ALGORAND*

Jing Chen
Computer Science Department
Stony Brook University
Stony Brook, NY 11794, USA
jingchen@cs.stonybrook.edu

Silvio Micali
CSAIL
MIT
Cambridge, MA 02139, USA
silvio@csail.mit.edu

Abstract

A public ledger is a tamperproof sequence of data that can be read and augmented by everyone. Public ledgers have innumerable and compelling uses. They can secure, in plain sight, all kinds of transactions —such as titles, sales, and payments— in the exact order in which they occur. Public ledgers not only curb corruption, but also enable very sophisticated applications —such as cryptocurrencies and smart contracts. They stand to revolutionize the way a democratic society operates. As currently implemented, however, they scale poorly and cannot achieve their potential.

Algorand is a truly democratic and efficient way to implement a public ledger. Unlike prior implementations based on proof of work, it requires a negligible amount of computation, and generates a transaction history that will not "fork" with overwhelmingly high probability.

Algorand is based on (a novel and super fast) message-passing Byzantine agreement.

For concreteness, we shall describe Algorand only as a money platform.

1 Introduction

Money is becoming increasingly virtual. It has been estimated that about 80% of United States dollars today only exist as ledger entries [5]. Other financial instruments are following suit.

In an ideal world, in which we could count on a universally trusted central entity, immune to all possible cyber attacks, money and other financial transactions could be solely electronic. Unfortunately, we do not live in such a world. Accordingly, decentralized cryptocurrencies, such as Bitcoin [29], and "smart contract" systems, such as Ethereum, have been proposed [4]. At the heart of these systems is a shared *ledger* that reliably records a sequence of transactions,

*This is the more formal (and asynchronous) version of the ArXiv paper by the second author [24], a paper itself based on that of Gorbunov and Micali [18]. 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

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 [29], financial applications [4], and the Internet of Things [3]. Several techniques to manage blockchain-based ledgers have been proposed: *proof of work* [29], *proof of stake* [2], *practical Byzantine fault-tolerance* [8], or some combination.

Currently, however, ledgers can be inefficient to manage. For example, Bitcoin's *proof-of-work* approach (based on the original concept of [14]) requires a vast amount of computation, is wasteful and scales poorly [1]. In addition, it *de facto* concentrates power in very few hands.

We therefore wish to put forward a new method to implement a public 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 use algorithmic randomness to select, based on the ledger constructed so far, a set of *verifiers* who are in charge of constructing the next block of valid transactions. Naturally, we ensure that such selections are provably immune from manipulations and unpredictable until the last minute, but also that they ultimately are universally clear.

Algorand's approach is quite democratic, in the sense that neither in principle nor *de facto* it creates different classes of users (as "miners" and "ordinary users" in Bitcoin). In Algorand "all power resides with the set of all users".

One notable property of Algorand is that its transaction history may fork only with very small probability (e.g., one in a trillion, that is, or even 10^{-18}). Algorand can also address some legal and political concerns.

The Algorand approach applies to blockchains and, more generally, to any method of generating a tamperproof sequence of blocks. We actually put forward a new method —alternative to, and more efficient than, blockchains— that may be of independent interest.

1.1 Bitcoin's Assumption and Technical Problems

Bitcoin is a very ingenious system and has inspired a great amount of subsequent research. Yet, it is also problematic. Let us summarize its underlying assumption and technical problems —which are actually shared by essentially all cryptocurrencies that, like Bitcoin, are based on *proof-of-work*.

For this summary, it suffices to recall that, in Bitcoin, a user may own multiple public keys of a digital signature scheme, that money is associated with public keys, and that a payment is a digital signature that transfers some amount of money from one public key to another. Essentially, Bitcoin organizes all processed payments in a chain of blocks, $B_1, B_2, \ldots$, each consisting of multiple payments, such that, all payments of B_1, taken in any order, followed by those of B_2, in any order, etc., constitute a sequence of valid payments. Each block is generated, on average, every 10 minutes.

This sequence of blocks is a *chain*, because it is structured so as to ensure that any change, even in a single block, percolates into all subsequent blocks, making it easier to spot any alteration of the payment history. (As we shall see, this is achieved by including in each block a *cryptographic hash* of the previous one.) Such block structure is referred to as a *blockchain*.

Assumption: Honest Majority of Computational Power Bitcoin assumes that no malicious entity (nor a coalition of coordinated malicious entities) controls the majority of the computational power devoted to block generation. Such an entity, in fact, would be able to modify the blockchain,

and thus re-write the payment history, as it pleases. In particular, it could make a payment $\wp$, obtain the benefits paid for, and then "erase" any trace of $\wp$.

Technical Problem 1: Computational Waste Bitcoin's proof-of-work approach to block generation requires an extraordinary amount of computation. Currently, with just a few hundred thousands public keys in the system, the top 500 most powerful supercomputers can only muster a mere 12.8% percent of the total computational power required from the Bitcoin players. This amount of computation would greatly increase, should significantly more users join the system.

Technical Problem 2: Concentration of Power Today, due to the exorbitant amount of computation required, a user, trying to generate a new block using an ordinary desktop (let alone a cell phone), expects to lose money. Indeed, for computing a new block with an ordinary computer, the expected cost of the necessary electricity to power the computation exceeds the expected reward. Only using *pools* of specially built computers (that do nothing other than "mine new blocks"), one might expect to make a profit by generating new blocks. Accordingly, today there are, *de facto*, two disjoint classes of users: ordinary users, who only make payments, and specialized mining pools, that only search for new blocks.

It should therefore not be a surprise that, as of recently, the total computing power for block generation lies within just five pools. In such conditions, the assumption that a majority of the computational power is honest becomes less credible.

Technical Problem 3: Ambiguity In Bitcoin, the blockchain is not necessarily 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 $B_1, \ldots, B_k, B''_{k+1}, B''_{k+2}, B''_{k+3}$ according another user. Only after several blocks have been added to the chain, can one be reasonably sure that the first $k+3$ blocks will be the same for all users. Thus, one cannot rely right away on the payments contained in the last block of the chain. It is more prudent to wait and see whether the block becomes sufficiently deep in the blockchain and thus sufficiently stable.

Separately, *law-enforcement* and *monetary-policy* concerns have also been raised about Bitcoin.[1]

1.2 Algorand, in a Nutshell

Setting Algorand works in a very tough setting. Briefly,

(a) *Permissionless and Permissioned Environments.* Algorand works efficiently and securely even in a totally permissionless environment, where arbitrarily many users are allowed to join the system at any time, without any vetting or permission of any kind. Of course, Algorand works even better in a *permissioned* environment.

[1]The (pseudo) anonymity offered by Bitcoin payments may be misused for money laundering and/or the financing of criminal individuals or terrorist organizations. Traditional banknotes or gold bars, that in principle offer perfect anonymity, should pose the same challenge, but the physicality of these currencies substantially slows down money transfers, so as to permit some degree of monitoring by law-enforcement agencies.

The ability to "print money" is one of the very basic powers of a nation state. In principle, therefore, the massive adoption of an independently floating currency may curtail this power. Currently, however, Bitcoin is far from being a threat to governmental monetary policies, and, due to its scalability problems, may never be.

(b) *Very Adversarial Environments.* Algorand withstands a very powerful Adversary, who can

(1) *instantaneously* corrupt *any user* he wants, at *any time* he wants, provided that, in a permissionless environment, 2/3 of the money in the system belongs to honest user. (In a permissioned environment, irrespective of money, it suffices that 2/3 of the users are honest.)

(2) *totally control and perfectly coordinate* all corrupted users; and

(3) *schedule the delivery of all messages,* provided that each message m sent by a honest user reaches 95% of the honest users within a time λ_m, which solely depends on the size of m.

Main Properties Despite the presence of our powerful adversary, in Algorand

- *The amount of computation required is minimal.* Essentially, no matter how many users are present in the system, each of fifteen hundred users must perform at most a few seconds of computation.

- *A New Block is Generated in less than 10 minutes, and will* de facto *never leave the blockchain.* For instance, in expectation, the time to generate a block in the first embodiment is less than $\Lambda + 12.4\lambda$, where Λ is the time necessary to propagate a block, in a peer-to-peer gossip fashion, *no matter what block size one may choose*, and λ is the time to propagate 1,500 200B-long messages. (Since in a truly decentralized system, Λ essentially is an intrinsic latency, in Algorand the limiting factor in block generation is network speed.) *The second embodiment has actually been tested experimentally (by ?), indicating that a block is generated in less than 40 seconds.*

  In addition, Algorand's blockchain *may fork only with negligible probability* (i.e., less than one in a trillion), and thus users can relay on the payments contained in a new block as soon as the block appears.

- *All power resides with the users themselves.* Algorand is a truy distributed system. In particular, there are no exogenous entities (as the "miners" in Bitcoin), who can control which transactions are recognized.

Algorand's Techniques.

