifier with probability proportional to the amount of money owned by it. In particular, our protocol works under the *honest-majority-of-money* (HMM) assumption. To highlight our main techniques, however, we start by assuming that each user owns only one public key, and present the system in Sections 4 and 5 under the simpler *honest-majority-of-user* (HMU) assumption. In Section 6, we show how to modify our protocol to work under the HMM assumption.

Cryptographic self-selection. It is easy to randomly and *publicly* select ℓ^r and SV^r from the set of users, based on r or $H(B^{r-1})$. However, even though the latter is random, the identities of the selected users become public right away, and our powerful Adversary could immediately corrupt them before they could honestly take any action. In addition, since our system is permissionless, the Adversary could bring in any user i he wants, once he realizes that i will be ℓ^r or in SV^r.

Instead, we introduce *cryptographic self-selection*. Assume for a moment that at the very beginning of round r, a quantity Q^r is, by magic, randomly selected and made universally available. Then, each current user i is able to *privately* compute the (256-bit) string $x_i^r = H(SIG_i(Q^r))$: that is, digitally sign Q^r and then hash it. Note that x_i^r is not only random, but also unique to i and r. This is because Q^r is uniquely and publicly associated to r, and because the adopted signature scheme guarantees that at most one string can be i's digital signature of Q^r. The string x_i^r is interpreted to be the binary expansion of a (256-bit) number between 0 and 1. If this number is less than a given threshold $p_v \in (0, 1)$, then i is a member of SV^r and $\sigma_i^r \triangleq SIG_i(Q^r)$ is his committee *credential*. If x_i^r is smaller than another threshold $p_l < p_v$, then i is a *potential leader*. For potential leaders, we only care that at least one of them is honest with reasonable probability, thus it suffices to randomly select only a few dozen users as potential leaders.

We will discuss the selection of the quantity Q^r in a moment. For now it is important to note that, under cryptographic self-selection, whether or not an honest user i is a potential leader or a verifier is known only by himself, until *after* user i has fulfilled his duty according to his selected role. Indeed, since the Adversary cannot predict $SIG_i(Q^r)$, $H(SIG_i(Q^r))$ is totally random to him.

Cryptographic self-selection is crucial not only to the security of Algorand, but also to its efficiency. To have millions of users engage in a communication protocol to select the leader ℓ^r and the committee SV^r would be totally unwieldy. By contrast, in Algorand each user i selects himself without the need to communicate with anyone. He does so by only performing fast internal computation: one digital signature and one hash. That is, i runs his own lottery: a lottery that is guaranteed to be fair and produces an unforgeable winning ticket when he wins.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Timing guarantee. Different from existing blockchains where all honest users will *eventually* accept the same block B^r, Algorand ensures that all honest users make up their mind about the same B^r within a small and fixed amount of time from each other. More precisely, this amount is λ, the upper-bound of the time by which a short control message (e.g., a credential) propagated by an honest user reaches the other honest users. Note that this amount does not depend on the size of the blocks.

This timing guarantee cannot be taken for granted, as Algorand does not require synchronized clocks, neither does a user know how many messages he should expect. More importantly, the underlying Byzantine agreement does not guarantee that all users will halt in the same step: when an honest user halts, he knows that all honest users *eventually* halt. The users are offset by at most 3 steps in the Byzantine agreement, but this is not enough for Algorand. To solve this problem, Algorand utilizes the propagation network and associates with each block B^r a proper *certificate*, generated besides the output of the Byzantine agreement. Different users may see different certificates for the same B^r. When an honest user sees his own certificate at his own time t, not only he knows an agreement has been reached on B^r, he also knows all the other honest users know the agreement within a time interval around t, of length at most λ. As shown in the protocol, this time-offset is shorter than a single step in the Byzantine agreement.

Leader selection. As soon as a user i is sure about block B^{r-1}, he secretly checks whether he is a potential leader for round r. If so, he secretly prepares and digitally signs B_i^r, his proposal for block B^r. Then he propagates B_i^r and, at the same time but as a separate message, his credential σ_i^r. Upon receiving σ_i^r, every user can verify its authenticity: namely, by verifying i's signature of Q^r and by comparing its hash with p_l. The *leader* ℓ^r of round r is defined to be the user whose hashed credential is the *smallest* among all potential leaders. However, no individual user knows whom the leader is unless he has seen the credentials of *all* potential leaders.

Of course, upon receiving σ_i^r, the Adversary learns that i is a potential leader and can immediately corrupt i. However, he cannot "confiscate" B_i^r or σ_i^r. Indeed, both strings are already being virally propagated, and the Adversary cannot do more to stop them than a government could with a message propagated by Wikileaks.

Leveraging our time invariant, if a user i becomes sure about block B^{r-1} at (his own!) time t, i knows that by time $t+\lambda$ all honest users are sure about B^{r-1}. Thus, by time $t+2\lambda$, i knows that he has received the credentials of all *honest* potential leaders. Then, i compares the hashes of all received credentials and individually decides ℓ_i^r, the leader of round r *in i's opinion,* to be the potential leader whose credential's hash is the *smallest among those received by i by time* $t+2\lambda$.

If the leader ℓ^r is honest, then $\ell_i^r = \ell^r$ *for all honest users i*. This guarantee, however, does not hold when ℓ^r is malicious. For instance, a malicious ℓ^r may start propagating his credential not when he becomes sure about B^{r-1}, but sufficiently later, so that only some users receive it "in time" according to their own clocks, and $\ell_i^r = \ell^r$ only for them. This confusion about the round-r leader may translate into initial confusion about the r-th block.[4] Fortunately, we are able to resolve this confusion and any other attack that the Adversary may launch by the subsequent Byzantine agreement (BA), and ensure that all honest users will end up with the same block B^r.

A special byzantine agreement protocol. Asking all users to participate in a BA protocol is highly infeasible and unscalable, as the system may have millions of users. Instead, in Algorand only users in SV^r are called to reach agreement on B^r. The initial value of each verifier i in the BA protocol is the hash of the block that he received from ℓ_i^r. Upon termination, every honest verifier i outputs the same hash $H(B^r)$, which he digitally signs and propagates together with his credential. This enables all users to learn B^r, by seeing which block is endorsed by sufficiently many verifiers.

For the BA protocol to work, not only must the selected committee have a 2/3 honest majority, it also needs to satisfy several stronger conditions formalized in Section 4. The expected committee size (e.g., thousands of users) is much smaller than the number of users in the system, but is still too large to run traditional BA protocols. We meet this challenge by (1) designing a new BA protocol based on the recent binary Byzantine agreement (BBA) protocol of [33] for synchronous environments; and (2) successfully adapting it to our substantially asynchronous environment. Our BA protocol starts with a *graded consensus protocol* (GC) and continues with the protocol $BBA^\star$ of [33]. It is extremely efficient in a synchronous setting: for any number of players n with $n \geq 3t+1$, where t is the maximum number of dishonest players, it finishes in 11 steps in expectation.

A new challenge, however, arises in our setting. In particular, although the Adversary cannot pre-determine which honest users are in SV^r, the members of SV^r must propagate their credentials together with their first messages for the BA protocol, in order to establish whose messages should be taken into consideration. Once they do, our powerful Adversary can immediately corrupt them and oblige them to send the messages he wants for all future steps. Indeed, cryptographic self-selection suffices to secure the potential leaders' messages, as each potential leader essentially only sends a single message (i.e., a block). Things are more challenging in the BA protocol, because the members of SV^r must continue to send messages over several steps.

We meet this additional challenge via a novel property of our BA protocol: *player-replaceability*. That is, no internal states need to be maintained by a player from one step to another, and the protocol correctly reaches consensus even if each step is executed by a totally new (independently and randomly selected) set of players. Accordingly, we do not ask each user i to select himself for participating the entire BA protocol. Rather, user i secretly determines whether he is a

[4] In addition, the malicious ℓ^r may propagate different proposed blocks to different users.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

selected verifier for *each step s*. Specifically, $i \in SV^{r,s}$ if and only if $H(SIG_i(r, s, Q^r)) < p_v$, and i's corresponding credential is $\sigma_i^{r,s} \triangleq SIG_i(r, s, Q^r)$. When his time to act in step s arrives, i propagates $\sigma_i^{r,s}$ together with his (r, s)-message $m_i^{r,s}$. Once i has done so, the Adversary may certainly corrupt him, but cannot stop $m_i^{r,s}$ from reaching the other honest users. Moreover, by corrupting i, the Adversary has no more control on the rest of the BA protocol than he has by corrupting a random users: the verifiers of all future steps will be randomly and independently selected.

The quantity Q^r. As said, cryptographic self-selection hinges on a random quantity Q^r to become universally available only at the start of round r. In Algorand Q^r is deduced from the just-agreed block B^{r-1}, thus any user who is sure about B^{r-1} is able to compute Q^r by himself.

For this approach to work, we must overcome the following difficulty. If the leader ℓ^{r-1} of round $r-1$ were malicious, he could examine many blocks and choose one to propose, so that a malicious user will be the leader of round r with high probability. Namely, ℓ^{r-1} chooses B^{r-1} and thus Q^r such that $H(SIG_i(Q^r))$ is very small for some malicious user i. To avoid this problem and a score of similar ones, we ensure that Q^r is substantially out of the control of ℓ^{r-1}. Essentially, we let $Q^r \triangleq H(\sigma_{\ell^{r-1}}^{r-1}, r)$, where $\sigma_{\ell^{r-1}}^{r-1}$ is the unique credential of ℓ^{r-1} for being a potential leader.

Still, another problem lurks on. When ℓ^{r-1} is malicious, Q^r may be known by the Adversary even before round $r-1$ starts. Even if no existing malicious user i has $H(SIG_i(Q^r))$ very small, in our permissionless system the Adversary can bring in a new malicious user j, choosing his public key so that $H(SIG_j(Q^r))$ is unusually small and j becomes the leader of round r with high probability. We handle this problem by demanding that all users eligible to become potential leaders or verifiers in round r of the protocol must already be in the system by round $r-k$, where the *look-back parameter* k is a fixed and suitably large integer (e.g., $k = 100$). This works if Q^r were totally unpredictable to the Adversary in round $r-k$, because at that round the Adversary would not know which user/public-key would be more advantageous for him to bring into the system.

But: *is Q^r totally unpredictable at round $r-k$?* The fact that H is a random oracle and $Q^r = H(\sigma_{\ell^{r-1}}^{r-1}, r)$ certainly helps, but the full story is more complex. Indeed, we want the system secure not only under the attacks listed here, but under *all* attacks the Adversary may be able to launch, including those that haven't been identified yet. We defer the technical details to Section 5. In a nutshell, with the above construction of Q^r and a suitable choice of the look-back parameter k, we show via a Markov-chain-based analysis that, if the honest majority among all users is h, then the probability that the leader of round r is honest is close to h.

Ephemeral keys. Although the Adversary cannot predict beforehand which users will be the leader or the verifiers of round r, he would know their identities after seeing their messages, and could then corrupt all of them and oblige them to certify a fake block $\widetilde{B}^r$. Since this block might be propagated after the legitimate one, users that have been paying attention would not be fooled.[5] Nonetheless, $\widetilde{B}^r$ would be syntactically correct and we want to prevent it from being manufactured.

We do so via *ephemeral keys*. Essentially, for his round-r-step-s message, each verifier $i \in SV^{r,s}$ uses an ephemeral public key $pk_i^{r,s}$. Different from his long-term public key, which identifies him (e.g., when he signs a payment), $pk_i^{r,s}$ is single-use-only and, once used, i will destroy the corresponding secret key $sk_i^{r,s}$. Thus, if a verifier is corrupted later on, the Adversary cannot force him to sign anything else he did not originally sign. Naturally, we must ensure that it is impossible for the Adversary to compute a fake new key $\widetilde{pk}_i^{r,s}$ and convince an honest user that it is the ephemeral key of $i \in SV^{r,s}$ for step s of round r. This goal can be achieved via Merkle trees [31], but costly. Instead, we achieve it via a novel use of identity-based signature [44]; see Section 8.

3. Related work

Blockchains (i.e., sequences of blocks, each containing the hash of the previous one, so as to guarantee the tamperproofness of all their data) go back to Nakamoto [36] and are used by almost all later proposals, including ours.[6] In a very recent paper [21], Gilad, Hemo, Micali, Vlachos and Zeldovich have validated the scalability of Algorand, in a realistic and challenging setting.

Bitcoin and proof-of-work. Bitcoin was the first cryptocurrency based on *proof-of-work*: a block includes a random string ρ, and a user tries different ρ's until the hash of the block has a desired number of leading 0's. This approach is very expensive. Currently, the electronic power consumed for block generation in Bitcoin is comparable to that consumed by the entire country of Iceland [4]. The consumption may dramatically increase in the future, if new users join the system. By contrast, in Algorand, block generation requires only a negligible amount of computation.

Bitcoin assumes that malicious entities do not have the majority of the computational power devoted to block generation. Else, they would be able to modify the blockchain as they please. Unfortunately, Bitcoin causes the power of generating

[5] Consider corrupting the news anchor of a major TV network and broadcasting today that Clinton won the last presidential election. Most of us would recognize it as a hoax. But someone getting out of a coma might be fooled.

[6] An earlier version of this paper [7] also introduces *blocktrees*, built on top of our blockchain and helping prove the content of a past block in a much more convenient manner. We do not discuss blocktrees in this version of the paper. Structures different from simple blockchains are also considered in [29,38].

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

blocks to fall into fewer and fewer hands. Today, due to the exorbitant amount of computation required, a user trying to generate a new block using an ordinary desktop expects to lose money, as his expected cost for the necessary electrical-power exceeds his expected reward. Only using *pools* of specially built computers that do nothing but "mine new blocks", one might expect to make a profit. Thus, today there are, *de facto*, two disjoint classes of Bitcoin users: ordinary users, who only make payments, and specialized mining pools, which only search for new blocks. It should therefore not be a surprise that, as of recently, the total computation power for block generation concentrates within just *five* pools. In such a system, the assumption that a majority of the computational power is honest becomes less credible.

Another problematic aspect of Bitcoin is that the (longest) blockchain is not always unique. Indeed, its latest portion often *forks*: the blockchain may be—say—$B_1, \ldots, B_k, B'_{k+1}, B'_{k+2}$, according to one user, and $B_1, \ldots, B_k, B''_{k+1}, B''_{k+2}$ according to another user. Only after several blocks have been added to the chain, can one be reasonably sure that a block is stable: that is, is part of the blockchain according to all users. Thus, rather than relying right away on the payments contained in the last-added block, one should wait for the block to be sufficiently deep in the blockchain. In sum, proof-of-work approaches (e.g., [36] and [17]) are quite orthogonal to ours.

Protocols without proof-of-work. Random sortition/selection was practiced across centuries—e.g., by the republics of Athens, Florence, and Venice [45]. In modern judicial systems, it is often used to choose juries. Recently, it has also been advocated for elections [6]. In our protocol, we resort to cryptography in order to select each verifier set $SV^{r,s}$ in a way that is guaranteed to be automatic (i.e., requiring no message exchange) and random.

Our approach could be considered as a pure form of *proof-of-stake* (PoS) [41]. In popular PoS protocols (e.g., [46,3,27]), a user i chooses whether to put some of his money as his 'stake'. The stake of i cannot be spent, and can be confiscated if i is caught cheating. The power of i in proposing a new block is proportional to his stake. A problem with this approach is that, in general, a user cannot afford to render 'hostage' but a small portion of his total money. Accordingly, it is quite possible that the system falls prey to malicious people with big pockets, posting large stake in order to gain control of the system. The fact that their stakes are confiscated, if they are caught cheating, may not be a credible deterrent: the money they may gain by cheating may vastly exceed their stake. By contrast, in Algorand, every user always has all her money at her disposal, and her total money (as opposite to the portion that she can afford to put at stake) determines her power in block building. Moreover, Algorand needs not to confiscate any money: so long as the majority of the money is in honest hands, subverting the system is impossible.

Delegated proof-of-stake (e.g., [2,16]) is another popular PoS approach. Here, a fixed set of delegates are given the power of block building for a fixed amount of time. Thus, a powerful Adversary could corrupt all delegates or mount a denial-of-service attack against them, so as to prevent them from receiving new transactions that should be put in a new block. By contrast, in Algorand the Adversary never knows which users are in charge of block building until it is too late.

The work closest to ours is Sleepy Consensus by Pass and Shi [39]. They also rely on (and kindly credit) Algorand's cryptographic self-selection. But several significant differences exist. In particular, their setting is mainly permissioned and they rely on a Nakamoto-style protocol, thus forks are frequent and one has to wait that a block becomes sufficiently deep in the chain. By contrast, Algorand's blockchain forks only with negligible probability and a new block can be immediately relied upon, even though the Adversary corrupts users immediately and adaptively.

Another work close to ours is the Ouroboros protocol by Kiayias, Russell, David, and Oliynykov [26]. Also their system appeared after ours. It uses cryptographic selection to dispense with proof-of-work in a provable manner, but the selection is not secret and the Adversary knows the selected users beforehand. They circumvent this problem by limiting the Adversary's corruption power. In particular, corruptions are subject to a delay measured in rounds linear in the security parameter. Also relying on Algorand's cryptographic self-selection, Ouroboros Praos [10] now allows corruptions without delay. Both Ouroboros protocols are again Nakamoto-style, thus forks are both unavoidable and frequent. Moreover, their systems are highly synchronous. By contrast, Algorand is fork-free with overwhelming probability, and does not rely on any of these conditions.

Finally, practical Byzantine fault tolerance [5] is a weaker form of Byzantine agreement,[7] and very different from our protocol.

4. The *Algorand* protocol

4.1. Summary of notions and notations in the protocol

- $r \geq 0$ and $s \geq 1$: the current round and the current step (in a given round).
- m: the maximum number of steps in a round.
- H and $\perp$: a random oracle and a distinguished string outside the range of H.
- PAY^r: the *payset* (i.e., the set of payments) of round r.

