Algorand Blockchain Features Specification
Version 1.0

May 30, 2019

This document provides a high-level description of the core features of Version 1.0 of the Algorand blockchain. It is not intended to be a detailed and exhaustive specification review. Rather, it is a simple guide to help users understand primary components in the source code.

Contents

1 Blockchain and Balance Table

The Algorand blockchain is a sequence of *blocks*. Initially, it consists of a single block B^0, called the *genesis block*. New blocks are appended to the blockchain incrementally. Once block B^r is finalized, a consensus protocol is run to generate and finalize block B^{r+1}. We refer to the rth execution of the consensus protocol, which finalizes B^r, as *round* r. A nominal round 0 denotes the initialization of the blockchain, and rounds of negative value denote round 0. For example, $B^{-1} = B^0$.

Block B^r contains a set of *transactions* and a random *seed* Q^r, together with metadata that support cryptographic proofs that everything is valid. Transactions are described in Section 5. The seed Q^r is used in round $r + 2$ as described in Sections 3 and 7.

The Algorand blockchain maintains a set of *accounts* owned by *users*. Each account is owned by a nominally distinct user, so there is a one-one correspondence between users and accounts.

An account is identified by its *account public key (APK)*, a unique 32-byte string in the Algorand blockchain. This string may correspond to a public key from a Ed25519 signature scheme or a hash of a collection of Ed25519 public keys for a multi-signature account.

In any round, a user may be *online* or *offline*; this is called the user's *status*. Online users participate in the consensus protocol to determine the next block; offline users do not (but may still issue transactions). Online users must maintain some additional information, described below.

Each block B^r is associated with a *balance table* S^r, which captures current information about all accounts in the system. S^0 is computed from B^0; S^r is computed from B^r and S^{r-1}.

The balance table S^r contains a balance entry for each user in the system. The balance entry for a user u records the following information:

- APK_u: account public key (the corresponding secret key is kept privately by the user).
- b_u^r: account balance.
- u's status.

A user's balance entry may optionally include additional information, which is needed to participate in the consensus protocol:

- $VRFPK_u$: Verifiable Random Function (VRF) public key used for cryptographic sortition.
- $PartPK_u$: participation signing public key.
- $[v, w]$: A range of rounds for which the user's participation signing key is valid.

VRFPK and *PartPK* are described in more detail in Section 2; as usual, their corresponding secret keys are kept privately by the user.

User status The status of user u is determined as follows:

- A user u is considered *online* at round r if and only if she had a nonempty $VRFPK_u$ specified in her balance entry at round $r - 322$ and her $PartPK_u$ is still valid at round r. (Here, 322 refers to a balance lock-back parameter).
- Otherwise, u is considered *offline* and doesn't participate in round r consensus.

Minimum account balance. Balances in the account entries must maintain a minimum account balance of 0.1 Algo. This restriction is set to mitigate account spamming attacks.

Smallest Algo amount. The smallest Algo amount the system keeps is 1 *micro Algo*, which is 10^{-6} Algo.

2 Key Management

Spending Keys. Algorand uses the Ed25519 signature scheme based on RFC 8032.

Multi-signature Addresses. Algorand supports threshold multi-signature addresses as follows. Suppose a user wants to create a threshold multi-signature address consisting of three keys, where at least two keys must co-sign any transaction. To accomplish this, the user must:

- Sample 3 key pairs $(PK1, SK1), (PK2, SK2), (PK3, SK3)$. (These keys may be samples on independent devices).
- Compute account public key:

$$APK = Hash(PK1, PK2, PK3, threshold = 2)$$

Now, suppose the user has some tokens in the balance table associated with her unique APK. A valid transaction with respect to APK includes the hash pre-image of APK and signatures of the same transaction under at least 2 of the keys $(PK1, PK2, PK3)$. The committee members in the consensus protocol check that the hash of the pre-image matches APK registered on chain, and that a sufficient number of signatures under the public keys are provided.

Verifiable Random Function (VRF) Keys. To participate in the consensus protocol, a user must register a VRF public key in her balance entry, with a corresponding public/secret key pair $(VRFPK, VRFSK)$. We use $VRF(x)$ to denote $VRFSK$'s signature of x. Algorand follows the EC-VRF specifications outlined in draft-irtf-cfrg-vrf-04.