1. A New and Fast Byzantine Agreement Protocol. Algorand generates a new block via a new cryptographic, *message-passing*, binary Byzantine agreement (BA) protocol, $BA^\star$. Protocol $BA^\star$ not only satisfies some additional properties (that we shall soon discuss), but is also very fast. Roughly said, its binary-input version consists of a 3-step loop, in which a player i sends a *single* message m_i to all other players. Executed in a complete and synchronous network, with more than 2/3 of the players being honest, with probability $> 1/3$, after each loop the protocol ends in agreement. (We stress that protocol $BA^\star$ satisfies the original definition of Byzantine agreement of Pease, Shostak, and Lamport [31], without any weakenings.)

Algorand leverages this binary BA protocol to reach agreement, in our different communication model, on each new block. The agreed upon block is then *certified*, via a prescribed number of digital signature of the proper verifiers, and propagated through the network.

2. Cryptographic Sortition. Although very fast, protocol $BA^\star$ would benefit from further speed when played by millions of users. Accordingly, Algorand chooses the players of $BA^\star$ to be

a much smaller subset of the set of all users. To avoid a different kind of concentration-of-power problem, each new block B^r will be constructed and agreed upon, via a new execution of $BA^\star$, by a separate set of *selected verifiers*, SV^r. In principle, selecting such a set might be as hard as selecting B^r directly. We traverse this potential problem by an approach that we term, embracing the insightful suggestion of Maurice Herlihy, *cryptographic sortition*. Sortition is the practice of selecting officials at random from a large set of eligible individuals [6]. (Sortition was practiced across centuries: for instance, by the republics of Athens, Florence, and Venice. In modern judicial systems, random selection is often used to choose juries. Random sampling has also been recently advocated for elections by David Chaum [9].) In a decentralized system, of course, choosing the random coins necessary to randomly select the members of each verifier set SV^r is problematic. We thus resort to cryptography in order to select each verifier set, from the population of all users, in a way that is guaranteed to be automatic (i.e., requiring no message exchange) and random. In essence, we use a cryptographic function to automatically determine, from the previous block B^{r-1}, a user, the *leader*, in charge of proposing the new block B^r, and the verifier set SV^r, in charge to reach agreement on the block proposed by the leader. Since malicious users can affect the composition of B^{r-1} (e.g., by choosing some of its payments), we specially construct and use additional inputs so as to prove that the leader for the rth block and the verifier set SV^r are indeed randomly chosen.

3. The Quantity (Seed) Q^r. We use the the last block B^{r-1} in the blockchain in order to automatically determine the next verifier set and leader in charge of constructing the new block B^r. The challenge with this approach is that, by just choosing a slightly different payment in the previous round, our powerful Adversary gains a tremendous control over the next leader. Even if he only controlled only 1/1000 of the players/money in the system, he could ensure that all leaders are malicious. (See the Intuition Section 4.1.) This challenge is central to all proof-of-stake approaches, and, to the best of our knowledge, it has not, up to now, been satisfactorily solved.

To meet this challenge, we purposely construct, and continually update, a separate and carefully defined quantity, Q^r, which *provably* is, not only unpredictable, but also not influentiable, by our powerful Adversary. We may refer to Q^r as the rth *seed*, as it is from Q^r that Algorand selects, via secret cryptographic sortition, all the users that will play a special role in the generation of the rth block.

4. Secret Crytographic Sortition and Secret Credentials. Randomly and unambiguously using the current last block, B^{r-1}, in order to choose the verifier set and the leader in charge of constructing the new block, B^r, is not enough. Since B^{r-1} must be known before generating B^r, the last non-influentiable quantity Q^{r-1} contained in B^{r-1} must be known too. Accordingly, so are the verifiers and the leader in charge to compute the block B^r. Thus, our powerful Adversary might immediately corrupt *all of them*, before they engage in any discussion about B^r, so as to get full control over the block they certify.

To prevent this problem, leaders (and actually verifiers too) *secretly* learn of their role, but can compute a proper *credential*, capable of proving to everyone that indeed have that role. When a user privately realizes that he is the leader for the next block, first he secretly assembles his own proposed new block, and then disseminates it (so that can be certified) together with his own credential. This way, though the Adversary will immediately realize who the leader of the next block is, and although he can corrupt him right away, it will be too late for the Adversary to influence the choice of a new block. Indeed, he cannot "call back" the leader's message no more

than a powerful government can put back into the bottle a message virally spread by WikiLeaks.

As we shall see, we cannot guarantee leader uniqueness, nor that everyone is sure who the leader is, including the leader himself! But, in Algorand, unambiguous progress will be guaranteed.

5. Player Replaceability. After he proposes a new block, the leader might as well "die" (or be corrupted by the Adversary), because his job is done. But, for the verifiers in SV^r, things are less simple. Indeed, being in charge of certifying the new block B^r with sufficiently many signatures, they must first run Byzantine agreement on the block proposed by the leader. The problem is that, no matter how efficient it is, $BA^\star$ requires *multiple* steps and the honesty of $> 2/3$ of its players. This is a problem, because, for efficiency reasons, the player set of $BA^\star$ consists the small set SV^r randomly selected among the set of all users. Thus, our powerful Adversary, although unable to corrupt $1/3$ of *all the users*, can certainly corrupt *all members of* SV^r!

Fortunately we'll prove that protocol $BA^\star$, executed by propagating messages in a peer-to-peer fashion, is *player-replaceable*. This novel requirement means that the protocol correctly and efficiently reaches consensus even if each of its step is executed by a totally new, and randomly and independently selected, set of players. Thus, with millions of users, each small set of players associated to a step of $BA^\star$ most probably has empty intersection with the next set.

In addition, the sets of players of different steps of $BA^\star$ will probably have totally different *cardinalities*. Furthermore, the members of each set do not know who the next set of players will be, and do not secretly pass any internal state.

The replaceable-player property is actually crucial to defeat the dynamic and very powerful Adversary we envisage. We believe that replaceable-player protocols will prove crucial in lots of contexts and applications. In particular, they will be crucial to execute securely small sub-protocols embedded in a larger universe of players with a dynamic adversary, who, being able to corrupt even a small fraction of the total players, has no difficulty in corrupting all the players in the smaller sub-protocol.

An Additional Property/Technique: Lazy Honesty A honest user follows his prescribed instructions, which include being online and run the protocol. Since, Algorand has only modest computation and communication requirement, being online and running the protocol "in the background" is not a major sacrifice. Of course, a few "absences" among honest players, as those due to sudden loss of connectivity or the need of rebooting, are automatically tolerated (because we can always consider such few players to be temporarily malicious). Let us point out, however, that Algorand can be simply adapted so as to work in a new model, in which honest users to be offline most of the time. Our new model can be informally introduced as follows.

> *Lazy Honesty.* Roughly speaking, 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, and with a suitable advance notice.

With such a relaxed notion of honesty, we may be even more confident that honest people will be at hand when we need them, and Algorand guarantee that, when this is the case,

The system operates securely even if, at a given point in time,
the majority of the participating players are malicious.

1.3 Closely Related work

Proof-of-work approaches (like the cited [29] and [4]) are quite orthogonal to our ours. So are the approaches based on message-passing Byzantine agreement or practical Byzantine fault tolerance (like the cited [8]). Indeed, these protocols cannot be run among the set of all users and cannot, in our model, be restricted to a suitably small set of users. In fact, our powerful adversary my immediately corrupt all the users involved in a small set charged to actually running a BA protocol.

Our approach could be considered related to proof of stake [2], in the sense that users' "power" in block building is proportional to *the money they own in the system* (as opposed to —say— to the money they have put in "escrow").

The paper closest to ours is the Sleepy Consensus Model of Pass and Shi [30]. To avoid the heavy computation required in the proof-of-work approach, their paper relies upon (and kindly credits) Algorand's secret cryptographic sortition. With this crucial aspect in common, several significant differences exist between our papers. In particular,

(1) *Their setting is only permissioned.* By contrast, Algorand is also a permissionless system.

(2) *They use a Nakamoto-style protocol, and thus their blockchain forks frequently.* Although dispensing with proof-of-work, in their protocol a secretly selected leader is asked to elongate the longest valid (in a richer sense) blockchain. Thus, forks are unavoidable and one has to wait that the block is sufficiently "deep" in the chain. Indeed, to achieve their goals with an adversary capable of adaptive corruptions, they require a block to be $poly(N)$ deep, where N represents the total number of users in the system. Notice that, even assuming that a block could be produced in a minute, if there were $N = 1M$ users, then one would have to wait for about 2M years for a block to become N^2-deep, and for about 2 years for a block to become N-deep. By contrast, Algorand's blockchain forks only with negligible probability, even though the Adversary corrupt users immediately and adaptively, and its new blocks can immediately be relied upon.

(3) *They do not handle individual Byzantine agreements.* In a sense, they only guarantee "eventual consensus on a growing sequence of values". Theirs is a *state replication* protocol, rather than a BA one, and cannot be used to reach Byzantine agreement on an individual value of interest. By contrast, Algorand can also be used only once, if so wanted, to enable millions of users to quickly reach Byzantine agreement on a specific value of interest.

(4) *They require weakly synchronized clocks.* That is, all users' clocks are offset by a small time δ. By contrast, in Algorand, clocks need only have (essentially) the same "speed".