[7] Essentially, PBFT is a deterministic 3-step protocol, with the set of players fixed and publicly known in advance. A given player proposes a value and all other players use the protocol to *try* to reach agreement on it. However, as implied by [20], any t-step deterministic protocol cannot guarantee to reach Byzantine agreement if there are at least $t+1$ malicious players. Thus an honest player in the protocol, rather than outputting a value with the guarantee that all honest users will output the same value, is allowed to output an extra default value "?", interpreted as "I don't know what the value should be". When this occurs, various deterministic mechanisms exist to replace the proposer with another player, until an honest one is found. For such mechanisms therefore, it is easy for the Adversary to figure out which minority of the users to corrupt so as to prevent agreement for a long time.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

- ℓ^r: the leader of round r.
- B^r: the block of round r, which can be *non-empty* or *empty*. $B^r \triangleq (r, H(B^{r-1}), SIG_{\ell^r}(Q^{r-1}), PAY^r)$ if non-empty; $B^r = B^r_\epsilon \triangleq (r, H(B^{r-1}), Q^{r-1}, \emptyset)$ if empty.[8]
- $Head(B^r)$: the *head* of B^r—that is, the first three components in B^r.
- Q^r: the *seed* of round r (for determining the potential leaders and verifiers of the next round). $Q^r \triangleq H(SIG_{\ell^r}(Q^{r-1}), r)$, if B^r is non-empty; $Q^r \triangleq H(Q^{r-1}, r)$, if B^r is empty.
- PK^r: the set of public keys by the end of round $r-1$ and at the beginning of round r.[9]
- k: the look-back parameter.
- p_l: the probability of a user being a potential leader.
- PL^r: the set of potential leaders of round r, $\{i \in PK^{r-k} : \ .H(SIG_i(r, 1, Q^{r-1})) \leq p_l\}$. $\ell^r \triangleq \arg\min_{i \in PL^r} H(SIG_i(r, 1, Q^{r-1}))$, with ties broken lexicographically.
- p_v: the probability of a user being a verifier.
- $SV^{r,s}$: the set of verifiers of step s of round r. $SV^{r,s} = \{i \in PK^{r-k} : \ .H(SIG_i(r, s, Q^{r-1})) \leq p_v\}$ for $s > 1$; and $SV^{r,1} \triangleq PL^r$.
- $\sigma_i^{r,s}$: the credential of a user i in $SV^{r,s}$ (or in PL^r for $s = 1$)—i.e., $SIG_i(r, s, Q^{r-1})$.
- t_H: the number of signatures needed to certify a block.
- $CERT^r$: a certificate of B^r, containing $Head(B^r)$ and t_H signatures of $H(B^r)$ from proper verifiers of round r. A user i is *sure about* B^r if he possesses $CERT^r$. A user i *knows* B^r if he possesses both B^r and $CERT^r$. The $CERT^r$ seen by different users may be different.
- Λ and λ: Λ upper-bounds the time to propagate a (non-empty) block, and λ upper-bounds the time to propagate one short message per verifier in a step $s \geq 1$.[10] Note that a block with an empty payset is essentially a short message, thus its propagation time is upper-bounded by λ. We assume $\Lambda = O(\lambda)$. Essentially, Λ and λ respectively upper-bound the time needed to execute Step 1 and the time needed for any other step of the Algorand protocol.

4.2. The protocol

The Algorand protocol starts at time 0 with $r = 0$. Since there does not exist "B^{-1}" or "$CERT^{-1}$", syntactically B^{-1} is a parameter including a random string Q^{-1}, and all users know B^{-1} at time 0.

In each step s of a round r, a verifier $i \in SV^{r,s}$ uses his long-term public-secret key pair to produce his credential $\sigma_i^{r,s} = SIG_i(r, s, Q^{r-1})$, as well as $SIG_i\left(Q^{r-1}\right)$ in case $s = 1$. Wlog the signatures are message-retrievable: if $sig_i(m)$ is i's signature for m that is not message-retrievable, then $SIG_i(m) \triangleq (i, m, sig_i(m))$. Verifier i uses his ephemeral key pair, $(pk_i^{r,s}, sk_i^{r,s})$, to sign any other message m. We use $esig_i(m)$ rather than $sig_{pk_i^{r,s}}(m)$ to denote i's ephemeral signature of m in this step, and $ESIG_i(m)$ for the corresponding message-retrievable signature.

Step 1: Block Proposal

Instructions for every user $i \in PK^{r-k}$: User i starts his own Step 1 of round r as soon as he has $CERT^{r-1}$ (thus is sure about B^{r-1}), which allows i to unambiguously compute $H(B^{r-1})$ and Q^{r-1}.

- User i uses Q^{r-1} to check whether $i \in PL^r$ or not. If $i \notin PL^r$, he does nothing for Step 1.
- If $i \in PL^r$, that is, if i is a potential leader, then he does the following.
  1. If i knows $B^0, \ldots, B^{r-1}$, then he collects the round-r payments that have been propagated to him so far and computes a *maximal valid* payset PAY_i^r from them.[a]
  2. If i hasn't known all $B^0, \ldots, B^{r-1}$ yet, then he sets $PAY_i^r = \emptyset$.
  3. Next, i computes his *candidate block* $B_i^r = (r, H(B^{r-1}), SIG_i(Q^{r-1}), PAY_i^r)$.
  4. Finally, i computes the message $m_i^{r,1} = (B_i^r, esig_i(H(B_i^r)), \sigma_i^{r,1})$, destroys his ephemeral secret key $sk_i^{r,1}$, and then propagates two $(r, 1)$-messages, $m_i^{r,1}$ and $(Head(B_i^r), \sigma_i^{r,1})$, separately but simultaneously.[b]

[a] PAY_i^r is *valid* if for every user j, the total payment of j in PAY_i^r is at most what j owns after block B^{r-1}. If there are multiple payments of j with the total amount exceeding what j owns, then i chooses a maximal valid subset of them. Other rules are also possible: for example, j's payments may all be considered invalid in this case.

[b] When i is the leader, $Head(B_i^r)$ (in particular, $SIG_i(Q^{r-1})$) allows others to compute $Q^r = H(SIG_i(Q^{r-1}), r)$.

[8] A non-empty block may still contain an empty payset PAY^r. However, a non-empty block implies the identity of ℓ^r, his credential $\sigma_{\ell^r}^{r,1}$ and $SIG_{\ell^r}(Q^{r-1})$ have all been timely revealed. The protocol guarantees that if ℓ^r is honest then B^r is non-empty with overwhelming probability.

[9] In a non-synchronous system like ours, the notions of "the end of round $r-1$" and "the beginning of round r" need to be carefully defined. Mathematically, PK^r is computed from the initial status and the blocks $B^1, \ldots, B^{r-1}$: it is the set of public keys each owning a positive amount of money after B^{r-1}.

[10] Using elliptic-curve signatures with 32-byte keys as in Bitcoin, a verifier message is at most 200 bytes in Algorand.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Selective Propagation

The technique of *selective propagation* does not affect the protocol's correctness or worst-case performance as stated in Theorem 1, thus readers focusing on the theoretical analysis can safely skip this paragraph. However, when the protocol is implemented in practice, to shorten the actual execution of the whole round it is important that the $(r,1)$-messages are selectively propagated, to reduce the network congestion. Indeed, $(r,1)$-messages include blocks of large sizes, at most one of which may become the block of round r. It is thus unnecessary to have all blocks propagated to the whole network. More precisely, for every user j in the system,

- For the first $(r,1)$-message that he ever receives and successfully verifies,[a] whether it contains a block or is just a credential and a block head, player j propagates it as usual.
- For all the other $(r,1)$-messages that player j receives and successfully verifies, he propagates it *only if* the hash of its credential is the *smallest* among the hashes of the credentials contained in all $(r,1)$-messages he has received and successfully verified so far.
- However, if j receives two different messages of the form $m_i^{r,1}$ from the same player i (thus i is malicious), he discards the second one no matter what the hash value of i's credential is.

It is useful that in Step 1 each potential leader i propagates his credential $\sigma_i^{r,1}$ separately from his message $m_i^{r,1}$ which contains the actual block.[b] Credentials are much smaller than blocks and thus travel faster. Selective propagation thus ensures that only a few blocks, those from potential leaders whose credentials have small hashes, are propagated to the full network. All other proposed blocks will quickly stop being propagated.

[a] That is, all the signatures are correct and, if it is of the form $m_i^{r,1}$, both the block and its hash are valid.
[b] We thank Georgios Vlachos for suggesting this.

Step 2: The First Step of the Graded Consensus Protocol GC

For every user $i \in PK^{r-k}$: User i starts his own Step 2 of round r as soon as he has $CERT^{r-1}$.

- User i waits a maximum amount of time $t_2 \triangleq \lambda + \Lambda$. While waiting, i acts as follows.
  1. After waiting for time 2λ, he finds the user ℓ such that $H(\sigma_\ell^{r,1}) \leq H(\sigma_j^{r,1})$ among all credentials $\sigma_j^{r,1}$ contained in the successfully verified $(r,1)$-messages he has received so far.[a]
  2. If he has known $B^0, \ldots, B^{r-1}$, and if he has received from ℓ a valid message $m_\ell^{r,1} = (B_\ell^r, esig_\ell(H(B_\ell^r)), \sigma_\ell^{r,1})$,[b] then i stops waiting and sets $v_i \triangleq (H(B_\ell^r), \ell)$.
  3. Otherwise, when time t_2 runs out, i sets $v_i \triangleq \perp$.
  4. Once the value of v_i is set, i computes Q^{r-1} from $CERT^{r-1}$ and checks whether $i \in SV^{r,2}$.
  5. If $i \in SV^{r,2}$, i computes $m_i^{r,2} \triangleq (ESIG_i(v_i), \sigma_i^{r,2})$,[c] destroys his ephemeral secret key $sk_i^{r,2}$, and then propagates $m_i^{r,2}$. Otherwise, i stops without propagating anything.

[a] Essentially, user i privately decides that the leader of round r is user ℓ.
[b] The protocol guarantees that i has seen $CERT^0, \ldots, CERT^{r-1}$ when starting round r. If B_ℓ^r contains an empty payset, then there is actually no need for i to see $B^0, \ldots, B^{r-1}$ before verifying whether B_ℓ^r is valid or not.
[c] $m_i^{r,2}$ signals that i considers $H(B_\ell^r)$ to be the hash of the next block, or considers the next block to be empty.

Step 3: The Second Step of GC

For every user $i \in PK^{r-k}$: User i starts his own Step 3 of round r as soon as he has $CERT^{r-1}$.

- User i waits a maximum amount of time $t_3 \triangleq t_2 + 2\lambda = 3\lambda + \Lambda$. While waiting, i acts as follows.
  1. If there exists a value v such that he has received at least t_H valid messages $m_j^{r,2}$ of the form $(ESIG_j(v), \sigma_j^{r,2})$,[a] then he stops waiting and sets $v_i \triangleq v$.
  2. Otherwise, when time t_3 runs out, he sets $v_i \triangleq \perp$.
  3. Once the value of v_i is set, i computes Q^{r-1} from $CERT^{r-1}$ and checks whether $i \in SV^{r,3}$.
  4. If $i \in SV^{r,3}$, i computes $m_i^{r,3} \triangleq (ESIG_i(v_i), \sigma_i^{r,3})$, destroys his ephemeral secret key $sk_i^{r,3}$, and then propagates $m_i^{r,3}$. Otherwise, i stops without propagating anything.

[a] If he has received two valid messages from a user j, respectively containing $ESIG_j(v)$ and a different $ESIG_j(v')$, then they are counted for v and v' respectively, even though they together imply that j is malicious.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Step 4: The Output of GC and The First Step of The Binary Byzantine Agreement $BBA^\star$

For every user $i \in PK^{r-k}$: i starts his own Step 4 of round r as soon as he finishes his own Step 3.

- User i waits a maximum amount of time 2λ.[a] While waiting, i acts as follows.
  1. He computes v_i and g_i, the output of GC, as follows.
     (a) If there exists a value $v \neq \perp$ such that he has received at least t_H valid messages $m_j^{r,3} = (ESIG_j(v), \sigma_j^{r,3})$, then he stops waiting and sets $v_i \triangleq v$ and $g_i \triangleq 2$.
     (b) If he has received at least t_H valid messages $m_j^{r,3} = (ESIG_j(\perp), \sigma_j^{r,3})$, then he stops waiting and sets $v_i \triangleq \perp$ and $g_i \triangleq 0$.[b]
     (c) Otherwise, when time 2λ runs out, if there exists a value $v \neq \perp$ such that he has received at least $\lceil \frac{t_H}{2} \rceil$ valid messages $m_j^{r,j} = (ESIG_j(v), \sigma_j^{r,3})$, then $v_i \triangleq v$ and $g_i \triangleq 1$.[c]
     (d) Else, when time 2λ runs out, he sets $v_i \triangleq \perp$ and $g_i \triangleq 0$.
  2. Once the values v_i and g_i are set, i computes b_i, the input of $BBA^\star$, as follows: $b_i \triangleq 0$ if $g_i = 2$, and $b_i \triangleq 1$ otherwise.
  3. i computes Q^{r-1} from $CERT^{r-1}$ and checks whether $i \in SV^{r,4}$ or not.
  4. If $i \in SV^{r,4}$, he computes $m_i^{r,4} \triangleq (ESIG_i(b_i), ESIG_i(v_i), \sigma_i^{r,4})$, destroys his ephemeral secret key $sk_i^{r,4}$, and propagates $m_i^{r,4}$. Otherwise, i stops without propagating anything.

[a] Thus, the maximum *total* amount of time since i starts his round r could be $t_4 \triangleq t_3 + 2\lambda = 5\lambda + \Lambda$.

[b] Whether sub-step (b) is in the protocol or not does not affect its correctness. However, the presence of sub-step (b) allows Step 4 to end faster if sufficiently many Step-3 verifiers have "signed $\perp$."

[c] It can be proved that the v in this case, if exists, must be unique.

Step s, $5 \leq s \leq m$, $s - 2 \equiv 0 \mod 3$: A Coin-Fixed-To-0 Step of $BBA^\star$

For every user $i \in PK^{r-k}$: i starts his own Step s as soon as he finishes his own Step $s-1$.

- User i waits a maximum amount of time 2λ.[a] While waiting, i acts as follows.
  - *Ending Condition 0:* If at any point there exists a string $v \neq \perp$ and a step s' such that
    (a) $5 \leq s' \leq s$, $s' - 2 \equiv 0 \mod 3$—that is, Step s' is a Coin-Fixed-To-0 step,
    (b) i has received $\geq t_H$ valid messages $m_j^{r,s'-1} = (ESIG_j(0), ESIG_j(v), \sigma_j^{r,s'-1})$, and
    (c) i has received a valid message $(Head(B_\ell^r), \sigma_\ell^{r,1})$ where ℓ is the *second* component of v,
    then, i stops waiting and ends his own execution of round r right away; sets $H(B^r)$ to be the *first* component of v; and sets his own $CERT^r$ to be the set of messages $m_j^{r,s'-1}$ of sub-step (b) together with $(Head(B_\ell^r), \sigma_\ell^{r,1})$.[b]
  - *Ending Condition 1:* If at any point there exists a step s' such that
    (a') $6 \leq s' \leq s$, $s' - 2 \equiv 1 \mod 3$—that is, Step s' is a Coin-Fixed-To-1 step, and
    (b') i has received $\geq t_H$ valid messages $m_j^{r,s'-1} = (ESIG_j(1), ESIG_j(v_j), \sigma_j^{r,s'-1})$,[c]
    then, i stops waiting and ends his own execution of round r right away without propagating anything as a (r,s)-verifier; sets $B^r = B_\epsilon^r$; and sets his own $CERT^r$ to be the set of messages $m_j^{r,s'-1}$ of sub-step (b') together with $Head(B_\epsilon^r)$.
  - If at any point he has received $\geq t_H$ valid $(r, s-1)$-messages $m_j^{r,s-1}$ of the form $(ESIG_j(1), ESIG_j(v_j), \sigma_j^{r,s-1})$, then he stops waiting and sets $b_i \triangleq 1$.
  - Otherwise, when time 2λ runs out, i sets $b_i \triangleq 0$.[d]
  - Once the value b_i is set, i computes Q^{r-1} from $CERT^{r-1}$ and checks whether $i \in SV^{r,s}$.
  - If $i \in SV^{r,s}$, i computes the message $m_i^{r,s} \triangleq (ESIG_i(b_i), ESIG_i(v_i), \sigma_i^{r,s})$ with v_i being the value he has computed in Step 4, destroys his ephemeral secret key $sk_i^{r,s}$, and then propagates $m_i^{r,s}$. Otherwise, i stops without propagating anything.

[a] Thus, the maximum *total* amount of time since i starts his round r could be $t_s \triangleq t_{s-1} + 2\lambda = (2s-3)\lambda + \Lambda$.

[b] User i now knows $H(B^r)$ and is sure about B^r. If he hasn't received B^r yet, he just needs to wait until B^r is propagated to him. He still helps propagating messages as a generic user, but does not initiate any propagation as a (r,s)-verifier. In particular, he has helped propagating all messages in his $CERT^r$, which is enough for our protocol.

[c] In this case, it does not matter what the v_j's are.

[d] That is, when user i does not see enough signatures for 1, he fixes his "coin" b_i to 0. In the next step, which is a Coin-Fixed-To-1 step, if user i does not see enough signatures for 0 then he fixes his "coin" b_i to 1.

Step s, $6 \leq s \leq m$, $s-2 \equiv 1 \mod 3$: A Coin-Fixed-To-1 Step of $BBA^\star$

For every user $i \in PK^{r-k}$: i starts his own Step s as soon as he finishes his own Step $s-1$.

