1 Introduction: Why Algorand?

Algorand is a cryptocurrency that works to reach consensus on transactions with a system based on Proof of Stake. One of the main attractions of Algorand is its lack of “forks” – that is, unlike in Bitcoin and many other blockchains, users will never\(^1\) see divergent versions of the cryptocurrency’s transaction history.

Algorand achieves this through a Byzantine Agreement protocol called BA⋆, in which users are privately and pseudo-randomly selected to participate in a committee to execute one step of the protocol. The privately selected committee members then broadcast a message which includes their proof of selection, followed by a consensus procedure. This creates a more scalable system than Proof of Work based coins, since the transaction confirmations are much more efficient.

Algorand vs. Bitcoin et al.

Both Algorand and many other commonly used cryptocurrencies consist of cryptographic transactions plus a ledger which records all the transactions that have taken place. The primary difference in Algorand is the way the ledger is updated and modified to accommodate new transactions; while in Bitcoin and Ethereum, miners solve cryptographic puzzles as a “proof of work” in order to add new blocks, Algorand replaces this with a Byzantine Agreement protocol based on proof of stake.

Why does Algorand use a different ledger than Bitcoin, Ethereum, etc? There are a few reasons:

- **“Bitcoin doesn’t scale”**: In Bitcoin, transaction throughput is severely limited: in a given unit of time, there is an upper bound on how many transactions can be processed.

- **Transaction latency**: It takes a long time to confirm a Bitcoin transaction – it would be better if you could confirm a transaction on the order of seconds, instead of in minutes or hours.

- **Wastefulness**: Proof of work wastes huge amounts of electricity and computing power! Coupling a user’s influence to their computational resources is not the only option – a user’s stake in the system (how many coins they own) is also a reasonable metric, and is much more efficient to prove. \(^2\)

- **Forks**: Bitcoin and others allow temporary forks in the ledger, which means it is hard to be sure that a transaction is confirmed even after it has been added to one fork. It would be preferable if we could be sure that every participant has the same view of the ledger.

\(^1\)By “never” what we really mean is “forks are vanishingly unlikely, so that the expected length of time between forks is \(\approx 10^{18}\) seconds – on par with the time elapsed since the big bang”.

\(^2\)For a more detailed discussion of Proof of Work vs. Proof of Stake (and other alternatives), see the 2017 SoK paper.
2 Algorand at a Glance

Threat model

- **Fraction honest:** We assume at least $\frac{2}{3}$ of all coins$^3$ are held by honest users. To be considered honest includes correctness of client software – an honest user is one who follows the Algorand protocol exactly as prescribed.

- **Network synchrony:** Algorand also relies on assumptions about the connectivity of participants, to ensure that the network cannot get too “out-of-sync” due to users failing to participate in the protocol for long intervals of time. Algorand uses the concepts of strong and weak synchrony to describe these assumptions:
  - **Strong synchrony:** Within a set time interval, most honest users ($\sim 95\%$) can send and receive messages from most other honest users ($\sim 95\%$).
  - **Weak synchrony:** In every time period of length $b$ ($\sim$ a few days), there must be a strongly synchronous period of length $s < b$ ($\sim$ a few hours).

The analysis of Algorand is simpler in the case of strong synchrony, but the protocol is designed to guarantee safety as long as at least weak synchrony holds.

Goals

The primary goals of Algorand are improving scalability (increasing transaction throughput), decreasing transaction latency to $\sim 1$ minute per transaction, and making more efficient use of computational resources by avoiding Proof of Work.

Additionally, in order to function reliably, the system should have Safety and Liveness properties, as defined below.

- **Safety:** All honest users agree on the same set of transactions. Specifically, if one honest user accepts transaction $A$, then any future transactions agreed upon by any honest users will be in a log already containing $A$.

- **Liveness:** The system should make progress on confirming valid transactions within a fixed time bound; that is, the process should never get stuck in a deadlock that it can’t resolve.

Setup

Every user is assigned a private key $p_k$ and a secret key $s_k$. The system also includes a log of cryptographic transactions stored as a chain of blocks, and a gossip protocol for broadcasting transactions$^4$.

At each round of the protocol, the following procedure is executed:

1. Cryptographic sortition is used to select a committee (usually on the order of $\sim 1000$ users). Users are elected to the committee with probability proportional to their stake in the system (that is, how many coins they own).