(5) *Their protocol works with lazy-but-honest users or with honest majority of online users.* They kindly credit Algorand for raising the issue of honest users going offline en masse, and for putting forward the lazy honesty model in response. Their protocol not only works in the lazy honesty model, but also in their *adversarial sleepy model*, where an adversary chooses which users are online and which are offline, provided that, at all times, the majority of online users are honest.[2]

[2]The original version of their paper actually considered only security in their adversarial sleepy model. The original version of Algorand, which precedes theirs, also explicitly envisaged assuming that a given majority of the online players is always honest, but explicitly excluded it from consideration, in favor of the lazy honesty model. (For instance, if at some point in time half of the honest users choose to go off-line, then the majority of the users on-line may very well be malicious. Thus, to prevent this from happening, the Adversary should *force* most of his corrupted players to go off-line too, which clearly is against his own interest.) Notice that a protocol with a majority of lazy-but-honest players works just fine if the majority of the users on-line are always malicious. This is so, because a sufficient number of honest players, knowing that they are going to be crucial at some rare point in time, will elect not to go off-line in those moments, nor can they be forced off-line by the Adversary, since he does not know who the crucial honest players might be.

(6) *They require a simple honest majority.* By contrast, the current version of Algorand requires a 2/3 honest majority.

Another paper close to us is *Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol*, by Kiayias, Russell, David, and Oliynykov [20]. Also their system appeared after ours. It also uses crytpographic sortition to dispense with proof of work in a provable manner. However, their system is, again, a Nakamoto-style protocol, in which forks are both unavoidable and frequent. (However, in their model, blocks need not as deep as the sleepy-consensus model.) Moreover, their system relies on the following assumptions: in the words of the authors themselves, "(1) the network is highly synchronous, (2) the majority of the selected stakeholders is available as needed to participate in each epoch, (3) the stakeholders do not remain offline for long periods of time, (4) the adaptivity of corruptions is subject to a small delay that is measured in rounds linear in the security parameter." By contrast, Algorand is, with overwhelming probability, fork-free, and does not rely on any of these 4 assumptions. In particular, in Algorand, the Adversary is able to instantaneously corrupt the users he wants to control.

2 Preliminaries

2.1 Cryptographic Primitives

Ideal Hashing. We shall rely on an efficiently computable cryptographic hash function, H, that maps arbitrarily long strings to binary strings of fixed length. Following a long tradition, we model H as a *random oracle*, essentially a function mapping each possible string s to a randomly and independently selected (and then fixed) binary string, $H(s)$, of the chosen length.

In this paper, H has 256-bit long outputs. Indeed, such length is short enough to make the system efficient and long enough to make the system secure. For instance, we want H to be *collision-resilient*. That is, it should be hard to find two different strings x and y such that $H(x) = H(y)$. When H is a random oracle with 256-bit long outputs, finding any such pair of strings is indeed difficult. (Trying at random, and relying on the birthday paradox, would require $2^{256/2} = 2^{128}$ trials.)

Digital Signing. Digital signatures allow users to to authenticate information to each other without sharing any sharing any secret keys. A *digital signature scheme* consists of three fast algorithms: a probabilistic *key generator* G, a *signing algorithm* S, and a *verification algorithm* V.

Given a security parameter k, a sufficiently high integer, a user i uses G to produce a pair of k-bit keys (i.e., strings): a "public" key pk_i and a matching "secret" signing key sk_i. Crucially, a public key does not "betray" its corresponding secret key. That is, even given knowledge of pk_i, no one other than i is able to compute sk_i in less than astronomical time.

User i uses sk_i to digitally sign messages. For each possible message (binary string) m, i first hashes m and then runs algorithm S on inputs $H(m)$ and sk_i so as to produce the k-bit string

$$sig_{pk_i}(m) \triangleq S(H(m), sk_i) \ .^3$$

[3]Since H is collision-resilient it is practically impossible that, by signing m one "accidentally signs" a different message m'.

The binary string $sig_{pk_i}(m)$ is referred to as i's digital signature of m (relative to pk_i), and can be more simply denoted by $sig_i(m)$, when the public key pk_i is clear from context.

Everyone knowing pk_i can use it to verify the digital signatures produced by i. Specifically, on inputs (a) the public key pk_i of a player i, (b) a message m, and (c) a string s, that is, i's alleged digital signature of the message m, the verification algorithm V outputs either YES or NO.

The properties we require from a digital signature scheme are:

1. *Legitimate signatures are always verified:* If $s = sig_i(m)$, then $V(pk_i, m, s) = YES$; and
2. *Digital signatures are hard to forge:* Without knowledge of sk_i the time to find a string s such that $V(pk_i, m, s) = YES$, for a message m never signed by i, is astronomically long.

   (Following the strong security requirement of Goldwasser, Micali, and Rivest [17], this is true even if one can obtain the signature of any other message.)

Accordingly, to prevent anyone else from signing messages on his behalf, a player i must keep his signing key sk_i secret (hence the term "secret key"), and to enable anyone to verify the messages he does sign, i has an interest in publicizing his key pk_i (hence the term "public key").

In general, a message m is not retrievable from its signature $sig_i(m)$. In order to virtually deal with digital signatures that satisfy the conceptually convenient *"retrievability" property* (i.e., to guarantee that the signer and the message are easily computable from a signature, we define

$$SIG_{pk_i}(m) = (i, m, sig_{pk_i}(m)) \quad \text{and} \quad SIG_i(m) = (i, m, sig_i(m)), \text{ if } pk_i \text{ is clear.}$$

Unique Digital Signing. We also consider digital signature schemes (G, S, V) satisfying the following additional property.

3. *Uniqueness.* It is hard to find strings pk', m, s, and s' such that
$$s \neq s' \quad \text{and} \quad V(pk', m, s) = V(pk', m, s') = 1.$$

   (Note that the uniqueness property holds also for strings pk' that are not legitimately generated public keys. In particular, however, the uniqueness property implies that, if one used the specified key generator G to compute a public key pk together with a matching secret key sk, and thus knew sk, it would be essentially impossible also for him to find two different digital signatures of a same message relative to pk.)

Remarks

- From Unique signatures to verifiable random functions. Relative to a digital signature scheme with the uniqueness property, the mapping $m \to H(sig_i(m))$ associates to each possible string m, a unique, randomly selected, 256-bit string, and the correctness of this mapping can be proved given the signature $sig_i(m)$.

  That is, ideal hashing and digital signature scheme satisfying the uniqueness property essentially provide an elementary implementation of a *verifiable random function*, as introduced and by Micali, Rabin, and Vadhan [27]. (Their original implementation was necessarily more complex, since they did not rely on ideal hashing.)

- THREE DIFFERENT NEEDS FOR DIGITAL SIGNATURES. In Algorand, a user i relies on digital signatures for

  (1) *Authenticating i's own payments.* In this application, keys can be "long-term" (i.e., used to sign many messages over a long period of time) and come from a ordinary signature scheme.

  (2) *Generating credentials proving that i is entitled to act at some step s of a round r.* Here, keys can be long-term, but must come from a scheme satisfying the uniqueness property.

  (3) *Authenticating the message i sends in each step in which he acts.* Here, keys must be ephemeral (i.e., destroyed after their first use), but can come from an ordinary signature scheme.

- A SMALL-COST SIMPLIFICATION. For simplicity, we envision each user i to have a single long-term key. Accordingly, such a key must come from a signature scheme with the uniqueness property. Such simplicity has a small computational cost. Typically, in fact, unique digital signatures are slightly more expensive to produce and verify than ordinary signatures.

2.2 The Idealized Public Ledger

Algorand tries to mimic the following payment system, based on an *idealized public ledger*.

1. *The Initial Status.* Money is associated with individual public keys (privately generated and owned by users). Letting $pk_1, \ldots, pk_j$ be the initial public keys and $a_1, \ldots, a_j$ their respective initial amounts of money units, then the *initial status* is

   $$S_0 = (pk_1, a_1), \ldots, (pk_j, a_j) \ ,$$

   which is assumed to be common knowledge in the system.

2. *Payments.* Let pk be a public key currently having $a \geq 0$ money units, pk' another public key, and a' a non-negative number no greater than a. Then, a (valid) payment $\wp$ is a digital signature, relative to pk, specifying the transfer of a' monetary units from pk to pk', together with some additional information. In symbols,

   $$\wp = SIG_{pk}(pk, pk', a', I, H(\mathcal{I})),$$

   where I represents any additional information deemed useful but not sensitive (e.g., time information and a payment identifier), and $\mathcal{I}$ any additional information deemed sensitive (e.g., the reason for the payment, possibly the identities of the owners of pk and the pk', and so on).

   We refer to pk (or its owner) as the *payer*, to each pk' (or its owner) as a *payee*, and to a' as the *amount* of the payment $\wp$.

   Free Joining Via Payments. Note that users may join the system whenever they want by generating their own public/secret key pairs. Accordingly, the public key pk' that appears in the payment $\wp$ above may be a newly generated public key that had never "owned" any money before.