- User i waits a maximum amount of time 2λ. While waiting, i acts as follows.
  - *Ending Conditions 0 and 1:* The same instructions as in a Coin-Fixed-To-0 step.
  - If at any point he has received $\geq t_H$ valid $(r,s-1)$-messages $m_j^{r,s-1}$ of the form $(ESIG_j(0), ESIG_j(v_j), \sigma_j^{r,s-1})$, then he stops waiting and sets $b_i \triangleq 0$.
  - Otherwise, when time 2λ runs out, i sets $b_i \triangleq 1$ (i.e., "coin fixed to 1").
  - Once the value b_i is set, i computes Q^{r-1} from $CERT^{r-1}$ and checks whether $i \in SV^{r,s}$.
  - If $i \in SV^{r,s}$, i computes the message $m_i^{r,s} \triangleq (ESIG_i(b_i), ESIG_i(v_i), \sigma_i^{r,s})$ with v_i being the value he has computed in Step 4, destroys his ephemeral secret key $sk_i^{r,s}$, and then propagates $m_i^{r,s}$. Otherwise, i stops without propagating anything.

Step s, $7 \leq s \leq m$, $s-2 \equiv 2 \mod 3$: A Coin-Genuinely-Flipped Step of $BBA^\star$

For every user $i \in PK^{r-k}$: i starts his own Step s as soon as he finishes his own step $s-1$.

- User i waits a maximum amount of time 2λ. While waiting, i acts as follows.
  - *Ending Conditions 0 and 1:* The same instructions as in a Coin-Fixed-To-0 step.
  - If at any point he has received $\geq t_H$ valid $(r,s-1)$-messages $m_j^{r,s-1}$ of the form $(ESIG_j(0), ESIG_j(v_j), \sigma_j^{r,s-1})$, then he stops waiting and sets $b_i \triangleq 0$.
  - If at any point he has received $\geq t_H$ valid $(r,s-1)$-messages $m_j^{r,s-1}$ of the form $(ESIG_j(1), ESIG_j(v_j), \sigma_j^{r,s-1})$, then he stops waiting and sets $b_i \triangleq 1$.
  - Otherwise, when time 2λ runs out, letting $SV_i^{r,s-1}$ be the set of $(r,s-1)$-verifiers from whom he has received a valid message $m_j^{r,s-1}$, and letting $\ell = \arg\min_{j \in SV_i^{r,s-1}} H(\sigma_j^{r,s-1})$, i sets $b_i \triangleq \mathtt{lsb}(H(\sigma_\ell^{r,s-1}, r))$, where $\mathtt{lsb}$ is the least significant bit.[a]
  - Once the value b_i is set, i computes Q^{r-1} from $CERT^{r-1}$ and checks whether $i \in SV^{r,s}$.
  - If $i \in SV^{r,s}$, i computes the message $m_i^{r,s} \triangleq (ESIG_i(b_i), ESIG_i(v_i), \sigma_i^{r,s})$ with v_i being the value he has computed in Step 4, destroys his ephemeral secret key $sk_i^{r,s}$, and then propagates $m_i^{r,s}$. Otherwise, i stops without propagating anything.

[a] Since H is a random oracle, user i "flips a (almost unbiased) coin" to decide his own bit b_i in this step, if he does not see enough signatures for 0 or for 1.

Reconstruction of the Round-r Block by Generic Users

For every user i in the system: i starts his own round r as soon as he has $CERT^{r-1}$.

- i follows the instructions of each step, participates the propagation of all messages, but does not initiate any propagation in a step where he is not a verifier.
- i ends his own round r by entering Ending Condition 0 or Ending Condition 1 in some step, with the corresponding $CERT^r$.
- From there on, he starts his round $r+1$ while waiting to receive the actual block B^r (if he hasn't already received it), whose hash $H(B^r)$ has been pinned down by $CERT^r$. Of course, if $CERT^r$ indicates that $B^r = B^r_\epsilon$, then i knows B^r the moment he has $CERT^r$.

Our protocol guarantees that each round finishes within m steps with overwhelming probability, thus we do not find it necessary to deal with the rare case where it doesn't finish. A variant of the protocol is to not explicitly impose an upper-bound m on the number of steps.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

5. Analysis of *Algorand*

5.1. The Byzantine agreement in classical settings

The notion of Byzantine agreement was introduced by Pease, Shostak and Lamport [40] for the *binary* case, that is, when every initial value consists of a bit. It was quickly extended to arbitrary initial values (see the surveys of Fischer [19] and Chor and Dwork [8]). By a BA protocol, we mean an arbitrary-value one.

As a warm up, let us describe the Byzantine agreement $BA^\star$ of Algorand in classical settings. Specifically, there are n players and at most t of them are malicious, with $n \geq 3t+1$. Players have synchronized clocks and peer-to-peer communication, and messages are delivered immediately. The protocol $BA^\star$ consists of two parts: the Graded Consensus protocol GC, where each player i outputs a value v_i together with a *grade of certainty* g_i; and the binary Byzantine agreement $BBA^\star$ of [33]. To be consistent with Algorand, below we call the first step of GC "Step 2". Before GC starts, each player i has received a private input v'_i. For each player i, step s and string x, let $\#_i^s(x)$ denote the number of players from whom player i has received x in step s. Following the literature's convention and without loss of generality, $n = 3t+1$ in the description below.

THE GRADED CONSENSUS PROTOCOL GC

STEP 2. *Each player i sends his private input v'_i to all players.*

STEP 3. *Each player i sends to all players the string x if and only if $\#_i^2(x) \geq 2t+1$.*

OUTPUT DETERMINATION. *Each player i outputs the pair (v_i, g_i) computed as follows:*

- *If $\#_i^3(x) \geq 2t+1$ for some x, then $v_i = x$ and $g_i = 2$.*
- *Else, if $\#_i^3(x) \geq t+1$ for some x, then $v_i = x$ and $g_i = 1$.*
- *Else, $v_i = \perp$ and $g_i = 0$.*

Remark. The notion of *graded consensus* is derived from that of *graded broadcast*, put forward by Feldman and Micali in [18] by strengthening the notion of a *crusader agreement*, which was introduced by Dolev [12] and refined by Turpin and Coan [47]. In [18], the authors provided a 3-step graded broadcasting protocol for $n \geq 3t+1$. A more complex graded broadcasting protocol for $n \geq 2t+1$ has later been found by Katz and Koo [25].

The lemma below essentially follows from the graded broadcasting protocol in [18], thus its proof has been omitted.

Lemma 5.1. *When $n \geq 3t+1$, the protocol GC satisfies the following properties:*

1. *For any two honest players i and j, $|g_i - g_j| \leq 1$.*
2. *For any two honest players i and j, if $g_i > 0$ and $g_j > 0$, then $v_i = v_j$.*
3. *If there exists a value v such that $v'_i = v$ for all honest players i, then $v_i = v$ and $g_i = 2$ for all honest players i.*

We now describe the protocol $BA^\star$, again starting with Step 2 to match the steps in Algorand. The initial value of each player i is v'_i.

THE BYZANTINE AGREEMENT PROTOCOL $BA^\star$

STEPS 2 AND 3. *Each player i executes the two steps of GC, with private input v'_i.*

STEP 4. *Each player i computes his output (v_i, g_i) of GC and sets his private binary input for $BBA^\star$ as follows: $b'_i = 0$ if $g_i = 2$, and $b'_i = 1$ otherwise. Player i then executes the first step of $BBA^\star$: that is, he sends b'_i to all players.*

STEPS 5, ... *Each player i executes the remaining steps of $BBA^\star$, until he is able to halt and compute his binary output, b_i, according to $BBA^\star$.*

OUTPUT DETERMINATION. *Each player i outputs the value out_i, computed as follows: $out_i = v_i$ if $b_i = 0$, and $out_i = \perp$ if $b_i = 1$.*

Remark. The protocol $BBA^\star$ is a probabilistic binary BA protocol and is analyzed in [33]. In the worst case, it finishes in 9 steps in expectation. Probabilistic binary BA protocols were first proposed by Ben-Or in asynchronous settings [1]. The protocol $BBA^\star$ is a novel adaptation to public-key settings of the binary BA protocol of [18], which was the first to work in an expected constant number of steps. It worked by having the players themselves implement a *common coin*, a notion proposed by Rabin [43], who implemented it via an external trusted party.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

To analyze $BA\star$, recall that an *arbitrary-value* Byzantine agreement is such that, for any set of values V not containing the special symbol $\perp$, when the players' initial values are from V, every honest player halts with probability 1 and the following two properties hold with probability 1[11]:

1. *Agreement:* There exists $out \in V \cup \{\perp\}$ such that $out_i = out$ for all honest players i.
2. *Consistency:* If all honest players have the same initial value $v \in V$, then $out = v$.

We have the following theorem.

Theorem 5.2. *When $n \geq 3t+1$, $BA\star$ is an arbitrary-value Byzantine agreement.*

Proof. We first prove Consistency. Assume there exists some value $v \in V$ such that $v'_i = v$ for all honest players i. By property 3 of Lemma 5.1, after the execution of GC, all honest players i have $v_i = v$ and $g_i = 2$, thus $b'_i = 0$ at the beginning of $BBA\star$. As $BBA\star$ is a binary Byzantine agreement, by its own Consistency property, $b_i = 0$ for all honest players i at the end of its execution. Accordingly, for all honest players i, $out_i = v_i = v$ at the end of $BA\star$, as desired.

We now prove Agreement, and we distinguish two cases. *Case 1: There exists an honest player i with $g_i = 2$ at the end of GC.* In this case, by property 1 of Lemma 5.1, all honest players j have $g_j \geq 1$. Thus, by property 2 of Lemma 5.1, there exists a value v such that $v_i = v$ for all honest players i at the end of GC, even though their grades may differ. Since $BBA\star$ is a binary BA protocol, by its Agreement property, there exists $b \in \{0,1\}$ such that $b_i = b$ for all honest players i at the end of $BBA\star$. If $b = 1$, then all honest players i output $out_i = \perp$: that is, $out = \perp$. If $b = 0$, then all honest players i output $out_i = v_i = v$: that is, $out = v$. Thus Agreement holds.

Case 2: All honest players i have $g_i \leq 1$ at the end of GC. In this case, $b'_i = 1$ for all of them. By the Consistency property of $BBA\star$, $b_i = 1$ for all of them after $BBA\star$. Thus all honest players i have $out_i = \perp$, and Agreement holds.

Since both Consistency and Agreement hold, $BA\star$ is an arbitrary-value BA protocol. □

As illustrated by the descriptions of *Algorand* and $BA\star$, the former is much more complex than the latter, because achieving consensus in the former's setting is much more challenging than in the classical setting. Next we analyze *Algorand* itself, and we start by introducing the notions and notations used in the analysis.

5.2. Summary of notions and notations in the analysis

- $MSV^{r,s}$ and $HSV^{r,s}$: the set of malicious verifiers and the set of honest verifiers in $SV^{r,s}$.
- $n_l \in \mathbb{Z}^+$ and $n \in \mathbb{Z}^+$: the expected cardinality of $PL^r (= SV^{r,1})$ and of every $SV^{r,s}$ for $s > 1$. $p_l \triangleq \frac{n_l}{|PK^{r-k}|}$ and $p_v \triangleq \frac{n}{|PK^{r-k}|}$.
- $h \in (2/3, 1]$: the fraction of honest users in each PK^r.
- $F \in (0,1)$: the allowed error probability.
- $p_h \in (0,1)$: the probability that the leader of a round r, ℓ^r, is honest. Ideally $p_h = h$. With the existence of the Adversary, the value of p_h will be determined in the analysis.
- $\alpha_i^{r,s}$ and $\beta_i^{r,s}$: respectively the (local) time a user i starts and ends his Step s of round r. Recall that in Algorand, the users' clocks need not be synchronized, but have the same speed. Only for the purpose of the analysis, we consider a reference clock and measure the players' related times with respect to it.
- $L^r \leq m/3$: a random variable representing the number of Bernoulli trials needed to see a 1, when each trial is 1 with probability $\frac{p_h}{2}$ and there are at most $m/3$ trials. If all trials fail then $L^r \triangleq m/3$. Essentially, L^r upper-bounds the time needed to generate block B^r.
- T^r: the time when the first honest user is sure about B^{r-1}. $T^0 = 0$ by initialization.
- I^r: the time interval $[T^r, T^r + \lambda]$ for $r \geq 1$. $I^0 \triangleq \{0\}$.
- t_s: the maximum amount of time since a user i starts his round r till the end of his Step s. Following the description of the protocol, $t_s \triangleq (2s-3)\lambda + \Lambda$ for $s \geq 2$. $t_1 \triangleq 0$.
- *Relationships among various parameters.*
  - The verifiers and potential leaders of round r are selected from the users in PK^{r-k}, where k is set so that the Adversary cannot predict Q^{r-1} back at round $r-k-1$ with probability better than F: otherwise, he will be able to introduce malicious users for round $r-k$, all of which will be potential leaders/verifiers in round r, succeeding in having a malicious leader or a malicious majority in $SV^{r,s}$ for some steps s chosen by him.

[11] Such a Byzantine agreement is said to have *soundness 1*. In general, a Byzantine agreement has soundness $\sigma \in (0,1)$ if Agreement and Consistency hold with probability at least σ.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

- For Step 1 of each round r, n_l is chosen so that with overwhelming probability, $SV^{r,1} \neq \emptyset$.
- For each step $s > 1$ of round r, n and t_H are chosen so that, with overwhelming probability,

$$|HSV^{r,s}| > t_H \quad \text{and} \quad |HSV^{r,s}| + 2|MSV^{r,s}| < 2t_H.$$

The two inequalities together imply that $|MSV^{r,s}| < t_H/2$ and $|HSV^{r,s}| > 2|MSV^{r,s}|$. In particular, $SV^{r,s}$ has a 2/3 honest majority. The closer to 1 the value of h is, the smaller n needs to be. We use (variants of) Chernoff bounds to ensure the desired conditions.

- *Example choices of important parameters.*
  - The outputs of H are 256-bit long.
  - $h = 80\%$ and $n_l = 35$.
  - $\Lambda = 1$ minute and $\lambda = 10$ seconds.
  - $F = 10^{-18}$. (With this error probability, if a new block is generated every second, one should expect for the age of the Universe to see a fork.)
  - $n \approx 4000$, $t_H \approx 0.69n$, $k = 70$, and $m = 180$.

In the analysis we will ignore the computation time, as it is negligible compared to the time needed to propagate messages. In any case, by using slightly larger λ and Λ, the computation time can be incorporated into the analysis directly. Most of the statements below hold "with overwhelming probability," and we may not repeatedly emphasize this fact in the analysis.

5.3. Main theorem

Theorem 1. *The following properties hold with overwhelming probability for each round $r \geq 0$:*

1. *All honest users agree on the same block B^r, and all payments in B^r are valid.*
2. *When the leader ℓ^r is honest, we have the following.*
   - *The block B^r is generated by ℓ^r and all honest users know B^r in the time interval I^{r+1}.*
   - *If $PAY^r = \emptyset$ then $T^{r+1} \leq T^r + 6\lambda$; otherwise B^r contains a maximal payset received by ℓ^r by time $\alpha_{\ell^r}^{r,1}$ and $T^{r+1} \leq T^r + 4\lambda + \Lambda$.*
   - *Let $B^{r'}$ be the last block before B^r with a non-empty payset. If $\ell^{r'}$ was honest or if $T^r - T^{r'+1} \geq \Lambda$, then $PAY^r \neq \emptyset$.*[12]
3. *When the leader ℓ^r is malicious, all honest users become sure about B^r in the time interval I^{r+1}, and $T^{r+1} \leq T^r + (6L^r + 8)\lambda + \Lambda$.*
4. *$p_h = h^2(1 + h - h^2)$ for L^r, and the leader ℓ^r is honest with probability at least p_h.*

Remarks. Before proving the theorem, let us make two remarks.

- *Block-Generation and True Latency.* The time to generate block B^r is defined to be $T^{r+1} - T^r$: that is, the difference between the first time some honest user becomes sure about B^r and the first time some honest user becomes sure about B^{r-1}. When the round-r leader is honest, Property 2 of Theorem 1 guarantees that the *exact* time to generate B^r is at most $4\lambda + \Lambda$,[13] no matter what the precise value of $h > 2/3$ may be. When the leader is malicious, Property 3 implies that the *expected* time to generate B^r is at most $(\frac{12}{p_h} + 8)\lambda + \Lambda$,[14] again no matter the precise value of h. However, the overall expected time to generate B^r depends on the precise value of h. By Property 4, $p_h = h^2(1 + h - h^2)$ and the leader is honest with probability at least p_h, thus

$$\mathbb{E}[T^{r+1} - T^r] \leq h^2(1 + h - h^2) \cdot (4\lambda + \Lambda) + [1 - h^2(1 + h - h^2)][(\frac{12}{h^2(1 + h - h^2)} + 8)\lambda + \Lambda].$$

[12] If $\ell^{r'}$ was malicious and started propagating $B^{r'}$ late, then some honest users may not have received $B^{r'}$ by the end of round r', even though they knew $H(B^{r'})$. However, all honest users would have received $B^{r'}$ by time $T^{r'+1} + \Lambda$ in the worst case.

Our protocol actually guarantees a stronger property. Strictly speaking, we could introduce two additional parameters: Λ_c upper-bounds the amount of time that remains to propagate a block B to (almost) all honest users when at least a constant c fraction of them have already received it; and $\beta^{r',2}$ is the time the last honest verifier in $SV^{r',2}$ finishes Step 2 of round r'. Then $PAY^r \neq \emptyset$ as long as $T^r - \beta^{r',2} \geq \Lambda_c$, with $c \geq 1/4$. Note that Λ_c can be much smaller than Λ, as a message being propagated spreads exponentially fast. Also note that the delay caused by the Adversary is now charged to the remaining steps of round r' rather than to future rounds.