$^3$Food for thought: As of 2007, 1% of the US population owned $> \frac{1}{3}$ of the total wealth. Is the incentive to maintain integrity of the network in order to protect their investments be enough to prevent them from colluding adversarially?

$^4$It was mentioned in class that the gossip protocol itself could be vulnerable to attacks, and its security properties haven’t been investigated as rigorously as they could be. [Alvisi et al., 2007] provides an introduction to the topic.
2. Each member of the committee proposes a block, and the committee attempts to come to a consensus about which proposed block to validate. When multiple blocks are proposed, the committee is to choose the block from the highest weight user.

3. BA* is used to reach consensus on a block (either a block containing correct transactions, or an empty block). During this protocol, new committees are chosen after each voting step, and each time, the votes are broadcast to all users.

3 Committee Selection

Committees are selected via **Cryptographic Sortition**; that is, a cryptographic procedure for randomly selecting a representative set of voters. The procedure has three important properties:

1. It must select users to join the committee with probability proportional to their weights (as measured by their stake in the system – the percentage of all Algorand coins owned by that user)

2. Committees must be unpredictable. This is crucial so that an adversary cannot target the committee members until after they are revealed by broadcasting their votes (at which point the committee is already disbanded).

3. Committee members must be able to privately check whether they are selected, and also be able to provide a checkable proof for others to verify the selection.

**Verifiable Random Functions**

A Verifiable Random Function, or VRF, is a pseudo-random function computed based on a secret key, whose output can be publicly verified to be correct without compromising the secret key.

For a given input $x$, and parameterized by a private key, the Verifiable Random Function can be expressed as

$$VRF_{s_k}(x) \rightarrow (hash, proof)$$

where the hash is deterministic, and unpredictable to anyone without knowledge of $s_k$. Given the hash, $p_k$, and the proof, any user can check that the hash does in fact correspond to value $x$.

**Weighted Proof of Stake**

To implement Proof of Stake cryptographic sortition using VRFs, each user with weight $w$ is thought of as a collection of $w$ sub-users. To decide how many subusers will participate in the round, Algorand computes the VRF based on a shared seed:

$$(hash, proof) \leftarrow VRF_{s_k}(seed || role)$$

and normalizes $hash$ to a value $h \in [0, 1)$:

$$h = \frac{hash}{2^{\text{length}(hash)} - 1}$$
Then to determine the number of subusers chosen, divide \([0, 1)\) into consecutive intervals:

\[
\begin{align*}
0 & \quad B(0) \\
h & \quad B(1) \\
& \cdots \\
1 & \quad B(w)
\end{align*}
\]

So that the width \(B(k)\) of the \(k\)th interval satisfies \(B(x) = \mathbb{P}[\text{exactly } k \text{ of the } w \text{ subusers are chosen}]\). That is, \(B\) is distributed binomially.

From here, the user can simply determine in which interval \(h\) falls, and declare the number of subusers corresponding to that interval, along with \(p_k, \text{ hash}, \) and \(\text{proof}\). With this, other participants can verify that each user has declared the correct number of subusers for that round based on the VRF.

### 4 Byzantine Consensus and BA\(^\star\)

The BA\(^\star\) algorithm is used to answer the question: “Which block should be added to the ledger next?” The algorithm relies on the votes of the rotating committee, and is essentially a more scalable, faster version of the original Byzantine Agreement protocol.

**Byzantine Agreement**

In the simplest protocol, each user broadcasts a vote to every user, and in turn receives the votes from every other user. Byzantine agreement protocols such as PBFT\(^5\) work well for a relatively small, constant number of servers, but it becomes difficult to handle a set of users that may vary over time, and as the set of users becomes large, it is inefficient to share \(O(n^2)\) messages per round. Protocols relying on a fixed list of users are also vulnerable to targeted attacks on known servers, which is important to avoid in Algorand.

**BA\(^\star\)**

In a single step of BA\(^\star\), the following two procedures occur:

1. Set up a new committee and vote; broadcast votes to all users via gossip protocol
2. Users process and verify the message, and tally votes received

Generally, if the votes tallied exceed a certain pre-specified threshold (as a fraction of the total committee members), consensus has been reached on the topic that was voted on. Note that the above specification does not say what the votes represent; this differs among the phases of BA\(^\star\).

---

\(^5\)See [Castro and Liskov, 1999]
Phases of BA⋆

1. **Reduction**: In this phase, the committee selects at most 1 out of the list of possible blocks. In the case that multiple different proposed blocks exist, the committee is to choose the one corresponding to the highest-weight user.