3. *The Magic Ledger.* In the Idealized System, all payments are valid and appear in a tamper-proof list L of sets of payments "posted on the sky" for everyone to see:

   $$L = PAY^1, PAY^2, \ldots,$$

Each block PAY^{r+1} consists of the set of all payments made since the appearance of block PAY^r. In the ideal system, a new block appears after a fixed (or finite) amount of time.

Discussion.

- *More General Payments and Unspent Transaction Output.* More generally, if a public key pk owns an amount a, then a valid payment $\wp$ of pk may transfer the amounts $a'_1, a'_2, \ldots$, respectively to the keys $pk'_1, pk'_2, \ldots$, so long as $\sum_j a'_j \leq a$.

  In Bitcoin and similar systems, the money owned by a public key pk is segregated into separate amounts, and a payment $\wp$ made by pk must transfer such a segregated amount a in its entirety. If pk wishes to transfer only a fraction $a' < a$ of a to another key, then it must also transfer the balance, the *unspent transaction output*, to another key, possibly pk itself.

  Algorand also works with keys having segregated amounts. However, in order to focus on the novel aspects of Algorand, it is conceptually simpler to stick to our simpler forms of payments and keys having a single amount associated to them.

- *Current Status.* The Idealized Scheme does not directly provide information about the current status of the system (i.e., about how many money units each public key has). This information is deducible from the Magic Ledger.

  In the ideal system, an active user continually stores and updates the latest status information, or he would otherwise have to reconstruct it, either from scratch, or from the last time he computed it. (In the next version of this paper, we shall augment Algorand so as to enable its users to reconstruct the current status in an efficient manner.)

- *Security and "Privacy".* Digital signatures guarantee that no one can forge a payment by another user. In a payment $\wp$, the public keys and the amount are not hidden, but the sensitive information $\mathcal{I}$ is. Indeed, only $H(\mathcal{I})$ appears in $\wp$, and since H is an ideal hash function, $H(\mathcal{I})$ is a random 256-bit value, and thus there is no way to figure out what $\mathcal{I}$ was better than by simply guessing it. Yet, to prove what $\mathcal{I}$ was (e.g., to prove the reason for the payment) the payer may just reveal $\mathcal{I}$. The correctness of the revealed $\mathcal{I}$ can be verified by computing $H(\mathcal{I})$ and comparing the resulting value with the last item of $\wp$. In fact, since H is *collision resilient,* it is hard to find a second value $\mathcal{I}'$ such that $H(\mathcal{I}) = H(\mathcal{I}')$.

2.3 Basic Notions and Notations

Keys, Users, and Owners Unless otherwise specified, each public key ("key" for short) is long-term and relative to a digital signature scheme with the uniqueness property. A public key i joins the system when another public key j already in the system makes a payment to i.

For color, we personify keys. We refer to a key i as a "he", say that i is honest, that i sends and receives messages, etc. *User* is a synonym for key. When we want to distinguish a key from the person to whom it belongs, we respectively use the term "digital key" and "owner".

Permissionless and Permissioned Systems. A system is *permissionless*, if a digital key is free to join at any time and an owner can own multiple digital keys; and its *permissioned*, otherwise.

Unique Representation Each object in Algorand has a unique representation. In particular, each set $\{(x, y, z, \ldots) : x \in X, y \in Y, z \in Z, \ldots\}$ is ordered in a pre-specified manner: e.g., first lexicographically in x, then in y, etc.

Same-Speed Clocks There is no global clock: rather, each user has his own clock. User clocks need not be synchronized in any way. We assume, however, that they all have the same speed.

For instance, when it is 12pm according to the clock of a user i, it may be 2:30pm according to the clock of another user j, but when it will be 12:01 according to i's clock, it will 2:31 according to j's clock. That is, "one minute is the same (sufficiently, essentially the same) for every user".

Rounds Algorand is organized in logical units, $r = 0, 1, \ldots$, called *rounds*.

We consistently use superscripts to indicate rounds. To indicate that a non-numerical quantity Q (e.g., a string, a public key, a set, a digital signature, etc.) refers to a round r, we simply write Q^r. Only when Q is a genuine number (as opposed to a binary string interpretable as a number), do we write $Q^{(r)}$, so that the symbol r could not be interpreted as the exponent of Q.

At (the start of a) round $r > 0$, the set of all public keys is PK^r, and the system status is

$$S^r = \left\{ \left(i, a_i^{(r)}, \ldots \right) : i \in PK^r \right\},$$

where $a_i^{(r)}$ is the amount of money available to the public key i. Note that PK^r is deducible from S^r, and that S^r may also specify other components for each public key i.

For round 0, PK^0 is the set of *initial public keys*, and S^0 is *the initial status*. Both PK^0 and S^0 are assumed to be common knowledge in the system. For simplicity, at the start of round r, so are $PK^1, \ldots, PK^r$ and $S^1, \ldots, S^r$.

In a round r, the system status transitions from S^r to S^{r+1}: symbolically,

$$\text{Round } r\text{: } S^r \longrightarrow S^{r+1}.$$

Payments In Algorand, the users continually make payments (and disseminate them in the way described in subsection 2.7). A payment $\wp$ of a user $i \in PK^r$ has the same format and semantics as in the Ideal System. Namely,

$$\wp = SIG_i(i, i', a, I, H(\mathcal{I})) \ .$$

Payment $\wp$ is *individually valid at a round* r (is a *round-r payment*, for short) if (1) its amount a is less than or equal to $a_i^{(r)}$, and (2) it does not appear in any official payset $PAY^{r'}$ for $r' < r$. (As explained below, the second condition means that $\wp$ has not already become effective.

A set of round-r payments of i is *collectively valid* if the sum of their amounts is at most $a_i^{(r)}$.

Paysets A round-r *payset* $\mathcal{P}$ is a set of round-r payments such that, for each user i, the payments of i in $\mathcal{P}$ (possibly none) are collectively valid. The set of all round-r paysets is $\mathbb{PAY}(r)$. A round-r payset $\mathcal{P}$ is *maximal* if no superset of $\mathcal{P}$ is a round-r payset.

We actually suggest that a payment $\wp$ also specifies a round ρ, $\wp = SIG_i(\rho, i, i', a, I, H(\mathcal{I}))$, and cannot be valid at any round outside $[\rho, \rho + k]$, for some fixed non-negative integer k.[4]

[4]This simplifies checking whether $\wp$ has become "effective" (i.e., it simplifies determining whether some payset PAY^r contains $\wp$. When $k = 0$, if $\wp = SIG_i(r, i, i', a, I, H(\mathcal{I}))$, and $\wp \notin PAY^r$, then i must re-submit $\wp$.

Official Paysets For every round r, Algorand publicly selects (in a manner described later on) a single (possibly empty) payset, PAY^r, the round's *official payset*. (Essentially, PAY^r represents the round-r payments that have *"actually"* happened.)

As in the Ideal System (and Bitcoin), (1) the only way for a new user j to enter the system is to be the recipient of a payment belonging to the official payset PAY^r of a given round r; and (2) PAY^r determines the status of the next round, S^{r+1}, from that of the current round, S^r. Symbolically,

$$PAY^r : S^r \longrightarrow S^{r+1}.$$

Specifically,

1. the set of public keys of round $r+1$, PK^{r+1}, consists of the union of PK^r and the set of all payee keys that appear, for the first time, in the payments of PAY^r; and
2. the amount of money $a_i^{(r+1)}$ that a user i owns in round $r+1$ is the sum of $a_i(r)$ —i.e., the amount of money i owned in the previous round (0 if $i \notin PK^r$)— and the sum of amounts paid to i according to the payments of PAY^r.

In sum, as in the Ideal System, each status S^{r+1} is deducible from the previous payment history:

$$PAY^0, \ldots, PAY^r.$$

2.4 Blocks and Proven Blocks

In $Algorand_0$, the block B^r corresponding to a round r specifies: r itself; the set of payments of round r, PAY^r; a quantity Q^r, to be explained, and the hash of the previous block, $H(B^{r-1})$. Thus, starting from some fixed block B^0, we have a traditional blockchain:

$$B^1 = (1, PAY^1, Q^0, H(B^0)), \quad B^2 = (2, PAY^2, Q^1, H(B^1)), \quad B^3 = (3, PAY^3, Q^2, H(B^2)), \quad \ldots$$

In Algorand, the authenticity of a block is actually vouched by a separate piece of information, a "block certificate" $CERT^r$, which turns B^r into a *proven block*, $\overline{B^r}$. The Magic Ledger, therefore, is implemented by the sequence of the proven blocks,

$$\overline{B^1}, \overline{B^2}, \ldots$$

Discussion As we shall see, $CERT^r$ consists of a set of digital signatures for $H(B^r)$, those of a majority of the members of SV^r, together with a proof that each of those members indeed belongs to SV^r. We could, of course, include the certificates $CERT^r$ in the blocks themselves, but find it conceptually cleaner to keep it separate.)