More specifically, the fact that an agreement has been reached on $H(B^{r'})$ implies a constant fraction of honest verifiers in Step 2 of round r' had received $B^{r'}$ when they finished that step. Since these verifiers are randomly chosen from all honest users, a constant c fraction of all honest users had received $B^{r'}$ by time $\beta^{r',2}$. So in the case that $\ell^{r'}$ is malicious, not only PAY^r is non-empty whenever $T^r - \beta^{r',2} \geq \tau_c$, but also it is non-empty with probability at least c even when $T^r - \beta^{r',2} < \tau_c$.

[13] Without loss of generality, $\Lambda \geq 2\lambda$ and round r takes longer when PAY^r is non-empty than when it is empty.

[14] That is, $\mathbb{E}[T^{r+1} - T^r] \leq (6\mathbb{E}[L^r] + 8)\lambda + \Lambda = (6 \cdot \frac{2}{p_h} + 8)\lambda + \Lambda = (\frac{12}{p_h} + 8)\lambda + \Lambda$.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

For instance, if $h = 80\%$ then $\mathbb{E}[T^{r+1} - T^r] \leq 9.2\lambda + \Lambda$.

- λ *vs.* Λ. Note that the size of the messages sent by the verifiers in a step in *Algorand* is dominated by the length of the digital signature keys, which can remain fixed, even when the number of users is enormous. Also note that, in any step $s > 1$, the same expected number n of verifiers can be used, whether the number of users is 100 K, 10 M, or 100 M. This is so because n solely depends on h and F. In sum, therefore, barring a sudden need to increase the length of secret keys, the value of λ remains the same no matter how large the number of users may be in the foreseeable future.
  By contrast, given any transaction rate, the number of transactions grows with the number of users. Therefore, to process all new transactions in a timely fashion, the size of a block should also grow with the number of users, causing Λ to grow too. Thus, in the long run, we should have $\lambda << \Lambda$. Accordingly, it is proper to have a larger coefficient for λ, and actually a coefficient of 1 for Λ.

Now we highlight the framework for proving Theorem 1 and the key components.

Proof of Theorem 1. We prove Properties 1–3 by induction. By the initialization of the protocol, they automatically hold for "round -1" when $r = 0$.

Since B^{r-1} is uniquely defined by the inductive hypothesis, the set $SV^{r,s}$ is uniquely defined for each step s of round r. By the choice of n_l, $(PL^r =) SV^{r,1} \neq \emptyset$ with overwhelming probability, thus the leader ℓ^r is well defined. We now state the following two lemmas, respectively proved in Sections 5.4 and 5.5, distinguishing whether ℓ^r is honest or not.

Lemma 5.3 *(Completeness Lemma). Assume Properties 1–3 hold for rounds* $0, \ldots, r-1$. *When the leader* ℓ^r *is honest, Properties 1 and 2 hold for round* r.

Lemma 5.4 *(Soundness Lemma). Assume Properties 1–3 hold for rounds* $0, \ldots, r-1$. *When the leader* ℓ^r *is malicious, Properties 1 and 3 hold for round* r.

Following the above two lemmas, Properties 1–3 hold by induction. Finally, we restate Property 4 as the following lemma, which is proved in Section 5.6.

Lemma 5.5. *Given Properties 1–3 for each round before round* r, $p_h = h^2(1+h-h^2)$ *for* L^r, *and the leader* ℓ^r *is honest with probability at least* p_h.

Combining the three lemmas together, Theorem 1 holds. □

The proofs of the above three lemmas have some common ingredients that are worth distilling out separately. First, Lemma 5.6 below shows several important timing properties about each round r. In particular, it shows that the honest users in our protocol act "almost simultaneously", even though they have asynchronous clocks. Second, Lemma 5.7 below shows an important counting property about each round r. Essentially, this property corresponds to the requirement that $n \geq 3t + 1$ for the Byzantine agreement in classical settings.

Lemma 5.6. *Assume Properties 1–3 hold for rounds* $0, \ldots, r-1$. *For each step* $s \geq 1$ *of round* r:

(a) *For any honest user* i, $\alpha_i^{r,1} = \alpha_i^{r,2} = \alpha_i^{r,3} \in I^r$, *and* $\beta_i^{r,s} \leq \alpha_i^{r,1} + t_s$.
(b) *If* $s \neq 2$ *then for any two honest users* i *and* i', $|\beta_i^{r,s} - \beta_{i'}^{r,s}| \leq \lambda$. *That is, all honest users finish their Step* s *within time* λ *of each other.*
(c) *If an honest user* i *has waited the maximum amount of time required by Step* s, *then by time* $\beta_i^{r,s}$, *she has received all messages sent by all honest verifiers in* $HSV^{r,s-1}$.

Proof. Property (a) follows directly from the inductive hypothesis and the description of the protocol. Indeed, player i becomes sure about B^{r-1} in the time interval I^r and starts her own Steps 1, 2, 3 of round r right away. Thus $\alpha_i^{r,1} = \alpha_i^{r,2} = \alpha_i^{r,3} \in I^r$ by definition. Moreover, the maximum total time user i would have waited from the beginning of her round r to the end of Step s is t_s by definition, and $\beta_i^{r,s} \leq \alpha_i^{r,1} + t_s$.

Next, we prove Properties (b) and (c) together, by induction on s. The initial steps are Steps 1, 2 and 3. When $s = 1$, $\beta_i^{r,1} = \alpha_i^{r,1} \in I^r$ for user i, and $\beta_{i'}^{r,1} = \alpha_{i'}^{r,1} \in I^r$ for user i'. Thus

$$|\beta_i^{r,1} - \beta_{i'}^{r,1}| \leq \lambda$$

and Property (b) holds. Property (c) does not apply to $s = 1$.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

When $s = 2$, Property (b) does not apply.[15] For Property (c), note that user i finishes Step 2 at time $\beta_i^{r,2} = \alpha_i^{r,2} + t_2 = \alpha_i^{r,2} + \lambda + \Lambda$. Since all honest users finish their Step 1 within time λ of each other, by time $\alpha_i^{r,1} + \lambda = \alpha_i^{r,2} + \lambda$, all honest potential leaders in $HSV^{r,1}$ have sent out their $(r,1)$-messages. Accordingly, by time $\alpha_i^{r,2} + 2\lambda$ user i has received all of their short messages (i.e., the head of their proposed blocks and their credentials); and by time $\alpha_i^{r,2} + \lambda + \Lambda$ user i has received all of their proposed blocks. Thus Property (c) holds.

When $s = 3$, Property (c) is the easier one to show. Because $\beta_i^{r,3} = \alpha_i^{r,3} + t_3 = \alpha_i^{r,3} + t_2 + 2\lambda$, and because all honest users start their Steps 1, 2 and 3 within the time interval I^r, by time $\alpha_i^{r,3} + \lambda + t_2$ all honest verifiers in $HSV^{r,2}$ have finished their Step 2 and sent out their $(r,2)$-messages. Thus by time $\alpha_i^{r,3} + t_2 + 2\lambda$ user i has received all of their messages, and Property (c) holds.

For Property (b) with $s = 3$, without loss of generality assume user i finishes her Step 3 before user i' finishes his: namely, $\beta_i^{r,3} \le \beta_{i'}^{r,3}$. If user i has waited time t_3 from her beginning of round r, then

$$\beta_i^{r,3} = \alpha_i^{r,3} + t_3 \le \beta_{i'}^{r,3}.$$

As they both started their Step 3 within I^r, $\alpha_{i'}^{r,3} \le \alpha_i^{r,3} + \lambda$ and by Property (a)

$$\beta_{i'}^{r,3} \le \alpha_{i'}^{r,3} + t_3 \le \alpha_i^{r,3} + t_3 + \lambda.$$

Thus

$$|\beta_i^{r,3} - \beta_{i'}^{r,3}| = \beta_{i'}^{r,3} - \beta_i^{r,3} \le \lambda.$$

If instead user i finishes her Step 3 before time $\alpha_i^{r,3} + t_3$, it must be that she has received at least t_H valid $(r,2)$-messages for the same value v. As user i has helped propagating those messages, by time at most $\beta_i^{r,3} + \lambda$ user i' has received all of them, as if their propagations were all initiated by the honest user i. Accordingly, user i' finishes no later than time $\beta_i^{r,3} + \lambda$: either because his own timer for Step 3 has run out, or because he has received t_H valid $(r,2)$-messages for the same v. Thus we again have

$$|\beta_i^{r,3} - \beta_{i'}^{r,3}| \le \lambda,$$

and Property (b) holds for $s = 3$.

We now prove Properties (b) and (c) for Step $s > 3$, assuming they hold for Step $s - 1$. Again Property (c) is easier to show. By the inductive hypothesis, all honest users finish their Step $s - 1$ no later than $\beta_i^{r,s-1} + \lambda$. As $\alpha_i^{r,s} = \beta_i^{r,s-1}$ by the definition of the protocol, by time $\alpha_i^{r,s} + \lambda$ all honest verifiers in $HSV^{r,s-1}$ have sent out their $(r, s-1)$-messages. Thus by time $\alpha_i^{r,s} + 2\lambda$ user i has received all of them. Note that 2λ is the maximum amount of waiting required by Step s. Thus Property (c) holds.

Property (b) for Step $s > 3$ is similar to that for Step 3. By the inductive hypothesis, i and i' finish their Step $s - 1$ and thus start their Step s within time λ of each other. If both have waited time 2λ in Step s, then they finish Step s within time λ of each other. Otherwise, assume user i finishes before user i' and has waited for less than 2λ time. By the definition of the protocol, it must be that user i has received t_H valid messages for the same value v or the same bit b, so that he does not need to wait the full 2λ time. For example, this happens in sub-steps (a) or (b) if $s = 4$, and in the Ending Conditions if $s > 4$. No matter which case it is, user i has helped propagating those messages and user i' will receive all of them no later than time $\beta_i^{r,s} + \lambda$. The same t_H messages allow user i' to finish Step s under the same condition as user i did, if i' has not finished it already. Thus $\beta_{i'}^{r,s} \le \beta_i^{r,s} + \lambda$ and $|\beta_i^{r,s} - \beta_{i'}^{r,s}| \le \lambda$.

In sum, Properties (b) and (c) hold. □

Lemma 5.7. *Assume Properties 1–3 hold for rounds* $0, \ldots, r-1$. *For each step* $s \ge 2$ *of round* r, *if there exists a value* v *such that at least* t_H *verifiers in* $SV^{r,s}$ *have signed* v *in Step* s, *then there does not exist another value* $v' \ne v$ *with the same length as* v, *such that at least* t_H *verifiers in* $SV^{r,s}$ *have signed* v'.

Proof. Note that the verifiers $i \in SV^{r,s}$ sign at most two things in Step s using their ephemeral secret keys $sk_i^{r,s}$: a value v_i of the same length as the output of the hash function, and also a bit $b_i \in \{0,1\}$ if $s \ge 4$. That is why in the statement of the lemma we require that v' has the same length as v: of course t_H verifiers may have signed both a hash value and a bit. The lemma applies no matter what the length of v is.

Assume for the sake of contradiction that there exist two different values v and v' of the same length, such that at least t_H verifiers in $SV^{r,s}$ have signed v using their ephemeral secret keys, and also at least t_H verifiers in $SV^{r,s}$ have

[15] Indeed, as users wait for the proposed blocks in Step 2, when they finish this step, they may be time Λ away from each other in the worst case. This is why Step 3 starts together with Steps 1 and 2 rather than after Step 2 finishes. The current design ensures that the honest users will "catch up" in Step 3 and become at most λ away.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

signed v' using their ephemeral secret keys. Some malicious verifiers may have signed both v and v'. However, at least $t_H - |MSV^{r,s}|$ honest verifiers in $HSV^{r,s}$ have signed v, and another group of at least $t_H - |MSV^{r,s}|$ honest verifiers in $HSV^{r,s}$ have signed v'. Since an honest verifier i does not sign both, and since i destroys his ephemeral secret key $sk_i^{r,s}$ before propagating his message, the Adversary cannot forge i's signature for a value that i did not sign, after learning that i is a verifier. Accordingly, the total number of verifiers in $SV^{r,s}$ is

$$|SV^{r,s}| = |HSV^{r,s}| + |MSV^{r,s}| \geq 2(t_H - |MSV^{r,s}|) + |MSV^{r,s}| = 2t_H - |MSV^{r,s}|.$$

However, since the choices of n and t_H guarantee $|HSV^{r,s}| + 2|MSV^{r,s}| < 2t_H$ with overwhelming probability, we have

$$|SV^{r,s}| = |HSV^{r,s}| + |MSV^{r,s}| < 2t_H - |MSV^{r,s}|,$$

a contradiction. Therefore Lemma 5.7 holds. □

Remark. Lemmas 5.6 and 5.7 play an important role in later analysis. At a high level, with the time invariants and the counting guarantee, we are able to deal with the permissionless, highly asynchronous and highly adversarial setting in a way that mimics what happens to the Byzantine agreement in classical settings. This is far from being sufficient though. For example, the consensus guarantees of the graded consensus protocol and the BBA protocol haven't been completely restored, as will become clear especially in the proof of the soundness lemma. As a second example, the time needed to generate a block is not the same as the number of steps needed for the BA protocol to reach an agreement. On the one hand, the maximum amount of waiting time imposed in each step of Algorand guarantees that the honest users' messages are properly pipelined. On the other hand, the fact that the waiting times are not hard constraints[16] allows honest users to finish as soon as they could, as shown in the proof of the completeness lemma. Third, Algorand continues generating one block after another, with different sets of users in charge in each step, while the Byzantine agreement is only about achieving consensus once, with a fixed set of users. Whether the Adversary is able to manipulate earlier rounds in order to "attack" a targeted round r is beyond the scope of the BA protocol and is analyzed in Section 5.6.

5.4. The completeness lemma

Lemma 5.3 *(Completeness Lemma, restated). Assume Properties 1–3 hold for rounds* $0, \ldots, r-1$. *When the leader* ℓ^r *is honest, Properties 1 and 2 hold for round* r.

Proof. The protocol is easy to analyze when the leader ℓ^r is honest, and below we essentially follow it step by step for round r.

Recall that $B^{r'}$ is the last block before B^r with a non-empty payset. If $\ell^{r'}$ was honest or if $T^r - T^{r'+1} \geq \Lambda$, then user ℓ^r has known $B^{r'}$ and all preceding blocks by time T^r. Indeed, a non-empty block $B^{r'}$ with $r' < r$ must have been signed by at least t_H verifiers in $SV^{r',2}$, and more than half of them are honest. Since the honest verifiers have helped propagating $B^{r'}$ before finishing their Step 2 of round r', $B^{r'}$ (and all non-empty blocks preceding it) reaches all honest users no latter than time Λ after round r' finishes: that is, no latter than time $T^{r'+1} + \Lambda$. The blocks between $B^{r'}$ and B^r have empty paysets by the definition of $B^{r'}$, thus ℓ^r knew them the moment she became sure about them. Accordingly, in this case ℓ^r proposes in Step 1 a block $B^r_{\ell^r}$ which has a non-empty payset. By the definition of the protocol, $B^r_{\ell^r}$ contains a maximal valid payset received by her by time $\alpha^{r,1}_{\ell^r}$.

Below we consider how and when the honest users reach agreement on B^r. By Lemma 5.6, all honest users start their first three steps of round r within the time interval I^r. Following the proof of Property (c) in this lemma, in Step 2 each honest user i has received *all* short messages from the honest verifiers in $HSV^{r,1}$ by time $\alpha_i^{r,2} + 2\lambda$, including the one from ℓ^r,[17] which consists of $(Head(B^r_{\ell^r}), \sigma^{r,1}_{\ell^r})$. By definition, $H(\sigma^{r,1}_{\ell^r})$ is the smallest among all hash values of the credentials of the potential leaders in $SV^{r,1}$. Therefore no matter which $(r,1)$-messages from the malicious verifiers in $MSV^{r,1}$ have been received by user i, i privately identifies ℓ^r as the leader of round r.

If $B^r_{\ell^r}$ has an empty payset, then by time $\alpha_i^{r,2} + 2\lambda$ each honest user i has also received the message $m^{r,1}_{\ell^r} = (B^r_{\ell^r}, esig_{\ell^r}(H(B^r_{\ell^r})), \sigma^{r,1}_{\ell^r})$. Since ℓ^r is honest, $m^{r,1}_{\ell^r}$ is valid; and user i does not need to know previous blocks to verify it, because $PAY^r_{\ell^r} = \emptyset$. Thus all honest users i finish Step 2 at time $\beta_i^{r,2} = \alpha_i^{r,2} + 2\lambda$, with all honest verifiers $i \in HSV^{r,2}$ signing and propagating $v_i = (H(B^r_{\ell^r}), \ell^r)$.

If $PAY^r_{\ell^r}$ is non-empty, then by time $\alpha_i^{r,2} + t_2$ each honest user i has received the message $m^{r,1}_{\ell^r}$. Not only so, but i has also known all previous blocks by then, as $\alpha_i^{r,2} + t_2 > \alpha_i^{r,2} + \Lambda \geq T^r + \Lambda$, and all previous blocks have been propagated before time T^r. Thus again all honest verifiers $i \in HSV^{r,2}$ sign and propagate $v_i = (H(B^r_{\ell^r}), \ell^r)$ by the end of their Step 2.

[16] Except sub-step 1 of Step 2, where a user "waits for time 2λ" instead of "waits a maximum amount of time 2λ".

[17] This is why the waiting time 2λ in sub-step 1 of Step 2 is a hard constraint rather than a maximum amount.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

When does the *last* honest user finish Step 2? Note that by time $T^r + \lambda$ all honest users have started both Steps 1 and 2, including the leader ℓ^r. Thus by time

$$\beta^{r,2} \triangleq T^r + \lambda + 2\lambda$$

they have all finished Step 2 if $PAY^r_{\ell^r}$ is empty, and this time is

$$\beta^{r,2} \triangleq T^r + \lambda + \Lambda$$

if $PAY^r_{\ell^r}$ is non-empty. In particular, although $t_2 = \lambda + \Lambda$ and an honest user i may start his Step 2 as late as time $T^r + \lambda$, he only needs to wait at most Λ time after $T^r + \lambda$, to receive the non-empty block from ℓ^r.