2. **BinaryBA⋆**: In this phase, the committee chooses to either except the single proposed block, or no block.

3. **Finalize**: Sometimes, BA⋆ is unable to guarantee safety, due to asynchrony or malicious users. In this case, a block can reach what is called “tentative consensus”, where the votes in favor of the block reach only the lower of two thresholds. This allows the protocol to move on to new transactions, even if there is a potential fork in the ledger temporarily.

When the network returns to a safe state, the committee will be able to achieve “final consensus” by reaching the higher vote threshold, which certifies that BA⋆ will not reach consensus on any other block that round. Once one block is finalized, all blocks above it in the chain are finalized as well.

\[
\text{procedure } BA⋆(ctx, round, block):
\]

\[
\begin{align*}
&hblock \leftarrow \text{Reduction}(ctx, round, H(block)) \\
&hblock⋆ \leftarrow \text{BinaryBA⋆}(ctx, round, hblock) \\
&// \text{Check if we reached } "\text{final}" \text{ or } "\text{tentative}" \text{ consensus} \\
&r \leftarrow \text{CountVotes}(ctx, round, \text{FINAL}, T_{\text{FINAL}}, r_{\text{FINAL}}, \lambda_{\text{STEP}}) \\
&\text{if } hblock⋆ = r \text{ then} \\
&\quad \text{return } \langle \text{FINAL}, \text{BlockOfHash}(hblock⋆) \rangle \\
&\text{else} \\
&\quad \text{return } \langle \text{TENTATIVE}, \text{BlockOfHash}(hblock⋆) \rangle
\end{align*}
\]

**Figure 3**: (Source: Algorand paper)

Resolving Conflicts

If a user does not receive the expected number of votes, either due to poor network connectivity or an adversary withholding votes, the user continues to wait a pre-determined length of time, and then times out to allow the protocol to continue.

Another possibility is that an adversarial user sends different votes to different users, causing some users to agree on block $A$, and others on block $B$. In this case, BinaryBA⋆ is used to break the tie: Committee members compute a hash function, and choose the least significant bit of the lowest hash to be the “common coin”, which will be broadcast in the next vote. Using the coin-flip value in the place of votes, the community can reach consensus within each successive round of BinaryBA⋆ with probability $\frac{1}{2}$. Because the adversary has no way to increase their probability of having the lowest hash in any given round, they cannot consistently interfere with this process.

5 Discussion

Defense against Common Attacks

- **Sybil Attacks**: Algorand defends against Sybil (or “Pseudonymous”) Attacks in which a single malicious user masquerades as several participants. In Algorand, it is irrelevant whether a user has multiple “accounts” – their influence will always be exactly proportional to the total fraction of coins
they own across all accounts, and the only way to increase their influence is to earn more coins through consensus-verified transactions.

- **DoS Attacks** Algorand is also robust to targeted Denial of Service attacks. If an influential committee member’s identity is known to an adversary, the adversary might be able to target that user and disrupt their connection to the rest of the network. The rotating committees make this impossible – the adversary must take down a large portion of the network in order to be sure it reaches a committee member, since committee membership is secret until after the vote has already taken place.

**Potential pitfalls**

- **Incentives** One potential shortcoming of Algorand comes from its lack of incentives. In Bitcoin and Ethereum, users are incentivized to participate in the mining pool by a reward for validating blocks. In Algorand, participating in the consensus protocol is much less computationally intensive than in Proof of Work based systems, so it is conceivable that users might participate in the protocol without any incentive (other than the incentive to add an extra honest user to help protect against adversaries). However, users might also choose not to participate without some form of reward, which could bring the system to a standstill or leave it vulnerable to adversarial committees.

- **Cost of Storing Ledger** While Algorand is more scalable than Proof of Work systems due to its relatively efficient block validation process, it does not address the memory cost inherent to storing the entire ledger on every individual server. Since Algorand is able to handle a very high transaction throughput, the ledger length could grow quickly to the point where the storage space is too high for casual users to participate.

**Algorand in Practice**

In tests, Algorand achieves its efficiency goal as compared to Bitcoin; Algorand can commit a 2 MByte block in 22 seconds, whereas to commit the same size of data in Bitcoin takes 20 minutes. With a 10 MByte block size, Algorand achieves 125x the throughput of Bitcoin.

The security properties of Algorand have not yet been tested in practice; as of September 2018, Algorand has not yet been deployed as a usable cryptocurrency. However, interested users can participate in its upcoming test network.