In Bitcoin each block must satisfy a special property, that is, must "contain a solution of a crypto puzzle", which makes block generation computationally intensive and forks both inevitable and not rare. By contrast, Algorand's blockchain has two main advantages: it is generated with minimal computation, and it will not fork with overwhelmingly high probability. Each block B^i is safely *final* as soon as it enters the blockchain.

2.5 Acceptable Failure Probability

To analyze the security of Algorand we specify the probability, F, with which we are willing to accept that something goes wrong (e.g., that a verifier set SV^r does not have an honest majority). As in the case of the output length of the cryptographic hash function H, also F is a parameter. But, as in that case, we find it useful to set F to a concrete value, so as to get a more intuitive grasp of the fact that it is indeed possible, in Algorand, to enjoy simultaneously sufficient security and sufficient efficiency. To emphasize that F is parameter that can be set as desired, in the first and second embodiments we respectively set

$$F = 10^{-12} \quad \text{and} \quad F = 10^{-18} \ .$$

Discussion Note that 10^{-12} is actually less than one in a trillion, and we believe that such a choice of F is adequate in our application. Let us emphasize that 10^{-12} is *not* the probability with which the Adversary can forge the payments of an honest user. All payments are digitally signed, and thus, if the proper digital signatures are used, the probability of forging a payment is far lower than 10^{-12}, and is, in fact, essentially 0. The bad event that we are willing to tolerate with probability F is that Algorand's blockchain *forks*. Notice that, with our setting of F and one-minute long rounds, a fork is expected to occur in Algorand's blockchain as infrequently as (roughly) once in 1.9 million years. By contrast, in Bitcoin, a forks occurs quite often.

A more demanding person may set F to a lower value. To this end, in our second embodiment we consider setting F to 10^{-18}. Note that, assuming that a block is generated every *second*, 10^{18} is the estimated number of seconds taken by the Universe so far: from the Big Bang to present time. Thus, with $F = 10^{-18}$, if a block is generated in a second, one should expect for the age of the Universe to see a fork.

2.6 The Adversarial Model

Algorand is designed to be secure in a very adversarial model. Let us explain.

Honest and Malicious Users A user is *honest* if he follows all his protocol instructions, and is perfectly capable of sending and receiving messages. A user is *malicious* (i.e., *Byzantine*, in the parlance of distributed computing) if he can deviate arbitrarily from his prescribed instructions.

The Adversary The *Adversary* is an efficient (technically polynomial-time) algorithm, personified for color, who can *immediately* make malicious *any user* he wants, at *any time* he wants (subject only to an upperbound to the number of the users he can corrupt).

The Adversary totally controls and perfectly coordinates all malicious users. He takes all actions on their behalf, including receiving and sending all their messages, and can let them deviate from their prescribed instructions in arbitrary ways. Or he can simply isolate a corrupted user sending and receiving messages. Let us clarify that no one else automatically learns that a user i is malicious, although i's maliciousness may transpire by the actions the Adversary has him take.

This powerful adversary however,

- Does not have unbounded computational power and cannot successfully forge the digital signature of an honest user, except with negligible probability; and

- Cannot interfere in any way with the messages exchanges among honest users.

Furthermore, his ability to attack honest users is bounded by one of the following assumption.

Honesty Majority of Money We consider a continuum of Honest Majority of Money (HMM) assumptions: namely, for each non-negative integer k and real $h > 1/2$,

> *$HHM_k > h$: the honest users in every round r owned a fraction greater than h of all money in the system at round $r - k$.*

Discussion. Assuming that all malicious users perfectly coordinate their actions (as if controlled by a single entity, the Adversary) is a rather pessimistic hypothesis. Perfect coordination among too many individuals is difficult to achieve. Perhaps coordination only occurs within separate groups of malicious players. But, since one cannot be sure about the level of coordination malicious users may enjoy, we'd better be safe than sorry.

Assuming that the Adversary can secretly, dynamically, and immediately corrupt users is also pessimistic. After all, realistically, taking full control of a user's operations should take some time.

The assumption $HMM_k > h$ implies, for instance, that, if a round (on average) is implemented in one minute, then, the majority of the money at a given round will remain in honest hands for at least two hours, if $k = 120$, and at least one week, if $k = 10,000$.

Note that the HMM assumptions and the previous Honest Majority of Computing Power assumptions are related in the sense that, since computing power can be bought with money, if malicious users own most of the money, then they can obtain most of the computing power.

2.7 The Communication Model

We envisage message propagation —i.e., "peer-to-peer gossip"[5]— to be the only means of communication.

Temporary Assumption: Timely Delivery of Messages in the Entire Network. For most part of this paper we assume that every propagated message reaches almost all honest users in a timely fashion. We shall remove this assumption in Section 10, where we deal with network partitions, either naturally occurring or adversarially induced. (As we shall see, we only assume timely delivery of messages within each connected component of the network.)

One concrete way to capture timely delivery of propagated messages (in the entire network) is the following:

> *For all* reachability $\rho > 95\%$ *and* message size $\mu \in \mathbb{Z}_+$, *there exists* $\lambda_{\rho,\mu}$ *such that,*
>
> *if a honest user propagates μ-byte message m at time t,*
>
> *then m reaches, by time $t + \lambda_{\rho,\mu}$, at least a fraction ρ of the honest users.*

[5]Essentially, as in Bitcoin, when a user propagates a message m, every active user i receiving m for the first time, randomly and independently selects a suitably small number of active users, his *"neighbors"*, to whom he forwards m, possibly until he receives an acknowledgement from them. The propagation of m terminates when no user receives m for the first time.

The above property, however, cannot support our Algorand protocol, without explicitly and separately envisaging a mechanism to obtain the latest blockchain —by another user/depository/etc. In fact, to construct a new block B^r not only should a proper set of verifiers timely receive round-r messages, but also the messages of previous rounds, so as to know B^{r-1} and all other previous blocks, which is necessary to determine whether the payments in B^r are valid. The following assumption instead suffices.

Message Propagation (MP) Assumption: *For all $\rho > 95\%$ and $\mu \in \mathbb{Z}_+$, there exists $\lambda_{\rho,\mu}$ such that, for all times t and all μ-byte messages m propagated by an honest user before $t - \lambda_{\rho,\mu}$, m is received, by time t, by at least a fraction ρ of the honest users.*

Protocol *Algorand′* actually instructs each of a small number of users (i.e., the verifiers of a given step of a round in *Algorand′*, to propagate a separate message of a (small) prescribed size, and we need to bound the time required to fulfill these instructions. We do so by enriching the MP assumption as follows.

For all n, $\rho > 95\%$, and $\mu \in \mathbb{Z}_+$, there exists $\lambda_{n,\rho,\mu}$ such that, for all times t and all μ-byte messages $m_1, \ldots, m_n$, each propagated by an honest user before $t - \lambda_{n,\rho,\mu}$, $m_1, \ldots, m_n$ are received, by time t, by at least a fraction ρ of the honest users.

Note

- The above assumption is deliberately simple, but also stronger than needed in our paper.[6]
- For simplicity, we assume $\rho = 1$, and thus drop mentioning ρ.
- We pessimistically assume that, provided he does not violate the MP assumption, the Adversary totally controls the delivery of all messages. In particular, without being noticed by the honest users, the Adversary he can arbitrarily decide which honest player receives which message when, and arbitrarily accelerate the delivery of any message he wants.[7]

3 The BA Protocol $BA^\star$ in a Traditional Setting

As already emphasized, Byzantine agreement is a key ingredient of Algorand. Indeed, it is through the use of such a BA protocol that Algorand is unaffected by forks. However, to be secure against our powerful Adversary, Algorand must rely on a BA protocol that satisfies the new player-replaceability constraint. In addition, for Algorand to be efficient, such a BA protocol must be very efficient.

BA protocols were first defined for an idealized communication model, *synchronous complete networks* (SC networks). Such a model allows for a simpler design and analysis of BA protocols.

[6]Given the honest percentage h and the acceptable failure probability F, Algorand computes an upperbound, N, to the maximum number of member of verifiers in a step. Thus, the MP assumption need only hold for $n \leq N$.

In addition, as stated, the MP assumption holds no matter how many other messages may be propagated alongside the m_j's. As we shall see, however, in Algorand messages at are propagated in essentially non-overlapping time intervals, during which either a single block is propagated, or at most N verifiers propagate a small (e.g., 200B) message. Thus, we could restate the MP assumption in a weaker, but also more complex, way.

[7]For instance, he can immediately learn the messages sent by honest players. Thus, a malicious user i', who is asked to propagate a message simultaneously with a honest user i, can always choose his own message m' based on the message m actually propagated by i. This ability is related to *rushing*, in the parlance of distributed-computation literature.

Accordingly, in this section, we introduce a new BA protocol, $BA^\star$, for SC networks and ignoring the issue of player replaceability altogether. The protocol $BA^\star$ is a contribution of separate value. Indeed, it is the most efficient cryptographic BA protocol for SC networks known so far.

To use it within our Algorand protocol, we modify $BA^\star$ a bit, so as to account for our different communication model and context, but make sure, in section X, to highlight how $BA^\star$ is used within our actual protocol *Algorand′*.