The VRF that Algorand uses is secure against malicious key registration. That is, even if a user maliciously samples and registers a $VRFPK$ on chain, all outputs she obtains are still unique and cannot be biased. This is accomplished partly because of the following: for a fixed $VRFPK$, in round r the VRF is applied on inputs (Q^{r-2}, APK), where $VRFPK$ must have been in the blockchain by round $r - 322$, and APK is the unique account public key of the user.

Participation Keys. Algorand uses the Ed25519 digital signature scheme for participation keys. If a user u wants to participate in the consensus protocol, she samples a key-pair $(PartPK_u, PartSK_u)$ and a collection of ephemeral keys for a range of rounds $[v, w]$

$$(EPK_u^v, ESK_u^v, \ldots, EPK_u^w, ESK_u^w).$$

Using $PartSK_u$, she then signs each ephemeral key EPK_u^i and the corresponding round number i to obtain a signature σ_i. After signing all ephemeral keys, the node deletes $PartSK_u$ and stores the collection:

$$\big((EPK_u^v, ESK_u^v, \sigma_v), \ldots, (EPK_u^w, ESK_u^w, \sigma_w)\big).$$

We use $ESIG_u^r(x)$ to denote ESK_u^r's signature of x. To participate in the consensus protocol the user registers $PartPK_u$ and the range $[v, w]$ for which it generated the ephemeral keys. Now, suppose a user is chosen by cryptographic self-selection to sign a message M at round r.

1. The user uses secret key ESK_u^r to sign M to obtain signature σ_M.
2. The user propagates $(\sigma_M, EPK_u^r, \sigma_r)$ through the network.
3. The tuple (EPK_u^r, σ_r) can be verified with respect to $PartPK_u$ registered on chain to ensure that indeed ESK_u^r was authorized to sign messages for round r.
4. After the user observes the confirmation of B^r, the user deletes ESK_u^r. After deletion, the user can never sign any other message for round r.

3 Cryptographic Self-Selection

For every round r, Algorand runs a fast, partition-resilient Byzantine consensus protocol to agree on a block B^r of transactions. A block is finalized if it has an associated certificate. A certificate for a block consists of a set of valid credentials from committee members (randomly selected via cryptographic self-selection for the round r) and their signatures of the block. Next, we outline how randomness is generated and how cryptographic self-selection is performed.

The random seed Q^r. Each block B^r contains a seed Q^r, which will be used for committee election in the consensus protocol of a future round. Q^{r+2} is computed deterministically by the proposer of B^{r+2} from Q^r and the proposer's own VRF keys.

At genesis block, we set $Q^0 = H(S^0)$, where H is a cryptographic hash function specified by the system. The seeds for future rounds are frequently re-randomized by different block proposers, ensuring liveness and safety.

Cryptographic self-selection for committees using VRFs. A round of the consensus protocol consists of multiple periods (attempts to reach consensus) divided into protocol steps.

For each step, a committee has each of its members secretly self-select at random using the seed Q^{r-2} and the balance table from S^{r-322}. For every round r, all *online* users' balances at round $r - 322$ are added up to compute a total *online* stake $StakeOn_{r-322}$.

To get elected into the committee for round r, period p, step s, an *online* user applies the VRF (keyed on her VRF secret key) to (Q^{r-2}, r, p, s), and gets elected if the output of this computation is small enough, where "small enough" is a function of her balance b_u^{r-322} and the total online stake $StakeOn_{r-322}$.

Committee election is fair, in the sense that for each online user, each token in her balance in round $r - 322$ represents one lottery ticket with a fixed probability of winning. A user with 100 tokens has 100 lottery tickets, whether she keeps them all together or pretends that they are owned by 10 or 100 different users. A user gains no advantage from splitting his stake into more than 1 user.

The cryptographic self-selection for round r uses S^{r-322} partly because users need to commit to their participation VRF keys prior to seeing the seed. (Otherwise, a malicious user could have chosen a VRF key to bias her committee selection probability.)

User timer and clock. Each user u keeps a local timer $timer_u$. Algorand *does not* assume any synchronicity between the users' timers. $timer_u$ may be reset by u and, when queried, it returns the time that passed, in seconds, since the latest reset. Users' timers are only used for the consensus protocol (for instance, a user needs to decide when to move to the next step in the protocol). The consensus protocol assumes that users' timers progress at the same rate for liveness.

4 Security Model

The parameter selection of Algorand blockchain is based on a combination of assumptions: (1) majority of online honest stake, (2) security of cryptographic primitives, and (3) upper bound on message latency.