By the choices of n and t_H, $|HSV^{r,2}| > t_H$. Thus in Step 3, by Properties (b) and (c) of Lemma 5.6 and by Lemma 5.7, all honest users finish after receiving t_H $(r,2)$-signatures for the same $v = (H(B^r_{\ell^r}), \ell^r)$. Indeed, there does not exist another v' that has been signed by t_H verifiers in $SV^{r,2}$. Accordingly, whether or not an honest user i's timer for Step 3 has run out, i finishes Step 3 after receiving t_H signatures for v, by time

$$\beta^{r,3} \triangleq \beta^{r,2} + \lambda.$$

By definition of the protocol, all honest verifiers $i \in HSV^{r,3}$ sign and propagate $v_i = v = (H(B^r_{\ell^r}), \ell^r)$ by the end of their Step 3.

Step 4 is similar. By time $\beta^{r,4} \triangleq \beta^{r,3} + \lambda$, all honest users have received all honest $(r,3)$-messages from $HSV^{r,3}$. Since $|HSV^{r,3}| > t_H$ and all verifiers in this set have signed $v = (H(B^r_{\ell^r}), \ell^r)$, all honest users i finish Step 4 by time $\beta^{r,4}$ no matter whether their timers for this step have run out or not, and they all have $v_i = v$ and $g_i = 2$. Again by Lemma 5.7, there does not exist another v' with t_H signatures from $SV^{r,3}$. Accordingly, all honest verifiers in $HSV^{r,4}$ have signed and propagated $v_i = (H(B^r_{\ell^r}), \ell^r)$ and $b_i = 0$.

Finally, by time $\beta^{r,5} \triangleq \beta^{r,4} + \lambda$, all honest users have received all honest $(r,4)$-messages from $HSV^{r,4}$, all signing for $v = (H(B^r_{\ell^r}), \ell^r)$ and $b = 0$. Moreover, they have already received the honest leader ℓ^r's messages from Step 1. As $|HSV^{r,4}| > t_H$, Ending Condition 0 is triggered and all honest users finish round r knowing $B^r = B^r_{\ell^r}$.

Tracing back the running time of each step, if $PAY^r_{\ell^r} = \emptyset$ then $\beta^{r,5} = \beta^{r,2} + 3\lambda = T^r + 6\lambda$; otherwise $\beta^{r,5} = \beta^{r,2} + 3\lambda = T^r + 4\lambda + \Lambda$. As all honest users have known B^r by time $\beta^{r,5}$, the first honest user knows B^r no later than that, and we have $T^{r+1} \leq \beta^{r,5}$. Moreover, all honest users get to know B^r within λ time after the first honest user knows B^r, as it takes at most λ time for the short messages in $CERT^r$ seen by the first honest user to be propagated to all of them. That is, they all know B^r within the time interval I^{r+1}.

In sum, Properties 1 and 2 hold for round r when the leader ℓ^r is honest. □

Remark. When the leader ℓ^r is honest, the running time of round r essentially "charges" time λ to each step 3, 4, 5, even though the maximum waiting time could be 2λ. This is because (1) the honest leader and honest verifiers always send their messages "in time", as required by the protocol; and (2) the honest messages from a Step $s-1$ alone are sufficient to ensure that honest users finish their Step s without having to wait for the maximum amount of time, as all honest $(r, s-1)$-messages agree with each other and there are at least t_H of them.

As we will see in the proof of the soundness lemma, a malicious leader ℓ^r may cause the honest users to wait for the maximum amount of time for each step, thus the running time of round r essentially "charges" time 2λ to each step after Step 2. However, the maximum delay caused by a malicious leader is upper-bounded as in Property 3 of Theorem 1, and the Adversary can never stop the honest users from reaching agreement on what the r-th block should be, except with negligible probability.

5.5. The soundness lemma

To prove the soundness lemma, we first consider how the graded consensus protocol GC behaves in Algorand, assuming Properties 1–3 of Theorem 1 hold for rounds $0, \ldots, r-1$.

Lemma 5.8. *If there exists an honest verifier $\hat{i} \in HSV^{r,4}$ who finishes her Step 4 of round r with $g_{\hat{i}} = 2$, then*

- *$g_i \geq 1$ for all honest users i when i finishes his Step 4 of round r;*
- *there exists a value $v \neq \perp$ such that $v_i = v$ for all honest users i; and*
- *there exists a valid message $m^{r,1}_{\ell}$ from some verifier $\ell \in SV^{r,1}$ such that $v = (H(B^r_{\ell}), \ell)$.*

Proof. Since player $\hat{i}$ is honest and sets $g_{\hat{i}} = 2$, she has received at least t_H valid $(r,3)$-messages signing for the same value $v \neq \perp$, and she has set $v_{\hat{i}} = v$.

By Lemma 5.7 (with $s = 3$), there does not exist another value $v' \neq v$ such that at least t_H verifiers in $SV^{r,3}$ have signed for v'. Accordingly, if an honest user i finishes his Step 4 without waiting the whole amount 2λ, he has seen at least t_H valid $(r,3)$-messages for some v', which must be $v' = v$. Thus i sets $g_i = 2$ and $v_i = v$ as desired.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

If an honest user i finishes his Step 4 after waiting time 2λ, by Property (c) of Lemma 5.6 he has received all the honest $(r,3)$-messages from $HSV^{r,3}$. Because $|MSV^{r,3}| < t_H/2$ by the choices of n and t_H, more than $t_H/2$ of the valid $(r,3)$-messages seen by $\hat{i}$ and signing for v are from honest verifiers in $HSV^{r,3}$. Thus user i has seen more than $t_H/2$ valid $(r,3)$-messages signing for v. Accordingly, $g_i \neq 0$. If $g_i = 2$ then it must be that $v_i = v$, again by Lemma 5.7.

If $g_i = 1$, we assume for the sake of contradiction that i has also seen more than $t_H/2$ valid $(r,3)$-messages signing for a different v'. Again because $|MSV^{r,3}| < t_H/2$, at least one honest verifier $j \in HSV^{r,3}$ has signed for v and at least another $j' \in HSV^{r,3}$ has signed for v'. As j and j' are both honest, by the definition of the protocol, j has received at least t_H valid $(r,2)$-messages signing for v and j' has received at least t_H valid $(r,2)$-messages signing for v'. However, this contradicts Lemma 5.7 (with $s=2$). Thus it must be that user i sets $v_i = v$ if $g_i = 1$, as desired.

Finally, the fact that $\hat{i}$ has seen at least t_H valid $(r,3)$-messages signing for v implies at least one honest verifier $j \in HSV^{r,3}$ has signed for v. (Again, $|MSV^{r,3}| < t_H/2$, so actually more than $t_H/2$ honest verifiers in $HSV^{r,3}$ have signed for v.) By definition, j has seen at least t_H valid $(r,2)$-messages signing for v. Because $|MSV^{r,2}| < t_H/2$, at least one honest verifier $j' \in HSV^{r,2}$ has signed for v. By the definition of the protocol, j' has received a valid message $m_\ell^{r,1}$ from some potential leader $\ell \in SV^{r,1}$, and $v = (H(B_\ell^r), \ell)$. In particular, all payments in B_ℓ^r are valid. Therefore Lemma 5.8 holds. □

We now consider the protocol $BBA^\star$ in Algorand, and we distinguish two cases.

Lemma 5.9. *If all honest verifiers $i \in HSV^{r,4}$ finish their Step 4 with $g_i < 2$, then $B^r = B_\epsilon^r$, $T^{r+1} \leq T^r + 8\lambda + \Lambda$ and all honest users know B^r in the time interval I^{r+1}.*

Proof. This is the complement scenario of Lemma 5.8. In this case, by the definition of the protocol, all honest verifiers in $HSV^{r,4}$ have signed and propagated $b_i = 1$ at the end of their Step 4 as their private inputs to $BBA^\star$. As we will see, it doesn't matter what their v_i's are. Moreover, by Property (a) of Lemma 5.6, all honest users finish their Step 4 no later than time

$$\beta^{r,4} \triangleq T^r + \lambda + t_4.$$

Because $|HSV^{r,4}| > t_H$ and all verifiers in this set have signed for $b = 1$, by Lemma 5.7, less than t_H verifiers in $SV^{r,4}$ have signed for $b' = 0$. For any honest user i, in Step 5, Ending Condition 0 is not triggered, because it looks for t_H signatures from Step 4 signing for 0.[18] Ending Condition 1 does not apply here, because it starts from the first Coin-Fixed-To-1 step, Step 6. Moreover, user i does not end Step 5 with $b_i = 0$, because he would have received all honest $(r,4)$-messages from $HSV^{r,4}$ signing for $b = 1$ by the time he is done waiting, following Property (c) of Lemma 5.6. Accordingly, all honest users i finish their Step 5 with $b_i = 1$, and they all finish no later than

$$\beta^{r,5} \triangleq \beta^{r,4} + \lambda,$$

by when they have received all honest $(r,4)$-messages. Note that similar to the proof of the completeness lemma, only time λ is charged for this step because the honest verifiers all start with the same private input.

Step 6 is similar. In particular, there do not exist at least t_H signatures from $SV^{r,4}$ signing for $b' = 0$, neither from $SV^{r,5}$. But there exist $|HSV^{r,5}| > t_H$ signatures from honest verifiers in $HSV^{r,5}$ signing for $b = 1$. Accordingly, Ending Condition 1 is triggered before an honest user i is done waiting in his Step 6, and i finishes with $B^r = B_\epsilon^r$, with his $CERT^r$ containing $Head(B_\epsilon^r)$ and the set of valid $(r,5)$-messages seen by him and signing for $b = 1$. Moreover, all honest users finish their Step 6 no later than

$$\beta^{r,6} \triangleq \beta^{r,5} + \lambda.$$

By definition, $t_4 = 5\lambda + \Lambda$. Thus $T^{r+1} \leq \beta^{r,6} = T^r + 8\lambda + \Lambda$. Moreover, once the first honest user finishes round r, it takes no more than λ time for messages in his $CERT^r$ to reach all honest users, thus all of them finish within the time interval I^{r+1}. □

Note that Lemma 5.9 implies Properties 1 and 3 of Theorem 1 hold when $g_i < 2$ for all honest verifiers in $HSV^{r,4}$. The remaining case corresponds to the scenario of Lemma 5.8 and is more complex.

Lemma 5.10. *If there exists an honest verifier $\hat{i} \in HSV^{r,4}$ who finishes her Step 4 of round r with $g_{\hat{i}} = 2$, then Properties 1 and 3 of Theorem 1 hold for round r.*

[18] Essentially, round r ends at a Coin-Fixed-To-0 step s with a non-empty block because of t_H signatures signing for 0 from step $s-1$, or by "mimicking" an earlier Coin-Fixed-To-0 step s'. Here $s' = s = 5$. In later analysis, it is possible $s' < s$, because the Adversary may hide malicious verifiers' signatures in Step $s'-1$ and release them late.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Proof. By the definition of the protocol, $b_{\hat{i}} = 0$. By Lemma 5.8, there exists a valid message $m_\ell^{r,1}$ such that $\ell \in SV^{r,1}$ and $v_i = (H(B_\ell^r), \ell)$ for all honest users i after Step 4. However, the verifiers in $HSV^{r,4}$ may not have an agreement on their b_i's.

We now consider the following event E: *there exists a step $s^* \geq 5$ such that, for the first time in the BBA protocol, some player $i^* \in SV^{r,s^*}$ should enter an Ending Condition, thus should stop without propagating anything.* We use "should stop" to emphasize the fact that, if player i^* is malicious, then he may pretend that he should not stop according to the protocol and propagate messages of the Adversary's choice.

More specifically, by the construction of the protocol, when E occurs, either

($E.0$) i^* is able to collect or generate at least t_H valid messages $m_j^{r,s'-1} = (ESIG_j(0), ESIG_j(v), \sigma_j^{r,s'-1})$ for some $v \neq \perp$ and s', with $5 \leq s' \leq s^*$ and $s' - 2 \equiv 0 \mod 3$; or

($E.1$) i^* is able to collect or generate at least t_H valid messages $m_j^{r,s'-1} = (ESIG_j(1), ESIG_j(v_j), \sigma_j^{r,s'-1})$ for some s', with $6 \leq s' \leq s^*$ and $s' - 2 \equiv 1 \mod 3$.

Because all honest $(r, s'-1)$-messages are received by all honest users before they are done waiting in Step s', and because the Adversary receives every message no later than the honest users, without loss of generality we have

$s' = s^*$ *and player i^* is malicious.*

Note that we did not require the value v in $E.0$ to be the hash of a valid block: as it will become clear in the analysis, $v = (H(B_\ell^r), \ell)$ in this sub-event: that is, the value agreed upon by all honest users after Step 4.

In the remaining part of the proof, we first analyze the protocol following event E, and then show that the value of s^* is essentially distributed according to L^r. The latter implies that event E happens before Step m with overwhelming probability, given the relations of the parameters.

Consensus after event E occurs. The goal of this part is not only to show that the honest users will agree on the same block B^r, but also to analyze the time needed for the consensus to happen after event E. Let us consider Step s^* and distinguish four cases.

Case 1.0. *Event $E.0$ happens and there exists an honest verifier $i' \in HSV^{r,s^*}$ who should also stop without propagating anything.*

In this case, by the definition of $E.0$ we have $s^* - 2 \equiv 0 \mod 3$, thus Step s^* is a Coin-Fixed-To-0 step. When player i' stops, he couldn't have triggered Ending Condition 1, because this implies there is a Coin-Fixed-To-1 step $s' < s^*$ such that i' has received t_H valid $(r, s'-1)$-messages signing for 1, violating the definition of Step s^* as being the first step where event E happens. Thus player i' stops by triggering Ending Condition 0, and the t_H signatures he has received must be from Step $s^* - 1$, otherwise the definition of Step s^* is again violated.

By Lemma 5.7, the t_H $(r, s^* - 1)$-messages received by i' all sign 0 and the same v as the t_H $(r, s^* - 1)$-messages received/generated by i^*. Since $|MSV^{r,s^*-1}| < t_H$, some honest verifiers in HSV^{r,s^*-1} have signed v. As an honest verifier i only signs his value v_i from his own Step 4, we have

$$v = (H(B_\ell^r), \ell)$$

as claimed earlier. Accordingly, user i' sets $B^r = B_\ell^r$ and sets his own $CERT^r$ to be $Head(B_\ell^r)$ together with the set of valid $(r, s^* - 1)$-messages for 0 and v that he has received.

Note that the messages in $CERT^r$ of user i' will reach all honest users by time $\beta_{i'}^{r,s^*} + \lambda$, because i' has helped propagating them before his Step s^* ends. If an honest user i has not stopped by then, i will stop with $B^r = B_\ell^r$ also. We only need to show that i will not stop with a different block before receiving $CERT^r$ for B_ℓ^r.

Indeed, since all honest verifiers only sign $(H(B_\ell^r), \ell)$ together with a bit, the only possibility is that i has seen t_H signatures of 1 from a step $s' - 1$, with s' being a Coin-Fixed-To-1 step, thus stopped with the empty block. By the definition of s^*, we have $s' > s^*$. However, because there exist t_H valid $(r, s^* - 1)$-messages signing for 0, by Lemma 5.7 there do not exist t_H valid $(r, s^* - 1)$-messages signing for 1. Thus no honest verifier in HSV^{r,s^*} will end his Step s^* signing for 1: they either finish via Ending Condition 0 as user i' does, or has b_i set to 0 after waiting for time 2λ. Malicious verifiers can certainly sign 1 if the Adversary wants. However, $|MSV^{r,s^*}| < t_H/2$. Accordingly, in Step $s^* + 1$, no honest user i will finish via Ending Condition 1. Before user i is done waiting for time 2λ in Step $s^* + 1$, he has received the $CERT^r$ of user i', because

$$\beta_{i'}^{r,s^*} + \lambda \leq \beta_i^{r,s^*} + 2\lambda = \alpha_i^{r,s^*+1} + 2\lambda,$$

where the inequality is by Property (b) of Lemma 5.6. Thus the above-mentioned step s' does not exist, and all honest users i finish via Ending Condition 0 (in Step s^* or Step $s^* + 1$), with $B^r = B_\ell^r$.

Moreover, by Lemma 5.6,

$$T^{r+1} \leq \beta_{i'}^{r,s^*} \leq \alpha_{i'}^{r,1} + t_{s^*} \leq T^r + \lambda + t_{s^*},$$

and all honest users become sure about B^r in the time interval I^{r+1}.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Case 1.1. *Event E.1 happens and there exists an honest verifier $i' \in HSV^{r,s^*}$ who should also stop without propagating anything.*
In this case, we have $s^* - 2 \equiv 1 \mod 3$ and Step s^* is a Coin-Fixed-To-1 step. The analysis is similar to Case 1.0 and many details have been omitted.
As before, player i' must have received at least t_H valid (r, s^*-1)-messages of the form $(ESIG_j(1), ESIG_j(v_j), \sigma_j^{r,s^*-1})$. Again by the definition of s^*, there does not exist a Coin-Fixed-To-0 step $s' < s^*$, where at least t_H verifiers in $SV^{r,s'-1}$ have signed 0 and the same v. Thus player i' enters Ending Condition 1 in Step s^*; sets $B^r = B^r_\epsilon$; and sets his own $CERT^r$ to be the set of valid $(r, s^* - 1)$-messages received by him and signing for 1.
Symmetric to Case 1.0, all honest users finish round r via Ending Condition 1 (in Step s^* or Step $s^* + 1$) no later than time $\beta^{r,s^*}_{i'} + \lambda$, with $B^r = B^r_\epsilon$. Moreover,

$$T^{r+1} \le \beta^{r,s^*}_{i'} \le \alpha^{r,1}_{i'} + t_{s^*} \le T^r + \lambda + t_{s^*},$$

and all honest users know B^r within the time interval I^{r+1}. (As B^r is the empty block, a user knows B^r the moment he is sure about B^r.)