We start by recalling the model in which $BA^\star$ operates and the notion of a Byzantine agreement.

3.1 Synchronous Complete Networks and Matching Adversaries

In a SC network, there is a common clock, ticking at each integral times $r = 1, 2, \ldots$

At each even time click r, each player i instantaneously and simultaneously sends a single message $m_{i,j}^r$ (possibly the empty message) to each player j, including himself. Each $m_{i,j}^r$ is received at time click $r + 1$ by player j, together with the identity of the sender i.

Again, in a communication protocol, a player is *honest* if he follows all his prescribed instructions, and *malicious* otherwise. All malicious players are totally controlled and perfectly coordinated by the Adversary, who, in particular, immediately receives all messages addressed to malicious players, and chooses the messages they send.

The Adversary can immediately make malicious any honest user he wants at any odd time click he wants, subject only to a possible upperbound t to the number of malicious players. That is, the Adversary "cannot interfere with the messages already sent by an honest user i", which will be delivered as usual.

The Adversary also has the additional ability to see instantaneously, at each even round, the messages that the currently honest players send, and instantaneously use this information to choose the messages the malicious players send at the same time tick.

Remarks

- *Adversary Power.* The above setting is very adversarial. Indeed, in the Byzantine agreement literature, many settings are less adversarial. However, some more adversarial settings have also been considered, where the Adversary, after seeing the messages sent by an honest player i at a given time click r, has the ability to erase all these messages from the network, immediately corrupt i, choose the message that the now malicious i sends at time click r, and have them delivered as usual. The envisaged power of the Adversary matches that he has in our setting.

- *Physical Abstraction.* The envisaged communication model abstracts a more physical model, in which each pair of players (i, j) is linked by a separate and private communication line $l_{i,j}$. That is, no one else can inject, interfere with, or gain information about the messages sent over $l_{i,j}$. The only way for the Adversary to have access to $l_{i,j}$ is to corrupt either i or j.

- *Privacy and Authentication.* In SC networks message privacy and authentication are guaranteed by assumption. By contrast, in our communication network, where messages are propagated from peer to peer, authentication is guaranteed by digital signatures, and privacy is non-existent. Thus, to adopt protocol $BA^\star$ to our setting, each message exchanged should be digitally signed (further identifying the state at which it was sent). Fortunately, the BA protocols that we consider using in Algorand do not require message privacy.

3.2 The Notion of a Byzantine Agreement

The notion of Byzantine agreement was introduced by Pease Shostak and Lamport [31] for the *binary* case, that is, when every initial value consists of a bit. However, it was quickly extended to arbitrary initial values. (See the surveys of Fischer [16] and Chor and Dwork [10].) By a BA protocol, we mean an arbitrary-value one.

Definition 3.1. *In a synchronous network, let $\mathcal{P}$ be a n-player protocol, whose player set is common knowledge among the players, t a positive integer such that $n \geq 2t+1$. We say that $\mathcal{P}$ is an* arbitrary-value *(respectively,* binary*)* (n,t)-Byzantine agreement protocol *with* soundness $\sigma \in (0,1)$ *if, for every set of values V not containing the special symbol $\perp$ (respectively, for $V = \{0,1\}$), in an execution in which at most t of the players are malicious and in which every player i starts with an* initial value $v_i \in V$*, every honest player j halts with probability 1, outputting a value $out_i \in V \cup \{\perp\}$ so as to satisfy, with probability at least σ, the following two conditions:*

1. Agreement: *There exists $out \in V \cup \{\perp\}$ such that $out_i = out$ for all honest players i.*
2. Consistency: *if, for some value $v \in V$, $v_i = v$ for all honest players, then $out = v$.*

We refer to out as $\mathcal{P}$'s output*, and to each out_i as* player i's output.

3.3 The BA Notation

In our BA protocols, a player is required to count how many players sent him a given message in a given step. Accordingly, for each possible value v that might be sent,

$$\#_i^s(v)$$

(or just $\#_i(v)$ when s is clear) is the number of players j from which i has received v in step s.

Recalling that a player i receives exactly one message from each player j, if the number of players is n, then, for all i and s, $\sum_v \#_i^s(v) = n$.

3.4 The Binary BA Protocol $BBA^\star$

In this section we present a new *binary* BA protocol, $BBA^\star$, which relies on the honesty of more than two thirds of the players and is very fast: no matter what the malicious players might do, each execution of its main loop brings the players into agreement with probability 1/3.

Each player has his own public key of a digital signature scheme satisfying the unique-signature property. Since this protocol is intended to be run on synchronous complete network, there is no need for a player i to sign each of his messages.

Digital signatures are used to generate a sufficiently common random bit in Step 3. (In Algorand, digital signatures are used to authenticate all other messages as well.)

The protocol requires a minimal set-up: a common random string r, independent of the players' keys. (In Algorand, r is actually replaced by the quantity Q^r.)

Protocol $BBA^\star$ is a 3-step loop, where the players repeatedly exchange Boolean values, and different players may exit this loop at different times. A player i exits this loop by propagating, at some step, either a special value $0*$ or a special value $1*$, thereby instructing all players to "pretend" they respectively receive 0 and 1 from i in all future steps. (Alternatively said: assume

that the last message received by a player j from another player i was a bit b. Then, in any step in which he does not receive any message from i, j acts as if i sent him the bit b.)

The protocol uses a counter γ, representing how many times its 3-step loop has been executed. At the start of $BBA^\star$, $\gamma = 0$. (One may think of γ as a global counter, but it is actually increased by each individual player every time that the loop is executed.)

There are $n \geq 3t+1$, where t is the maximum possible number of malicious players. A binary string x is identified with the integer whose binary representation (with possible leadings 0s) is x; and $\mathtt{lsb}(x)$ denotes the least significant bit of x.

PROTOCOL $BBA^\star$

(COMMUNICATION) STEP 1. [Coin-Fixed-To-0 Step] *Each player i sends b_i.*

1.1 If $\#_i^1(0) \geq 2t+1$, then i sets $b_i = 0$, sends $0*$, outputs $out_i = 0$, and HALTS.

1.2 If $\#_i^1(1) \geq 2t+1$, then, then i sets $b_i = 1$.

1.3 Else, i sets $b_i = 0$.

(COMMUNICATION) STEP 2. [Coin-Fixed-To-1 Step] *Each player i sends b_i.*

2.1 If $\#_i^2(1) \geq 2t+1$, then i sets $b_i = 1$, sends $1*$, outputs $out_i = 1$, and HALTS.

2.2 If $\#_i^2(0) \geq 2t+1$, then i set $b_i = 0$.

2.3 Else, i sets $b_i = 1$.

(COMMUNICATION) STEP 3. [Coin-Genuinely-Flipped Step] *Each player i sends b_i and $SIG_i(r,\gamma)$.*

3.1 If $\#_i^3(0) \geq 2t+1$, then i sets $b_i = 0$.

3.2 If $\#_i^3(1) \geq 2t+1$, then i sets $b_i = 1$.

3.3 Else, letting $S_i = \{j \in N$ who have sent i a proper message in this step 3 $\}$, i sets $b_i = c \triangleq \mathtt{lsb}(\min_{j \in S_i} H(SIG_i(r,\gamma)))$; increases γ_i by 1; and returns to Step 1.

Theorem 3.1. *Whenever $n \geq 3t+1$, $BBA^\star$ is a binary (n,t)-BA protocol with soundness 1.*

A proof of Theorem 3.1 is given in [26]. Its adaptation to our setting, and its player-replaceability property are novel.

Historical Remark Probabilistic binary BA protocols were first proposed by Ben-Or in asynchronous settings [7]. Protocol $BBA^\star$ is a novel adaptation, to our public-key setting, of the binary BA protocol of Feldman and Micali [15]. Their protocol 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, who implemented it via an external trusted party [32].

3.5 Graded Consensus and the Protocol *GC*

Let us recall, for arbitrary values, a notion of consensus much weaker than Byzantine agreement.

Definition 3.2. *Let $\mathcal{P}$ be a protocol in which the set of all players is common knowledge, and each player i privately knows an arbitrary* initial value v'_i.

We say that $\mathcal{P}$ is an (n,t)-graded consensus protocol *if, in every execution with n players, at most t of which are malicious, every honest player i halts outputting a* value-grade *pair (v_i, g_i), where $g_i \in \{0,1,2\}$, so as to satisfy the following three conditions:*

1. *For all honest players i and j, $|g_i - g_j| \leq 1$.*
2. *For all honest players i and j, $g_i, g_j > 0 \Rightarrow v_i = v_j$.*
3. *If $v'_1 = \cdots = v'_n = v$ for some value v, then $v_i = v$ and $g_i = 2$ for all honest players i.*

Historical Note The notion of a graded consensus is simply derived from that of a *graded broadcast*, put forward by Feldman and Micali in [15], by strengthening the notion of a *crusader agreement*, as introduced by Dolev [12], and refined by Turpin and Coan [33].[8]

In [15], the authors also provided a 3-step (n,t)-graded broadcasting protocol, *gradecast*, for $n \geq 3t+1$. A more complex (n,t)-graded-broadcasting protocol for $n > 2t+1$ has later been found by Katz and Koo [19].