Case 2.0. *Event E.0 happens and there does not exist an honest verifier $i' \in HSV^{r,s^*}$ who should also stop without propagating anything.*
Similar to Case 1.0, Step s^* is a Coin-Fixed-To-0 step and the t_H $(r, s^* - 1)$-messages the Adversary is able to collect or generate are for bit 0 and $v = (H(B^r_\ell), \ell)$, where the latter is the value all honest users agreed upon by the end of Step 4. So player i^* has a valid certificate $CERT^r_{i^*}$ for B^r_ℓ. However, the malicious users may not help propagating those messages, so we cannot conclude that the honest users will receive them in time λ. In fact, $|MSV^{r,s^*-1}|$ of those messages may be from malicious $(r, s^* - 1)$-verifiers, who may not propagate their messages at all and only send them to the malicious verifiers in Step s^*.
By the definition of this case and by Lemma 5.7, all honest users have finished their Step s^* after waiting time 2λ, without seeing t_H (r, s^*-1)-messages signing for 1. Thus all honest verifiers $i \in HSV^{r,s^*}$ have signed and propagated 0 and v by the end of their Step s^*. Moreover, all honest users finish their Step s^* no later than

$$\beta^{r,s^*} \triangleq T^r + \lambda + t_{s^*}.$$

From here on, we are in the easy case, as all honest users have agreed on 0 and v by the end of Step s^*, and $|HSV^{r,s^*}| > t_H$. They cannot enter Ending Condition 0 in Step $s^* + 1$ with $s' = s^* + 1$, because the so-defined s' is not a Coin-Fixed-To-0 step. Assuming the malicious users do not release $CERT^r_{i^*}$, by Property (c) of Lemma 5.6 all honest users i have received all honest (r, s^*)-messages signing for 0, before they are done waiting in Step $s^* + 1$. So all honest verifiers in HSV^{r,s^*+1} also sign and propagate 0 and v.[19] Moreover, all honest users i finish Step $s^* + 1$ no later than

$$\beta^{r,s^*+1} \triangleq \beta^{r,s^*} + \lambda.$$

Step $s^* + 2$ is a Coin-Genuinely-Flipped step and is just similar. Assuming the malicious users do not release $CERT^r_{i^*}$, all honest verifiers in HSV^{r,s^*+2} sign and propagate 0 and v before they are done waiting: that is, they do not "flip a coin" in this case. Moreover, all honest users finish Step $s^* + 2$ no later than

$$\beta^{r,s^*+2} \triangleq \beta^{r,s^*+1} + \lambda.$$

Again note that they do not enter Ending Condition 0, because neither Step $s^* + 1$ nor Step $s^* + 2$ is a Coin-Fixed-To-0 step.
Finally, the honest users are in Step $s^* + 3$, which is another Coin-Fixed-To-0 step. All of them would have received at least t_H valid messages for 0 and v from HSV^{s^*+2} by the time they are done waiting. Thus, whether or not the Adversary releases $CERT^r_{i^*}$ does not really matter at this point. All honest users enter Ending Condition 0 in Step $s^* + 3$ (with $s' = s^* + 3$) and finishes round r with $B^r = B^r_\ell$. Moreover, they all finish no later than

$$\beta^{r,s^*+3} \triangleq \beta^{r,s^*+2} + \lambda.$$

If the Adversary releases $CERT^r_{i*}$ during the above steps, he is only helping the honest users to finisher earlier. In this case, some honest users may have their own $CERT^r$ for $B^r = B^r_\ell$ containing those $(r, s^* - 1)$-messages in $CERT^r_{i^*}$, and the others have their own $CERT^r$ containing $(r, s^* + 2)$-messages from HSV^{r,s^*+2}. In any case, from the moment the first honest user i finishes, it takes at most time λ for i's $CERT^r$ to reach all honest users. Therefore

$$T^{r+1} \le \beta^{r,s^*+3} = T^r + 4\lambda + t_{s^*},$$

[19] Note that Step $s^* + 1$ is a Coin-Fixed-To-1 step. So if an honest verifier in HSV^{r,s^*+1} has not received the t_H (r, s^*)-messages signing for 0 when he is done waiting, he would have signed 1. Thus it is important to maintain the time invariants as shown in Lemma 5.6.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

and all honest users become sure about B^r within the time interval I^{r+1}.

Case 2.1. *Event E.1 happens and there does not exist an honest verifier $i' \in HSV^{r,s^*}$ who should also stop without propagating anything.*

The analysis of this case is similar to that of Cases 1.1 and 2.0, thus many details have been omitted. In particular, Step s^* is a Coin-Fixed-To-1 step, $CERT^r_{i^*}$ consists of the t_H desired (r, s^*-1)-messages for bit 1 that the Adversary is able to collect or generate, and there do not exist t_H valid (r, s^*-1)-messages signing for 0.

Accordingly, all honest users agree on bit 1 by the end of Step s^*, $|HSV^{r,s^*}| > t_H$, and they all finish no later than $\beta^{r,s^*} \triangleq T^r + \lambda + t_{s^*}$. Similar to Case 2.0, assuming the Adversary does not release $CERT^r_{i^*}$, in 3 steps the protocol reaches Step s^*+3, which is another Coin-Fixed-To-1 step. In this step, all honest users enter Ending Condition 1 with $B^r = B^r_\epsilon$. Moreover, each of the three steps takes no more than λ time, and all honest users finish Step s^*+3 no later than $\beta^{r,s^*+3} \triangleq \beta^{r,s^*} + 3\lambda$. Thus

$$T^{r+1} \le \beta^{r,s^*+3} = T^r + 4\lambda + t_{s^*},$$

and all honest users know $B^r (= B^r_\epsilon)$ within the time interval I^{r+1}.

Combining the above four cases, we have that

- $T^{r+1} \le T^r + \lambda + t_{s^*}$ in Cases 1.0 and 1.1,
- $T^{r+1} \le T^r + 4\lambda + t_{s^*}$ in Cases 2.0 and 2.1, and
- all honest users become sure about B^r within the time interval I^{r+1}.

Since B^r is either the block B^r_ℓ whose hash all honest users have agreed upon by the end of their Step 4, or the empty block, all payments in B^r are valid.

The total time needed to reach consensus. It remains to upper-bound s^* and thus T^{r+1}, and we do so by considering how many times the Coin-Genuinely-Flipped steps are actually executed in the protocol: that is, some honest verifiers actually have flipped a coin.

In particular, arbitrarily fix a Coin-Genuinely-Flipped step s' (i.e., $7 \le s' \le m$ and $s' - 2 \equiv 2 \mod 3$), and let $\ell' \triangleq \arg\min_{j \in SV^{r,s'-1}} H(\sigma^{r,s'-1}_j)$. For now let us assume $s' < s^*$, because otherwise no honest verifier actually flips a coin in Step s', according to previous discussions.

By the definition of $SV^{r,s'-1}$, the hash value of the $(r, s'-1)$-credential of ℓ' is also the smallest among all $(r, s'-1)$-credentials of all users in PK^{r-k}. Thus ℓ' is honest with the same probability p_h as the leader ℓ^r. In the BA protocol in classical settings with $n \ge 3t+1$, ℓ' is honest with probability at least $2/3$. As we will show in Section 5.6, In Algorand, even if the Adversary tries his best to predict the output of the random oracle H and tilt the probability, player ℓ' (as well as ℓ^r) is still honest with probability at least $p_h = h^2(1+h-h^2)$. Below we consider the case when $\ell' \in HSV^{r,s'-1}$.

By Property (c) of Lemma 5.6, all honest users i have received all $(r, s'-1)$-messages from $HSV^{r,s'-1}$ after waiting time 2λ in their Step s'. If player i needs to flip a coin, then by the definition of the protocol i has indeed waited 2λ time (moreover, he wasn't able to enter Ending Conditions, neither has he seen t_H valid $(r, s'-1)$-messages for 0 or for 1). Accordingly, user i correctly identifies user ℓ' as the one with the smallest hashed credential, and sets $b_i = \mathtt{lsb}(H(\sigma^{r,s'-1}_{\ell'}, r))$.

Because $s' < s^*$, by the definition of s^*, no user (honest or malicious) is able to enter Ending Conditions in Step s'. So all honest users count the $(r, s'-1)$-messages propagated to them to decide their actions in Step s'. If there exists an honest user i' who has received at least t_H valid $(r, s'-1)$-messages signing for the same bit b by the end of his Step s', then by Lemma 5.7, no honest user would have received at least t_H valid $(r, s'-1)$-messages signing for a bit $b' \ne b$.

Since $\mathtt{lsb}(H(\sigma^{r,s'-1}_{\ell'}, r)) = b$ with probability $1/2$, all honest users reach an agreement on b with probability $1/2$ by the end of their Step s': either by setting $b_i = \mathtt{lsb}(H(\sigma^{r,s'-1}_{\ell'}, r))$ when flipping a coin, or by setting $b_i = b$ following the t_H $(r, s'-1)$-messages signing for b. Of course, if no honest user has seen t_H $(r, s'-1)$-messages signing for b, then all honest users i have flipped a coin and agreed on $b_i = \mathtt{lsb}(H(\sigma^{r,s'-1}_{\ell'}, r))$ with probability 1.

Combining the probability for $\ell' \in HSV^{r,s'-1}$, we have that the honest users reach an agreement on a bit $b \in \{0,1\}$ with probability at least $\frac{p_h}{2}$. Moreover, recall from Lemma 5.8 and the hypothesis of Lemma 5.10 that all honest users i have agreed on $v = (H(B^r_\ell), \ell)$ by the end of their own Step 4. Thus, once an agreement on b is reached by the end of Step s', we go back to the easy case: all honest verifiers in this and future steps will only sign b and v, and there are more than t_H of them. Accordingly, all honest users agree on B^r in at most 2 steps: if $b = 0$ then they all enter Ending Condition 0 in Step $s'+1$, which is a Coin-Fixed-To-0 step; and if $b = 1$ then they all enter Ending Condition 1 in Step $s'+2$, which is a Coin-Fixed-To-1 step. Furthermore, all honest users finish their Step s' no later than $\beta^{r,s'} \triangleq T^r + \lambda + t_{s'}$, and each of the next two steps take at most λ time, therefore we have

$$T^{r+1} \le \beta^{r,s'} + \lambda = T^r + 2\lambda + t_{s'}$$

if $b = 0$, and

$$T^{r+1} \le \beta^{r,s'} + 2\lambda = T^r + 3\lambda + t_{s'}$$

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

if $b = 1$. Indeed, once an agreement on b is reached by the end of Step s', no further Coin-Genuinely-Flipped step will be executed.

Accordingly, before Step s^*, the number of times the Coin-Genuinely-Flipped steps are executed is distributed according to the random variable L^r. Letting Step s' be the last Coin-Genuinely-Flipped step according to L^r, by the construction of the protocol we have

$$s' = 4 + 3L^r.$$

When should the Adversary make Step s^* (i.e., Event E) happen if he wants to delay T^{r+1} as much as possible? We can even assume the Adversary knows the realization of L^r and thus s' in advance. There are three possibilities, as follows.

First, if $s^* > s'$ then it is not useful to the Adversary, because the honest verifiers have already reached an agreement on b (and v) by the end of their Step s'. To be sure, in this case s^* would be $s' + 1$ or $s' + 2$, again depending on whether $b = 0$ or $b = 1$. However, this is just Case 1.0 or 1.1, and the resulting T^{r+1} here is actually better than in those previous cases. More precisely, because $t_{s'+1} = t_{s'} + 2\lambda$ and $t_{s'+2} = t_{s'} + 4\lambda$, when $b = 0$, we have $s^* = s' + 1$ and

$$T^{r+1} \leq T^r + 2\lambda + t_{s'} = T^r + t_{s'+1} = T^r + t_{s^*};$$

while when $b = 1$, we have $s^* = s' + 2$ and

$$T^{r+1} \leq T^r + 3\lambda + t_{s'} < T^r + t_{s'+2} = T^r + t_{s^*}.$$

Second, if $s^* < s' - 3$—that is, s^* is before the second-last Coin-Genuinely-Flipped step—then by the analysis of Cases 2.0 and 2.1, Step s' will not be executed by honest users and

$$T^{r+1} \leq T^r + 4\lambda + t_{s^*} < T^r + t_{s'}.$$

That is, the Adversary is actually making the agreement on B^r happen faster.

Third, if $s^* = s' - 2$ or $s' - 1$—that is, the Coin-Fixed-To-0 step or the Coin-Fixed-To-1 step immediately before Step s'—then by the analysis of the four cases, the honest verifiers in Step s' do not get to flip coins anymore, because they have either entered Ending Conditions or have seen at least t_H valid $(r, s' - 1)$-messages signing for the same bit b. Therefore we have

$$T^{r+1} \leq T^r + 4\lambda + t_{s^*} = T^r + 2\lambda + t_{s^*+1} \leq T^r + 2\lambda + t_{s'}.$$

In sum, no matter what s^* is, we have

$$\begin{aligned} T^{r+1} &\leq T^r + 3\lambda + t_{s'} = T^r + 3\lambda + t_{4+3L^r} \\ &= T^r + 3\lambda + [2(4 + 3L^r) - 3]\lambda + \Lambda \\ &= T^r + (6L^r + 8)\lambda + \Lambda, \end{aligned}$$

as we wanted to show. The worst case is when $b = 1$ and $s^* = s' + 2$.

Putting everything together, Lemma 5.10 holds. □

Lemma 5.4 *(Soundness Lemma, restated). Assume Properties 1–3 hold for rounds* $0, \ldots, r - 1$. *When the leader* ℓ^r *is malicious, Properties 1 and 3 hold for round* r.

Proof. Combining Lemmas 5.9 and 5.10, Lemma 5.4 holds. □

5.6. Security of the seed Q^r and probability of an honest leader

To prove the main theorem all is left to prove is Lemma 5.5: no matter what the Adversary does, the leader of each round r is honest with probability at least p_h. Note that we do not restrict the Adversary to specific ways of deviating from the protocol when trying to increase his chance of having a malicious leader. Of course one can think of several natural ways: he may try to predict the seed used for leader selection in a round, and then introduce new malicious users whose corresponding credentials have small hash values; he may try to manipulate future seeds, when the leader of some round happens to be malicious and thus has some control about the next seed; etc. Our protocol guarantees that, under these and all possible ways of deviating not mentioned here, the probability of a malicious leader is at most $1 - p_h$.

To be more precise, recall that the verifiers in round r are taken from PK^{r-k} and are chosen according to the quantity Q^{r-1}. The reason for introducing the look-back parameter k is to make sure that, back at (the beginning of) round $r - k$, when the Adversary is able to add new malicious users to PK^{r-k}, he cannot predict the quantity Q^{r-1} except with negligible probability. Note that the hash function is a random oracle and Q^{r-1} is one of its inputs when selecting verifiers for round r. Thus, when the Adversary fails in predicting Q^{r-1}, no matter how malicious users are added to PK^{r-k}, from the Adversary's point of view each one of them is still selected to be a verifier in a step of round r with the required probability p_v (or p_l for Step 1). More precisely, we have the following lemma.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Lemma 5.11. *With $k = O(\log_{1/2} F)$, for each round r, with overwhelming probability the Adversary did not query Q^{r-1} to the random oracle back at round $r-k$.*

Proof. We proceed by induction. Assume that for each round $\gamma < r$, the Adversary did not query $Q^{\gamma-1}$ to the random oracle back at round $\gamma - k$.[20] Consider the following mental game played by the Adversary at round $r-k$, trying to predict Q^{r-1}.

In Step 1 of each round $\gamma = r-k, \ldots, r-1$, given a specific $Q^{\gamma-1}$ not queried to the random oracle, by ordering the players $i \in PK^{\gamma-k}$ according to the hash values $H(SIG_i(\gamma, 1, Q^{\gamma-1}))$ increasingly, we obtain a random permutation over $PK^{\gamma-k}$. By definition, the leader ℓ^γ is the first user in the permutation and is honest with probability h. Moreover, when $PK^{\gamma-k}$ is large enough, for any integer $x \geq 1$, the probability that the first x users in the permutation are all malicious but the $(x+1)$st is honest is $(1-h)^x h$. A more precise calculation will give the exact combinatorial probabilities, but will not change the final result.

If ℓ^γ is honest, then $Q^\gamma = H(SIG_{\ell^\gamma}(Q^{\gamma-1}), \gamma)$. As the Adversary cannot forge the signature of ℓ^γ, Q^γ is distributed uniformly at random from the Adversary's point of view and, except with exponentially small probability,[21] was not queried to H at round $r-k$. Since each $Q^{\gamma+1}, Q^{\gamma+2}, \ldots, Q^{r-1}$ respectively is the output of H with $Q^\gamma, Q^{\gamma+1}, \ldots, Q^{r-2}$ as one of the inputs, they all look random to the Adversary and the Adversary could not have queried Q^{r-1} to H at round $r-k$.

Accordingly, the only case where the Adversary can predict Q^{r-1} with good probability at round $r-k$ is when all the leaders $\ell^{r-k}, \ldots, \ell^{r-1}$ are malicious. Again consider a round $\gamma \in \{r-k \ldots, r-1\}$ and the random permutation over $PK^{\gamma-k}$ induced by sorting the corresponding hash values. If for some $x \geq 2$, the first $x-1$ users in the permutation are all malicious and the x-th user is honest, then the Adversary has x possible choices for Q^γ: either of the form $H(SIG_i(Q^{\gamma-1}), \gamma)$, where i is one of the first $x-1$ malicious users, by making player i the actually leader of round γ; or $H(Q^{\gamma-1}, \gamma)$, by forcing $B^\gamma = B^\gamma_\epsilon$. Otherwise, the leader of round γ will be the first honest user in the permutation and Q^{r-1} becomes unpredictable to the Adversary.

Which of the above x options of Q^γ should the Adversary pursue? To help the Adversary answer this question, in the mental game we make him more powerful than he actually is, as follows. First of all, in reality, the Adversary cannot compute the hash of a honest user's signature, thus cannot decide, for each Q^γ, the number $x(Q^\gamma)$ of malicious users at the beginning of the random permutation in round $\gamma+1$ induced by Q^γ. In the mental game, we give him the numbers $x(Q^\gamma)$ for free. Second of all, in reality, having the first x users in the permutation all being malicious does not necessarily mean they can all be made into the leader, because the hash values of their signatures must also be less than p_l. We have ignored this constraint in the mental game, giving the Adversary even more advantages.

Without loss of generality below we focus on the quantity $\hat{Q}^\gamma$, which produces the longest sequence of malicious users at the beginning of the random permutation in round $\gamma+1$. Indeed, given a specific Q^γ, the protocol does not depend on $Q^{\gamma-1}$ anymore and the Adversary can solely focus on the new permutation in round $\gamma+1$, which has the same distribution for the number of malicious users at the beginning. Accordingly, in each round γ, the above mentioned $\hat{Q}^\gamma$ gives him the largest number of options for $Q^{\gamma+1}$ and maximizes the probability that the consecutive leaders are all malicious.

Thus winning probability of the Adversary in the mental game follows a Markov Chain from round $r-k$ to round $r-1$, with the state space being $\{0\} \cup \{x : x \geq 2\}$. State 0 represents the fact that the first user in the random permutation in the current round γ is honest, thus the Adversary fails the game for predicting Q^{r-1}; and each state $x \geq 2$ represents the fact that the first $x-1$ users in the permutation are malicious and the x-th is honest, thus the Adversary has x options for Q^γ. The transition probabilities $P(x, y)$ are as follows.

- $P(0, 0) = 1$ and $P(0, y) = 0$ for any $y \geq 2$. That is, the Adversary fails the game once the first user in the permutation becomes honest.
- $P(x, 0) = h^x$ for any $x \geq 2$. That is, with probability h^x, all the x random permutations have their first users being honest, thus the Adversary fails the game in the next round.
- For any $x \geq 2$ and $y \geq 2$, $P(x, y)$ is the probability that, among the x random permutations induced by the x options of Q^γ, the longest sequence of malicious users at the beginning of some of them is $y-1$, thus the Adversary has y options for $Q^{\gamma+1}$ in the next round. That is,

$$\begin{aligned} P(x, y) &= \left(\sum_{i=0}^{y-1} (1-h)^i h\right)^x - \left(\sum_{i=0}^{y-2} (1-h)^i h\right)^x \\ &= (1-(1-h)^y)^x - (1-(1-h)^{y-1})^x. \end{aligned}$$

Note that state 0 is the unique absorbing state in the transition matrix P, and every other state x has a positive probability of going to 0. We are interested in upper-bounding the number k of rounds needed for the Markov Chain to converge to 0 with overwhelming probability: that is, no matter which state the chain starts at, with overwhelming probability the Adversary loses the game and fails to predict Q^{r-1} at round $r-k$.

[20] As k is a small integer, without loss of generality one can assume that the first k rounds of the protocol are run under a safe environment and the inductive hypothesis holds for those rounds.

[21] That is, exponential in the length of the output of H. Note that this probability is way smaller than F.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Consider the transition matrix $P^{(2)} \triangleq P \cdot P$ after two rounds. It is easy to see that $P^{(2)}(0,0) = 1$ and $P^{(2)}(0,x) = 0$ for any $x \geq 2$. For any $x \geq 2$ and $y \geq 2$, as $P(0,y) = 0$, we have

$$P^{(2)}(x,y) = P(x,0)P(0,y) + \sum_{z \geq 2} P(x,z)P(z,y) = \sum_{z \geq 2} P(x,z)P(z,y).$$

Letting $\bar{h} \triangleq 1 - h$, we have

$$P(x,y) = (1-\bar{h}^y)^x - (1-\bar{h}^{y-1})^x$$

and

$$P^{(2)}(x,y) = \sum_{z \geq 2} [(1-\bar{h}^z)^x - (1-\bar{h}^{z-1})^x][(1-\bar{h}^y)^z - (1-\bar{h}^{y-1})^z].$$

Below we compute the limit of $\frac{P^{(2)}(x,y)}{P(x,y)}$ as h goes to 1—that is, $\bar{h}$ goes to 0. Note that $\bar{h} \in [0,1)$ and the lowest order of $\bar{h}$ in $P(x,y)$ is $\bar{h}^{y-1}$, with coefficient x. Accordingly,

$$\begin{aligned}
&\lim_{h \to 1} \frac{P^{(2)}(x,y)}{P(x,y)} = \lim_{\bar{h} \to 0} \frac{P^{(2)}(x,y)}{P(x,y)} = \lim_{\bar{h} \to 0} \frac{P^{(2)}(x,y)}{x\bar{h}^{y-1} + O(\bar{h}^y)} \\
&= \lim_{\bar{h} \to 0} \frac{\sum_{z \geq 2} [x\bar{h}^{z-1} + O(\bar{h}^z)][z\bar{h}^{y-1} + O(\bar{h}^y)]}{x\bar{h}^{y-1} + O(\bar{h}^y)} = \lim_{\bar{h} \to 0} \frac{2x\bar{h}^y + O(\bar{h}^{y+1})}{x\bar{h}^{y-1} + O(\bar{h}^y)} \\
&= \lim_{\bar{h} \to 0} \frac{2x\bar{h}^y}{x\bar{h}^{y-1}} = \lim_{\bar{h} \to 0} 2\bar{h} = 0.
\end{aligned}$$

When h is sufficiently close to 1,[22] we have

$$\frac{P^{(2)}(x,y)}{P(x,y)} \leq \frac{1}{2}$$

for any $x \geq 2$ and $y \geq 2$. By induction, for any $k > 2$, $P^{(k)} \triangleq P^k$ is such that

- $P^{(k)}(0,0) = 1$, $P^{(k)}(0,x) = 0$ for any $x \geq 2$, and
- for any $x \geq 2$ and $y \geq 2$,

$$\begin{aligned}
P^{(k)}(x,y) &= P^{(k-1)}(x,0)P(0,y) + \sum_{z \geq 2} P^{(k-1)}(x,z)P(z,y) \\
&= \sum_{z \geq 2} P^{(k-1)}(x,z)P(z,y) \\
&\leq \sum_{z \geq 2} \frac{P(x,z)}{2^{k-2}} \cdot P(z,y) = \frac{P^{(2)}(x,y)}{2^{k-2}} \leq \frac{P(x,y)}{2^{k-1}}.
\end{aligned}$$

As $P(x,y) \leq 1$, after $k \geq 1 + \log_{1/2} F$ rounds, the transition probability $P^{(k)}(x,y)$ into any state $y \geq 2$ is negligible, starting with any state $x \geq 2$. Although there are many such states y, it is easy to see that

$$\begin{aligned}
\lim_{y \to +\infty} \frac{P(x,y)}{P(x,y+1)} &= \lim_{y \to +\infty} \frac{(1-\bar{h}^y)^x - (1-\bar{h}^{y-1})^x}{(1-\bar{h}^{y+1})^x - (1-\bar{h}^y)^x} = \lim_{y \to +\infty} \frac{\bar{h}^{y-1} - \bar{h}^y}{\bar{h}^y - \bar{h}^{y+1}} \\
&= \frac{1}{\bar{h}} = \frac{1}{1-h}.
\end{aligned}$$

Therefore each row x of the transition matrix P decreases as a geometric sequence with rate $\frac{1}{1-h} > 2$ when y is large enough, and the same holds for $P^{(k)}$. Accordingly, when k is large enough but still on the order of $\log_{1/2} F$, $\sum_{y \geq 2} P^{(k)}(x,y) < F$ for any $x \geq 2$. That is, with overwhelming probability the Adversary loses the game and fails to predict Q^{r-1} at round $r-k$. For $h \in (2/3, 1]$, a more complex analysis shows that there exists a constant C slightly larger than $1/2$, such that it suffices to take $k = O(\log_C F)$. Thus Lemma 5.11 holds. □

[22] For example, $h = 80\%$ as suggested by the specific choices of parameters.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

Lemma 5.5 *(restated). Given Properties 1–3 for each round before r, $p_h = h^2(1 + h - h^2)$ for L^r, and the leader ℓ^r is honest with probability at least p_h.*

Proof. Following Lemma 5.11, the Adversary cannot predict Q^{r-1} back at round $r-k$ except with negligible probability. Note that this does not mean the probability of an honest leader is h for each round. Indeed, given Q^{r-1}, depending on how many malicious users are at the beginning of the random permutation of PK^{r-k} induced by Q^{r-1}, the Adversary may have more than one options for Q^r and thus can increase the probability of a malicious leader in round $r+1$—again we are giving him some unrealistic advantages as in Lemma 5.11, so as to simplify the analysis.

However, for each Q^{r-1} that was not queried to H by the Adversary back at round $r-k$, for any $x \geq 1$, with probability $(1-h)^{x-1}h$ the first honest user occurs at position x in the resulting random permutation of PK^{r-k}. When $x = 1$, the probability of an honest leader in round $r+1$ is indeed h; while when $x = 2$, the Adversary has two options for Q^r and the resulting probability is h^2—the probability that both permutations start with an honest user. Only by considering these two cases, we have that the probability of an honest leader in round $r+1$ is at least $p_h = h \cdot h + (1-h)h \cdot h^2 = h^2(1 + h - h^2)$ as desired.

Note that the above probability only considers the randomness in the protocol from round $r-k$ to round r. When all the randomness from round 0 to round r is taken into consideration, Q^{r-1} is even less predictable to the Adversary and the probability of an honest leader in round $r+1$ is at least p_h. Replacing $r+1$ with r and shifting everything back by one round, the leader ℓ^r is honest with probability at least p_h.

Similarly, in each Coin-Genuinely-Flipped step s, the "leader" of that step—that is, the verifier in $SV^{r,s}$ whose credential has the smallest hash value—is honest with probability at least p_h, as defined in the random variable L^r. □

Note that because $p_h < h$, the Adversary may be able to make the fraction of "malicious blocks" larger than the fraction of users he controls. For example, when $h = 0.9$, $p_h = 0.8829$.[23]

6. *Algorand* with Honest Majority of Money

We now, finally, show how to replace the Honest Majority of Users assumption with the much more meaningful Honest Majority of Money assumption. Following the HMM assumption, $h \in (2/3, 1]$ is now the fraction of total money owned by honest users in each PK^r. The basic idea is (in a proof-of-stake flavor) "to select a user $i \in PK^{r-k}$ to belong to $SV^{r,s}$ with a weight proportional to the amount of money owned by i."

6.1. The simplest implementation

There are many ways to implement the above idea. The conceptually simplest way would be to have each public key "flip a coin" for each unit of money it owns. More specifically, let n and n_l respectively be the expected number of money units selected as verifiers and as potential leaders. Let a_i^r be the amount of money owned by key i in round r and $A^r = \sum_{i \in PK^r} a_i^r$ the total amount of money in round r. Then $p_v \triangleq \frac{n}{A^{r-k}}$, and each pair (i, v) with $i \in PK^{r-k}$ and $v \in \{1, 2, \ldots, a_i^{r-k}\}$ is in $SV^{r,s}$ if and only if $.H(SIG_i(r, s, v, Q^{r-1})) \leq p_v$. Potential leaders are selected in the same way with respect to $p_l \triangleq \frac{n_l}{A^{r-k}}$. Each key i may have more than one pairs (i, v) in PL^r or $SV^{r,s}$, and each pair (i, v) has its own ephemeral key pair $(pk_{i,v}^{r,s}, sk_{i,v}^{r,s})$ for a step s of round r. The parameters n, t_H and n_l are chosen such that the conditions about PL^r, $HSV^{r,s}$ and $MSV^{r,s}$ required in Section 5 hold with overwhelming probability. With this implementation, each unit of money is treated as an individual user, thus Theorem 1 holds under the HMM assumption.

6.2. An efficient implementation

The above simple implementation would "force a rich participant in the system to sign many things in each step". Instead, we can achieve the same result while only requiring one signature from each key i in each step s.

Verifier selection. The idea is that we only care about the final distribution for the number of copies a key i has in $SV^{r,s}$, not the exact coin flips for each unit of money owned by i. Fixing a key $i \in PK^{r-k}$, we let $a = a_i^{r-k}$ to simplify the notation. For any integer $x \in \{0, 1, \ldots, a\}$, the probability that i has x copies in $SV^{r,s}$ is $p_{i,x} = \binom{a}{x} p_v^x (1-p_v)^{a-x}$. Since $\sum_{x=0}^{a} p_{i,x} = 1$, we can partition the interval $[0, 1]$ into $a+1$ consecutive sub-intervals, where the x-th interval I_x has length $p_{i,x}$. That is, $I_x = \left[\sum_{x'<x} p_{i,x'}, \sum_{x' \leq x} p_{i,x'}\right]$, with I_0 also including 0.

Let $y \triangleq .H(SIG_i(r, s, Q^{r-1}))$. When Q^{r-1} is randomly selected, y is uniformly distributed over $[0, 1]$. Let $x \in \{0, 1, \ldots, a\}$ be the unique value with $y \in I_x$. If $x = 0$ then i is not an (r, s)-verifier. Otherwise, i is considered to have x copies in $SV^{r,s}$,

[23] In a proof-of-work protocol where proposing a block is expensive and block proposers must be rewarded, the difference between 0.9 and 0.8829 implies that an Adversary capable of controlling 10% of the computation power is able to collect 11.7% of the reward. In Algorand, however, proposing a block does not require solving a complex cryptographic riddle. Thus the difference between 0.9 and 0.8829 only affects liveness. That is, a good block is proposed 88.29% of the time rather than 90% of the time.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

and the credential of i for proving this fact is $\sigma_i^{r,s} = SIG_i(r, s, Q^{r-1})$. Indeed, any user knowing $\sigma_i^{r,s}$ and a is able to compute y and find the corresponding x.

Note that the number of copies of i in $SV^{r,s}$ according to this approach has exactly the same distribution as when i has flipped a coin for each unit of money he owns.[24] Thus all the required conditions about $HSV^{r,s}$ and $MSV^{r,s}$ hold with overwhelming probability. The only change in the protocol is that, when a user j receives an (r, s)-message of i (which contains i's credential proving $x > 0$), j counts it as x messages.

Leader selection. To decide whether or not a key i is the leader of round r, the only thing we need to know is the smallest hash value

$$x_i \triangleq \min_{v\in\{1,\ldots,a\}} H(SIG_i(r, 1, v, Q^{r-1})).$$

Key i is in PL^r (more precisely, has at least one copy in PL^r) if and only if $.x_i \le p_l$. Moreover, no matter how many copies each key i has in PL^r, the leader is $\ell^r = \min_{i\in PL^r} x_i = \min_{i\in PK^{r-k}} x_i$.

Let $\{0, 1, \ldots, U\}$ be the image set of the hash function H. For each $u \le U$, the probability that $x_i = u$ is $p_{i,u} = (\frac{U-u+1}{U+1})^a - (\frac{U-u}{U+1})^a$. Since $\sum_{u=0}^{U} p_{i,u} = 1$, again we can partition the interval $[0, 1]$ into $U + 1$ consecutive sub-intervals, with the u-th interval $I_u \triangleq \left(\sum_{u'<u} p_{i,u}, \sum_{u'\le u} p_{i,u}\right]$ and I_0 also containing 0.

Let $y \triangleq .H(SIG_i(r, 1, Q^{r-1}))$ and u be the unique value with $y \in I_u$. Then, i is considered to have $x_i = u$, $i \in PL^r$ if and only if $x_i \le p_l$, and his credential for this fact is $\sigma_i^{r,1} = SIG_i(r, 1, Q^{r-1})$. Indeed, the so-defined x_i has exactly the same distribution as when i has computed his signature for each unit of money he owns and then set x_i to be the smallest hash of them.

Accordingly, the same choice of n_l still guarantees that $PL^r \neq \emptyset$ with overwhelming probability. The corresponding leader ℓ^r has the same distribution as before, thus is honest with probability h when Q^{r-1} is randomly selected. Moreover, any user j knowing $\sigma_i^{r,1}$, a and H is able to compute x_i. In Step 2 of the protocol, user j chooses his own leader ℓ_j^r to be the key with the smallest x_i among all valid $(r, 1)$-credentials received by him.

Leaders for coin flips. Although the credential $\sigma_i^{r,s}$ for $s > 1$ has the same form $SIG_i(r, s, Q^{r-1})$ as before, its meaning has changed. Thus in a Coin-Genuinely-Flipped step s, a key i who needs to flip a coin should no longer consider the smallest hashed $(r, s-1)$-credential received by him: this no longer guarantees that the selected key ℓ is honest with probability h.

Instead, at the end of each Coin-Fixed-To-1 step $s-1$, another set of "potential leaders for coin flips", $PL^{r,s-1}$, is selected similar to PL^r, except that each key $i \in PK^{r-k}$ computes $y^{r,s-1} \triangleq .H(SIG_i(r, s-1, \text{"coin"}, Q^{r-1}))$. The corresponding "coin-flip credential" for belonging to $PL^{r,s-1}$ is $\sigma_i^{r,s-1,coin} = SIG_i(r, s-1, \text{"coin"}, Q^{r-1})$, which is independent from $\sigma_i^{r,s-1}$. Each key $i \in PL^{r,s-1}$ propagates $\sigma_i^{r,s-1,coin}$ at the end of his own Step $s-1$, together with other $(r, s-1)$-messages he should propagate if he is also in $SV^{r,s-1}$.

Note that, in expectation, no more than n_l coin-flip credentials are propagated at the end of a Coin-Fixed-To-1 step. The "coin-flip leader" $\ell^{r,s-1}$ is defined to be the key with the smallest x_i in $PL^{r,s-1}$, and $\ell^{r,s-1}$ is honest with probability h when Q^{r-1} is randomly selected.