The following two-step protocol *GC* consists of the last two steps of *gradecast*, expressed in our notation. To emphasize this fact, and to match the steps of protocol *Algorand′* of section 4.1, we respectively name 2 and 3 the steps of *GC*.

PROTOCOL *GC*

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

Theorem 3.2. *If $n \geq 3t+1$, then GC is a (n,t)-graded broadcast protocol.*

The proof immediately follows from that of the protocol *gradecast* in [15], and is thus omitted.[9]

[8] In essence, in a graded-broadcasting protocol, (a) the input of every player is the identity of a distinguished player, the *sender*, who has an arbitrary value v as an additional private input, and (b) the outputs must satisfy the same properties 1 and 2 of graded consensus, plus the following property 3′: *if the sender is honest, then $v_i = v$ and $g_i = 2$ for all honest player i.*

[9] Indeed, in their protocol, in step 1, the sender sends his own private value v to all players, and each player i lets v'_i consist of the value he has actually received from the sender in step 1.

3.6 The Protocol $BA^\star$

We now describe the arbitrary-value BA protocol $BA^\star$ via the binary BA protocol $BBA^\star$ and the graded-consensus protocol GC. Below, the initial value of each player i is v'_i.

PROTOCOL $BA^\star$

STEPS 1 AND 2. *Each player i executes GC, on input v'_i, so as to compute a pair (v_i, g_i).*

STEP 3, ... *Each player i executes $BBA^\star$ —with initial input 0, if $g_i = 2$, and 1 otherwise— so as to compute the bit out_i.*

OUTPUT DETERMINATION. *Each player i outputs v_i, if $out_i = 0$, and $\perp$ otherwise.*

Theorem 3.3. *Whenever $n \geq 3t + 1$, $BA^\star$ is a (n, t)-BA protocol with soundness 1.*

Proof. We first prove Consistency, and then Agreement.

PROOF OF CONSISTENCY. Assume that, for some value $v \in V$, $v'_i = v$. Then, by property 3 of graded consensus, after the execution of GC, all honest players output $(v, 2)$. Accordingly, 0 is the initial bit of all honest players in the end of the execution of $BBA^\star$. Thus, by the Agreement property of binary Byzantine agreement, at the end of the execution of $BA^\star$, $out_i = 0$ for all honest players. This implies that the output of each honest player i in $BA^\star$ is $v_i = v$. □

PROOF OF AGREEMENT. Since $BBA^\star$ is a binary BA protocol, either
(A) $out_i = 1$ for all honest player i, or
(B) $out_i = 0$ for all honest player i.
In case A, all honest players output $\perp$ in $BA^\star$, and thus Agreement holds. Consider now case B. In this case, in the execution of $BBA^\star$, the initial bit of at least one honest player i is 0. (Indeed, if initial bit of all honest players were 1, then, by the Consistency property of $BBA^\star$, we would have $out_j = 1$ for all honest j.) Accordingly, after the execution of GC, i outputs the pair $(v, 2)$ for some value v. Thus, by property 1 of graded consensus, $g_j > 0$ for all honest players j. Accordingly, by property 2 of graded consensus, $v_j = v$ for all honest players j. This implies that, at the end of $BA^\star$, every honest player j outputs v. Thus, Agreement holds also in case B. □

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

Historical Note Turpin and Coan were the first to show that, for $n \geq 3t+1$, any binary (n, t)-BA protocol can be converted to an arbitrary-value (n, t)-BA protocol. The reduction arbitrary-value Byzantine agreement to binary Byzantine agreement via graded consensus is more modular and cleaner, and simplifies the analysis of our Algorand protocol *Algorand′*.

Generalizing $BA^\star$ for use in Algorand Algorand works even when all communication is via gossiping. However, although presented in a traditional and familiar communication network, so as to enable a better comparison with the prior art and an easier understanding, protocol $BA^\star$ works also in gossiping networks. In fact, in our detailed embodiments of Algorand, we shall present it directly for gossiping networks. We shall also point out that it satisfies the player replaceability property that is crucial for Algorand to be secure in the envisaged very adversarial model.

Any BA player-replaceable protocol working in a gossiping communication network can be securely employed within the inventive Algorand system. In particular, Micali and Vaikunthanatan have extended $BA^\star$ to work very efficiently also with a simple majority of honest players. That protocol too could be used in Algorand.

4 Two Embodiments of Algorand

As discussed, at a very high level, a round of Algorand ideally proceeds as follows. First, a randomly selected user, the leader, proposes and circulates a new block. (This process includes initially selecting a few potential leaders and then ensuring that, at least a good fraction of the time, a single common leader emerges.) Second, a randomly selected committee of users is selected, and reaches Byzantine agreement on the block proposed by the leader. (This process includes that each step of the BA protocol is run by a separately selected committee.) The agreed upon block is then digitally signed by a given threshold (T_H) of committee members. These digital signatures are circulated so that everyone is assured of which is the new block. (This includes circulating the credential of the signers, and authenticating just the hash of the new block, ensuring that everyone is guaranteed to learn the block, once its hash is made clear.)

In the next two sections, we present two embodiments of Algorand, $Algorand'_1$ and $Algorand'_2$, that work under a majority-of-honest-users assumption. In Section 8 we show how to adopts these embodiments to work under a honest-majority-of-money assumption.

$Algorand'_1$ only envisages that $> 2/3$ of the committee members are honest. In addition, in $Algorand'_1$, the number of steps for reaching Byzantine agreement is capped at a suitably high number, so that agreement is guaranteed to be reached with overwhelming probability within a fixed number of steps (but potentially requiring longer time than the steps of $Algorand'_2$). In the remote case in which agreement is not yet reached by the last step, the committee agrees on the empty block, which is always valid.

$Algorand'_2$ envisages that the number of honest members in a committee is always greater than or equal to a fixed threshold t_H (which guarantees that, with overwhelming probability, at least 2/3 of the committee members are honest). In addition, $Algorand'_2$ allows Byzantine agreement to be reached in an arbitrary number of steps (but potentially in a shorter time than $Algorand'_1$).

It is easy to derive many variants of these basic embodiments. In particular, it is easy, given $Algorand'_2$, to modify $Algorand'_1$ so as to enable to reach Byzantine agreement in an arbitrary number of steps.

Both embodiments share the following common core, notations, notions, and parameters.

4.1 A Common Core

Objectives Ideally, for each round r, Algorand would satisfy the following properties:

1. *Perfect Correctness.* All honest users agree on the same block B^r.
2. *Completeness 1.* With probability 1, the payset of B^r, PAY^r, is maximal.[10]

[10] Because paysets are defined to contain valid payments, and honest users to make only valid payments, a maximal PAY^r contains the "currently outstanding" payments of all honest users.

Of course, guaranteeing perfect correctness alone is trivial: everyone always chooses the official payset PAY^r to be empty. But in this case, the system would have completeness 0. Unfortunately, guaranteeing both perfect correctness and completeness 1 is not easy in the presence of malicious users. Algorand thus adopts a more realistic objective. Informally, letting h denote the percentage of users who are honest, $h > 2/3$, the goal of Algorand is

Guaranteeing, with overwhelming probability, perfect correctness and completeness close to h.

Privileging correctness over completeness seems a reasonable choice: payments not processed in one round can be processed in the next, but one should avoid *forks*, if possible.

Led Byzantine Agreement Perfect Correctness could be guaranteed as follows. At the start of round r, each user i constructs his own candidate block B_i^r, and then all users reach Byzantine agreement on one candidate block. As per our introduction, the BA protocol employed requires a 2/3 honest majority and is player replaceable. Each of its step can be executed by a small and randomly selected set of *verifiers*, who do not share any inner variables.

Unfortunately, this approach has no completeness guarantees. This is so, because the candidate blocks of the honest users are most likely totally different from each other. Thus, the ultimately agreed upon block might always be one with a non-maximal payset. In fact, it may always be the empty block, B_ε, that is, the block whose payset is empty. well be the default, empty one.

Algorand′ avoids this completeness problem as follows. First, a leader for round r, ℓ^r, is selected. Then, ℓ^r propagates his own candidate block, $B_{\ell^r}^r$. Finally, the users reach agreement on the block they actually receive from ℓ^r. Because, whenever ℓ^r is honest, Perfect Correctness and Completeness 1 both hold, *Algorand′* ensures that ℓ^r is honest with probability close to h. (When the leader is malicious, we do not care whether the agreed upon block is one with an empty payset. After all, a malicious leader ℓ^r might always maliciously choose $B_{\ell^r}^r$ to be the empty block, and then honestly propagate it, thus forcing the honest users to agree on the empty block.)

Leader Selection In Algorand's, the rth block is of the form $B^r = (r, PAY^r, Q^r, H(B^{r-1}))$. As already mentioned in the introduction, the quantity Q^{r-1} is carefully constructed so as to be essentially non-manipulatable by our very powerful Adversary. (Later on in this section, we shall provide some intuition about why this is the case.) At the start of a round r, all users know the blockchain so far, $B^0, \ldots, B^{r-1}$, from which they deduce the set of users of every prior round: that is, $PK^1, \ldots, PK^{r-1}$. A *potential leader* of round r is a user i such that

$$.H\left(SIG_i\left(r, 1, Q^{r-1}\right)\right) \leq p \ .$$

Let us explain. Note that, since the quantity Q^{r-1} is part of block B^{r-1}, and the underlying signature scheme satisfies the uniqueness property, $SIG_i\left(r, 1, Q^{r-1}\right)$ is a binary string uniquely associated to i and r. Thus, since H is a random oracle, $H\left(SIG_i\left(r, 1, Q^{r-1}\right)\right)$ is a random 256-bit long string uniquely associated to i and r. The symbol "." in front of $H\left(SIG_i\left(r, 1, Q^{r-1}\right)\right)$ is the *decimal* (in our case, *binary*) *point*, so that $r_i \triangleq .H\left(SIG_i\left(r, 1, Q^{r-1}\right)\right)$ is the binary expansion of a random 256-bit number between 0 and 1 uniquely associated to i and r. Thus the probability that r_i is less than or equal to p is essentially p. (Our potential-leader selection mechanism has been inspired by the micro-payment scheme of Micali and Rivest [28].)

The probability p is chosen so that, with overwhelming (i.e., $1 - F$) probability, at least one potential verifier is honest. (If fact, p is chosen to be the smallest such probability.)

Note that, since i is the only one capable of computing his own signatures, he alone can determine whether he is a potential verifier of round 1. However, by revealing his own *credential*, $\sigma_i^r \triangleq SIG_i\left(r, 1, Q^{r-1}\right)$, i can prove to anyone to be a potential verifier of round r.

The leader ℓ^r is defined to be the potential leader whose hashed credential is smaller that the hashed credential of all other potential leader j: that is, $H(\sigma_{\ell^r}^{r,s}) \leq H(\sigma_j^{r,s})$.

Note that, since a malicious ℓ^r may not reveal his credential, the correct leader of round r may never be known, and that, barring improbable ties, ℓ^r is indeed the only leader of round r.

Let us finally bring up a last but important detail: a user i can be a potential leader (and thus the leader) of a round r only if he belonged to the system for at least k rounds. This guarantees the non-manipulatability of Q^r and all future Q-quantities. In fact, one of the potential leaders will actually determine Q^r.

Verifier Selection Each step $s > 1$ of round r is executed by a small set of verifiers, $SV^{r,s}$. Again, each verifier $i \in SV^{r,s}$ is randomly selected among the users already in the system k rounds before r, and again via the special quantity Q^{r-1}. Specifically, $i \in PK^{r-k}$ is a *verifier* in $SV^{r,s}$, if

$$.H\left(SIG_i\left(r, s, Q^{r-1}\right)\right) \leq p' \ .$$

Once more, only i knows whether he belongs to $SV^{r,s}$,but, if this is the case, he could prove it by exhibiting his credential $\sigma_i^{r,s} \triangleq H(SIG_i\left(r, s, Q^{r-1}\right))$. A verifier $i \in SV^{r,s}$ sends a message, $m_i^{r,s}$, in step s of round r, and this message includes his credential $\sigma_i^{r,s}$, so as to enable the verifiers f the nest step to recognize that $m_i^{r,s}$ is a legitimate step-s message.

The probability p' is chosen so as to ensure that, in $SV^{r,s}$, letting $\#good$ be the number of honest users and $\#bad$ the number of malicious users, with overwhelming probability the following two conditions hold.

For embodiment $Algorand'_1$:
(1) $\#good > 2 \cdot \#bad$ and
(2) $\#good + 4 \cdot \#bad < 2n$, where n is the expected cardinality of $SV^{r,s}$.

For embodiment $Algorand'_2$:
(1) $\#good > t_H$ and
(2) $\#good + 2\#bad < 2t_H$, where t_H is a specified threshold.

These conditions imply that, with sufficiently high probability, (a) in the last step of the BA protocol, there will be at least given number of honest players to digitally sign the new block B^r, (b) only one block per round may have the necessary number of signatures, and (c) the used BA protocol has (at each step) the required 2/3 honest majority.

Clarifying Block Generation If the round-r leader ℓ^r is honest, then the corresponding block is of the form

$$B^r = \left(r, PAY^r, SIG_{\ell^r}\left(Q^{r-1}\right), H\left(B^{r-1}\right)\right) \ ,$$

where the payset PAY^r is maximal. (recall that all paysets are, by definition, collectively valid.)

Else (i.e., if ℓ^r is malicious), B^r has one of the following two possible forms:

$$B^r = \left(r, PAY^r, SIG_i\left(Q^{r-1}\right), H\left(B^{r-1}\right)\right) \quad \text{and} \quad B^r = B_\varepsilon^r \triangleq \left(r, \emptyset, Q^{r-1}, H\left(B^{r-1}\right)\right) \ .$$

In the first form, PAY^r is a (non-necessarily maximal) payset and it may be $PAY^r = \emptyset$; and i is a potential leader of round r. (However, i may not be the leader ℓ^r. This may indeed happen if if ℓ^r keeps secret his credential and does not reveal himself.)

The second form arises when, in the round-r execution of the BA protocol, all honest players output the default value, which is the empty block B^r_ε in our application. (By definition, the possible outputs of a BA protocol include a default value, generically denoted by $\perp$. See section 3.2.)

Note that, although the paysets are empty in both cases, $B^r = \left(r, \emptyset, SIG_i\left(Q^{r-1}\right), H\left(B^{r-1}\right)\right)$ and B^r_ε are syntactically different blocks and arise in two different situations: respectively, "all went smoothly enough in the execution of the BA protocol", and "something went wrong in the BA protocol, and the default value was output".

Let us now intuitively describe how the generation of block B^r proceeds in round r of *Algorand′*. In the first step, each eligible player, that is, each player $i \in PK^{r-k}$, checks whether he is a potential leader. If this is the case, then i is asked, using of all the payments he has seen so far, and the current blockchain, $B^0, \ldots, B^{r-1}$, to secretly prepare a maximal payment set, PAY^r_i, and secretly assembles his candidate block, $B^r = \left(r, PAY^r_i, SIG_i\left(Q^{r-1}\right), H\left(B^{r-1}\right)\right)$. That,is, not only does he include in B^r_i, as its second component the just prepared payset, but also, as its third component, his own signature of Q^{r-1}, the third component of the last block, B^{r-1}. Finally, he propagate his round-r-step-1 message, $m^{r,1}_i$, which includes (a) his candidate block B^r_i, (b) his proper signature of his candidate block (i.e., his signature of the hash of B^r_i, and (c) his own credential $\sigma^{r,1}_i$, proving that he is indeed a potential verifier of round r.

(Note that, until an honest i produces his message $m^{r,1}_i$, the Adversary has no clue that i is a potential verifier. Should he wish to corrupt honest potential leaders, the Adversary might as well corrupt random honest players. However, once he sees $m^{r,1}_i$, since it contains i's credential, the Adversary knows and could corrupt i, but cannot prevent $m^{r,1}_i$, which is virally propagated, from reaching all users in the system.)

In the second step, each selected verifier $j \in SV^{r,2}$ tries to identify the leader of the round. Specifically, j takes the step-1 credentials, $\sigma^{r,1}_{i_1}, \ldots, \sigma^{r,1}_{i_n}$, contained in the proper step-1 message $m^{r,1}_i$ he has received; hashes all of them, that is, computes $H\left(\sigma^{r,1}_{i_1}\right), \ldots, H\left(\sigma^{r,1}_{i_n}\right)$; finds the credential, $\sigma^{r,1}_{\ell^r_j}$, whose hash is lexicographically minimum; and considers ℓ^r_j to be the leader of round r.

Recall that each considered credential is a digital signature of Q^{r-1}, that $SIG_i\left(r, 1, Q^{r-1}\right)$ is uniquely determined by i and Q^{r-1}, that H is random oracle, and thus that each $H(SIG_i\left(r, 1, Q^{r-1}\right)$ is a random 256-bit long string unique to each potential leader i of round r.

From this we can conclude that, if the 256-bit string Q^{r-1} were itself *randomly and independently selected*, than so would be the hashed credentials of all potential leaders of round r. In fact, all potential leaders are well defined, and so are their credentials (whether actually computed or not). Further, the set of potential leaders of round r is a random subset of the users of round $r-k$, and an honest potential leader i always properly constructs and propagates his message m^r_i, which contains i's credential. Thus, since the percentage of honest users is h, no matter what the malicious potential leaders might do (e.g., reveal or conceal their own credentials), the minimum hashed potential-leader credential belongs to a honest user, who is necessarily identified by everyone to be the leader ℓ^r of the round r. Accordingly, if the 256-bit string Q^{r-1} were itself *randomly and independently selected*, with probability exactly h (a) the leader ℓ^r is honest and (b) $\ell_j = \ell^r$ for all honest step-2 verifiers j.

In reality, the hashed credential are, yes, randomly selected, but depend on Q^{r-1}, which is