When a key i needs to flip a coin in Step s, he finds the key $\ell \in PL^{r,s-1}$ whose corresponding x_ℓ is the smallest among those received by him, and sets $b_i \triangleq \texttt{lsb}(H(\sigma_\ell^{r,s-1,coin}, r))$.

Business as usual. Having defined how the verifiers and the potential leaders are selected and how their credentials are computed in each step of a round r, the execution of a round is similar to that defined in Section 4. Following almost the same analysis as in Section 5.6, the leader ℓ^r is honest with probability at least p_h as before, and the random variable L^r is also distributed according to p_h. With this implementation, Theorem 1 holds under the HMM assumption.

7. Worst case vs. optimistic case

The time it takes to generate a block as stated in Theorem 1 is the worst-case: in particular, it holds when the malicious minority is exactly $1 - h$. In an optimistic environment, the malicious minority may be much less. For example, to be on the safe side the system may estimate that 20% of the money is controlled by malicious users, while in reality this number may be just 2%.

To speed up the protocol in good environments, while still maintain its security and worst-case performance, we add a new rule as follows. Given the original parameters h, n and t_H, let $t'_H > t_H$ be such that $|MSV^{r,s}| < t'_H - t_H$ with overwhelming probability. In any step $s \ge 3$, if a user i has received at least t'_H valid $(r, 2)$-messages signing for the same $v = (H(B_\ell^r), \ell)$, then i finishes his own round r immediately, with $B^r = B_\ell^r$ and $CERT^r$ containing those $(r, 2)$-messages.

Indeed, since $|MSV^{r,2}| < t'_H - t_H$, the existence of at least t'_H $(r, 2)$-messages for v implies that at least t_H honest verifiers in $HSV^{r,2}$ have signed for v. Thus we are in the same scenario as in the completeness lemma, and the original protocol is

[24] We thank Georgios Vlachos for suggesting this.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

guaranteed to finish with $B^r = B^r_\ell$ in Step 5. In this case, the new rule makes the shortcut and lets the honest users finish immediately in Step 3, without causing any ambiguity about what B^r should be. The resulting running time of round r in this case is at most $2\lambda + \Lambda$ when $PAY^r \neq \emptyset$, and is at most 4λ otherwise.

Note that such a short running time is guaranteed to happen if the leader ℓ^r is honest and $|HSV^{r,2}| \geq t'_H$, no matter what the Adversary does. Moreover, recall that ℓ^r is simply the first user in the random permutation defined by Q^{r-1} (via signatures and hashes). Therefore the probability of having an honest leader in round r is determined by the real honest majority, denoted by h', not the lower-bound h used by the system. In good environments where h' is close to 1, with high probability ℓ^r is honest and $|HSV^{r,2}| \geq t'_H$. Indeed, following the example parameters in Section 5 with $h = 80\%$, it suffices to take $t'_H \approx 3,800$. If the real honest majority is $h' = 98\%$, then ℓ^r is honest with probability $p_{h'} \approx 97.9\%$ and $|HSV^{r,2}| \geq t'_H$ with probability $> 94\%$. Thus the protocol proceeds very fast, with almost all blocks finishing within 3 steps and generated by honest leaders.

8. Implementing ephemeral keys in *Algorand*

As discussed, a verifier $i \in SV^{r,s}$ digitally signs his message of step s in round r relative to an ephemeral public key $pk_i^{r,s}$, using an ephemeral secrete key $sk_i^{r,s}$ that he promptly destroys after using. We thus need an efficient method to ensure that every user can verify that $pk_i^{r,s}$ is indeed the key to use to verify i's signature of $m_i^{r,s}$. We do so by a (to the best of our knowledge) new use of identity-based signature schemes [44].

At a high level, in such a scheme, a central authority A generates a public master key, PMK, and a corresponding secret master key, SMK. Given the identity, U, of a player U, A computes via SMK a secret signing key sk_U relative to the public key U, and privately gives sk_U to U. Indeed, in an identity-based signature scheme, the public key of a user U is U itself. This way, if A destroys SMK after computing the secret keys for the users he chose, and does not keep any computed secret key sk_U, then U is the only one who can digitally sign messages relative to the public key U. Anyone who knows "U's name" automatically knows U's public key, and thus can verify U's signatures (possibly using also the public master key PMK).

In our application, the authority A is user i himself, and the set of all possible users U consists of the user-round-step triples (i, r, s) in, say, $S = \{i\} \times \{r' + 1, \ldots, r' + 10^6\} \times \{1, 2, \ldots, m\}$. Here r' is a given round from which user i prepares his ephemeral keys for, say, the next 10^6 rounds. Recall that m is the upper-bound for the number of steps within a round. This way, $pk_i^{r,s} \triangleq (i, r, s)$ and every user seeing i's signature $SIG_{pk_i^{r,s}}(v)$ for some value v can immediately verify it, for the first million rounds r following r'.

In other words, user i first generates PMK and SMK, and then uses SMK to privately produce and store the secret keys $sk_i^{r,s}$ for each triple $(i, r, s) \in S$. Having done so, he destroys SMK and publicizes that PMK is his master public key for any round $r \in \{r' + 1, \ldots, r' + 10^6\}$. If i determines that he is not in $SV^{r,s}$, then he may leave $sk_i^{r,s}$ alone (as the protocol does not require that he authenticates any message in Step s of round r). Else, i first uses $sk_i^{r,s}$ to digitally sign his message in Step s of round r, and then destroys $sk_i^{r,s}$.

Note that user i can publicize his first public master key PMK when he first enters the system. That is, the same payment transaction $\mathcal{P}$ that brings i into the system (at round r' or at a round close to r') may also specify, as auxiliary information and at i's request, that i's public master key for any round $r \in \{r' + 1, \ldots, r' + 10^6\}$ is PMK. In particular, $\mathcal{P}$ may include a triple of the form $(PMK, r', 10^6)$.

Also note that, assuming a round takes a minute, the stash of ephemeral keys so produced will last i for almost two years. At the same time, these ephemeral secret keys will not take i too long to produce. Using an elliptic-curve based system with 32-Byte keys, each secret key is computed in a few microseconds. If $m = 180$ as in our example choices of parameters, then all 180 M secret keys can be computed in less than one hour.

When the current round is getting close to $r' + 10^6$, to handle the next million rounds, i generates a new master-key pair (PMK', SMK') and computes the next stash of ephemeral secret keys using SMK'. Then he publicizes PMK' by—for example—having $SIG_i(PMK', r' + 10^6, 10^6)$ enter a new block, either as a separate "transaction" or as auxiliary information in a payment transaction. By so doing, i informs everyone that he/she should use PMK' when verifying i's ephemeral signatures in the next million rounds. And so on.

Following this basic approach, other ways for implementing ephemeral keys without using identity-based signatures are certainly possible. For instance, via Merkle trees [31].[25]

9. Handling offline honest users

An honest user follows all his prescribed instructions, which include being online and running the protocol: that is, continual participation. This is not a major burden in Algorand, since the computation and bandwidth required from an

[25] In this method, user i generates a public-secret key pair $(pk_i^{r,s}, sk_i^{r,s})$ for each round-step pair (r, s) in, say, $\{r' + 1, \ldots, r' + 10^6\} \times \{1, \ldots, m\}$. He orders the public keys in a canonical way, stores the jth public key in the jth leaf of a Merkle tree, and computes the root value R_i, which he publicizes. When he wants to sign a message relative to key $pk_i^{r,s}$, i not only provides the actual signature, but also the authenticating path for $pk_i^{r,s}$ relative to R_i. Note that this authenticating path also proves $pk_i^{r,s}$ is stored in the jth leaf. The rest of the details can be easily filled.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001

honest user are quite modest. However, Algorand can be easily modified so as to work in alternative models where honest users are allowed to go offline in great numbers.

Before discussing such a model, let us first recall from Section 7 that if the percentage of honest users were 95%, Algorand could still be run with all parameters set according to, say, $h = 80\%$. In this case, Algorand would continue to work even if as many as half of the honest users chose to go offline (which is a major case of "absenteeism"). Indeed, at least 80% of the online users would still be honest. One way for handling this case is to essentially "sense" the number of active users and adjust the parameters.

From continual participation to lazy honesty. Note that the Continual Participation requirement ensures a crucial property: namely, the underlying BA protocol has a proper honest majority. We now explain how *lazy honesty* provides an alternative and attractive way to satisfy this property.

More specifically, a user i is *lazy-but-honest* if (1) he follows all his prescribed instructions when he is asked to participate to the protocol, and (2) he is asked to participate to the protocol only rarely—e.g., once a week—with suitable advance notice.[26]

To allow Algorand to work with such users, it suffices to "choose the verifiers of the current round from users already in the system in a much earlier round." Indeed, recall that the verifiers for a round r are chosen from users in round $r-k$, and the selections are made based on the quantity Q^{r-1}. Assuming a round takes roughly 5 minutes on average, a week has roughly 2,000 rounds. Assume at some point of time a user i wishes to plan his time and know whether he is going to be a verifier in the coming week. To facilitate i's decision, the protocol now chooses the verifiers for a round r from users in round $r-k-2,000$, and the selections are based on $Q^{r-2,001}$.

Note that at a round r', user i already knows the values $Q^{r'-2,000}, \ldots, Q^{r'-1}$, as they are part of the blockchain. Moreover, for each $r \in \{r'+1, \ldots, r'+2,000\}$, i is a verifier in a step s of round r if and only if

$$.H\left(SIG_i\left(r, s, Q^{r-2,001}\right)\right) \leq p_v.$$

Thus, at round r' user i is able to check whether he is going to be called to act as a verifier (or a potential leader) in the next 2,000 rounds. If computing a digital signature takes a millisecond, then the entire computation takes him about 1 minute. If he is not selected as a verifier in any of these rounds, then he can go offline with an "honest conscience": had he continuously participated, he would have essentially taken 0 steps in the next 2,000 rounds anyway. If instead, he is selected to be a verifier in one of these rounds, then he readies himself (e.g., by obtaining all the information necessary) to act as an honest verifier at the proper round.

By so acting, a lazy-but-honest user i only misses participating to the propagation of messages; but message propagation is typically robust. Moreover, the payers and the payees of recently propagated payments are expected to be online to watch what happens to their payments, and will participate to message propagation if they are honest.

10. Forks

Having reduced the probability of forks to 10^{-18}, it is practically unnecessary to handle them in the remote chance that they occur. Algorand, however, can also employ various fork resolution procedures, with or without proof of work. We shall discuss alternative approaches for fork resolution in a forthcoming paper.

Acknowledgements

We would like to first acknowledge Sergey Gorbunov, coauthor of the cited Democoin system.

Most sincere thanks go to Maurice Herlihy, for many enlightening discussions, for pointing out that pipelining will improve Algorand's throughput performance, and for greatly improving the exposition of an earlier version of this paper. Many thanks to Sergio Rajsbaum, for his comments on an earlier version of this paper.

Silvio Micali would like to personally thank Ron Rivest for innumerable discussions and guidance in cryptographic research over more than 3 decades, for coauthoring the micropayment system [35] that has inspired one of the verifier selection mechanisms of Algorand.

We hope to bring this technology to the next level. Meanwhile the travel and companionship are great fun, for which we are very grateful.

References

[1] M. Ben-Or, Another advantage of free choice: completely asynchronous agreement protocols, in: 2nd Symposium on Principles of Distributed Computing, PODC, 1983, pp. 27–30.

[2] BitShares, https://bitshares.org/.

[26] And potentially receiving significant rewards when he participates. The reward mechanism is actually non-trivial and will appear in a forthcoming paper.

[3] BlackCoin, https://blackcoin.co/.
[4] S. Bubna, Bitcoin mining now consumes as much electricity as Iceland, Frontera.net, October, 2017.
[5] M. Castro, B. Liskov, Practical Byzantine fault tolerance, in: 3rd Symposium on Operating Systems Design and Implementation, OSDI, 1999, pp. 173–186.
[6] D. Chaum, Random-sample elections, https://chaum.com/publications/Random-SampleElections.pdf, 2012.
[7] J. Chen, S. Micali, Algorand, Technical report, https://arxiv.org/abs/1607.01341v9, 2017.
[8] B. Chor, C. Dwork, Randomization in Byzantine agreement, in: S. Micali (Ed.), Advances in Computing Research 5: Randomness and Computation, JAI Press, 1989, pp. 443–497.
[9] J. Clark, K. Whitbourne, How much actual money is there in the world? HowStuffWorks.com, September, 2009.
[10] B. David, P. Gaži, A. Kiayias, A. Russell, Ouroboros praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain, in: EUROCRYPT, 2018, pp. 66–98, in press.
[11] C. Decker, R. Wattenhofer, Information propagation in the Bitcoin network, in: 13th International Conference on Peer-to-Peer Computing, P2P, 2013.
[12] D. Dolev, The Byzantine generals strike again, J. Algorithms 3 (1) (1982) 14–30.
[13] D. Dolev, H.R. Strong, Authenticated algorithms for Byzantine agreement, SIAM J. Comput. 12 (4) (1983) 656–666.
[14] J.R. Douceur, The Sybil attack, in: 1st International Workshop on Peer-to-Peer Systems, IPTPS, 2002, pp. 251–260.
[15] C. Dwork, M. Naor, Pricing via processing or combatting junk mail, in: CRYPTO, 1992, pp. 139–147.
[16] EOS, https://eos.io/.
[17] Ethereum, https://www.ethereum.org/.
[18] P. Feldman, S. Micali, An optimal probabilistic algorithm for synchronous Byzantine agreement, SIAM J. Comput. 26 (4) (1997) 873–933 (Preliminary version in STOC'88).
[19] M. Fischer, The consensus problem in unreliable distributed systems (a brief survey), in: International Conference on Foundations of Computation Theory, FCT, 1983, pp. 127–140.
[20] M. Fischer, N. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, J. ACM 32 (2) (1985) 374–382.
[21] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, N. Zeldovich, Algorand: scaling Byzantine agreements for cryptocurrencies, in: 26th ACM Symposium on Operating Systems Principles, SOSP, October 2017, pp. 51–68.
[22] O. Goldreich, Foundations of Cryptography: Volume 1, Basic Tools, Cambridge University Press, 2007.
[23] S. Goldwasser, S. Micali, R. Rivest, A digital signature scheme secure against adaptive chosen-message attack, SIAM J. Comput. 17 (2) (1988) 281–308.
[24] S. Gorbunov, S. Micali, Democoin: a publicly verifiable and jointly serviced cryptocurrency, https://eprint.iacr.org/2015/521, May 2015.
[25] J. Katz, C-Y. Koo, On expected constant-round protocols for Byzantine agreement, J. Comput. System Sci. 75 (2) (2009) 91–112.
[26] A. Kiayias, A. Russell, B. David, R. Oliynykov, Ouroburos: a provably secure proof-of-stake blockchain protocol, in: CRYPTO, 2017, pp. 357–388.
[27] S. King, S. Nadal PPCoin, Peer-to-peer crypto-currency with proof-of-stake White paper, https://peercoin.net/assets/paper/peercoin-paper.pdf, 2012.
[28] V. King, J. Saia, Byzantine agreement in expected polynomial time, J. ACM 63 (2) (2016) 13.
[29] Y. Lewenberg, Y. Sompolinsky, A. Zohar, Inclusive block chain protocols, in: International Conference on Financial Cryptography and Data Security, FC, 2015, pp. 528–547.
[30] N. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers, 1996.
[31] R. Merkle, A digital signature based on a conventional encryption function, in: CRYPTO, 1987, pp. 369–378.
[32] S. Micali, Algorand: the efficient public ledger, https://arxiv.org/abs/1607.01341, July 2016.
[33] S. Micali, Fast and furious Byzantine agreement, in: 8th Innovation in Theoretical Computer Science, ITCS, January 2017. Single-page abstract. Full version available at https://people.csail.mit.edu/silvio/SelectedScientificPapers/DistributedComputation/, with title "Byzantyne Agreement, Made Trivial".
[34] S. Micali, M. Rabin, S. Vadhan, Verifiable random functions, in: 40th Foundations of Computer Science, FOCS, 1999, pp. 120–130.
[35] S. Micali, R.L. Rivest, Micropayments revisited, in: CT-RSA, 2002, pp. 149–163.
[36] S. Nakamoto Bitcoin, A peer-to-peer electronic cash system, White paper, http://www.bitcoin.org/bitcoin.pdf, 2008.
[37] R. Pass, L. Seeman, a. shelat, Analysis of the blockchain protocol in asynchronous networks, in: EUROCRYPT, 2017, pp. 643–673.
[38] R. Pass, E. Shi, FruitChain: a fair blockchain, in: Symposium on Principles of Distributed Computing, PODC, 2017, pp. 315–324.
[39] R. Pass, E. Shi, The sleepy model of consensus, in: ASIACRYPT, 2017, pp. 380–409.
[40] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, J. ACM 27 (2) (1980) 228–234.
[41] Proof of stake instead of proof of work, Bitcoin Forum, https://bitcointalk.org/index.php?topic=27787.0, 2011.
[42] V. Pureswaran, S. Panikkar, S. Nair, P. Brody, Empowering the edge: practical insights on a decentralized Internet of things, in: IBM Institute for Business Value Executive Report, April 2015.
[43] M. Rabin, Randomized Byzantine generals, in: 24th Foundations of Computer Science, FOCS, 1983, pp. 403–409.
[44] A. Shamir, Identity-based cryptosystems and signature schemes, in: CRYPTO, 1984, pp. 47–53.
[45] Sortition, Wikipedia, https://en.wikipedia.org/wiki/Sortition.
[46] Tendermint, https://tendermint.com/.
[47] R. Turpin, B. Coan, Extending binary Byzantine agreement to multivalued Byzantine agreement, Inform. Process. Lett. 18 (2) (1984) 73–76.

Please cite this article in press as: J. Chen, S. Micali, Algorand: A secure and efficient distributed ledger, Theoret. Comput. Sci. (2019), https://doi.org/10.1016/j.tcs.2019.02.001