Honest majority assumption. For every r, at least 80% of $\mathit{StakeOn}_{r-322}$ is *honest* in round r. (Larger committee sizes would be required if we assume a smaller honest majority ratio.)

Cryptographic assumptions. Algorand uses a digital signature scheme for message authentication. It uses a VRF and a hash function, modeled as random oracles in the context of the consensus protocol analysis.

We allow an adversary to perform up to 2^{128} hash operations over the lifetime of the system. This is an extraordinarily large number! With this many hash operations, the adversary can find collisions in SHA-256 function, or to mine 100 billion years' worth of Bitcoin blocks at today's hash rate.

Security Against dynamic corruption. In the Algorand protocol users change their ephemeral participation keys used for every round. That is, after a user signs her message for round r, she deletes the ephemeral key used for signing, and fresh ephemeral keys will be used in future rounds. This allows Algorand to be secure against dynamic corruptions where an adversary may corrupt a user after seeing her propagate a message through the network. (Recall that since users use their VRFs to perform cryptographic self-selection, an adversary does not even know whom to corrupt prior to round r).

Moreover, even if in the future an adversary corrupts all committee members for a round r, as the users holding the supermajority of stakes were honest in round r and erased their ephemeral keys, no two distinct valid blocks can be produced for the same round.

Network model. Algorand guarantees liveness assuming a maximum propagation delay on messages sent through the network. Explicit conditions are specified in Section 7.3. Algorand guarantees safety ("no forks") even in the case of network partitions. When a network partition heals, liveness recovers in linear time against an adversary capable of dynamic corruptions, and in constant time otherwise.

5 Transactions

- *Basic spend transactions.* Users can perform basic transfers from one account to another. Any transaction must be cryptographically signed with the secret key of the sender.

- *Account status change:* online *vs* offline. A special transaction allows a user to register a VRF public key and a participation signing key on chain to participate in the consensus protocol.

  Registered users are considered *online* during rounds for which their participation signing keys are valid.

  A similar special transaction can be issued to move the user to *offline* status. Offline users are removed from participation in consensus.

Any transaction propagated through the network must also specify a round validity interval and a transaction fee.

Transaction validity interval. Each transaction must include a validity interval $[r_1, r_2]$. The transaction is valid and can be included only in blocks in that range. This interval is used to decide whether the transaction has appeared on the blockchain or not: a user only needs to look back into blocks in range $[r_1, r-1]$ to determine if the transaction is valid for block r.

Transaction fees. Each transaction must include a transaction fee of 0.001 Algos or higher. Assuming blocks are not full, a minimal fee should suffice to be included in a block. The minimum fee exists to make spamming more costly for adversaries.

6 Rewards

In every round r, users in the system receive rewards proportional to the balance in their account at round $r - 322$, i.e., as it appears in the balance table S^{r-322}. Rewards are earned regardless of a user's online/offline status in round r or round $r - 322$. These rewards are drawn from a special *RewardsPool* account, which cannot be used for any other purpose. Any account may transfer tokens to *RewardsPool* and, in particular, it is refilled by Algorand Foundation at its discretion. Every half million rounds, the balance of *RewardsPool* is used to determine the rewards distributed in the next half million rounds. Specifically, at each round r divisible by 500,000, the total amount of rewards for each of the next 500,000 rounds (i.e., round r' for $r' \in (r, r + 500,000])$ is determined as $X^r = \frac{b}{500{,}000}$, where $b = b^r_{RewardsPool} - 0.1$, as 0.1 is the minimum account balance.

Rewards earned by a user u in a round are accumulated in the system and are immediately spendable by u. When u is involved in a transaction of any type, say in a round r, the rewards he earned since the last time his balance was updated will be added to his balance, and u's updated balance after round r will earn future rewards since round $r + 322$.

If a user u closes his account at round r, then u will not earn rewards in rounds $r, r+1, \ldots, r+321$.

7 Consensus Protocol

We give the instructions for a generic user u.

Each round r consists of one or more periods. At any point in time, a user u is working on exactly one round and period (r, p). A period (r, p) is further broken into steps (r, p, s), with $0 \leq s \leq 255$. To determine when to act for each step in his current period (r, p), u uses $timer_u$, which is reset whenever u starts a new period.

7.1 Randomly Elected Committees

For each step (r, p, s), a committee is elected to vote. Each step (r, p, s) has two associated parameters, which depend only on s:

- CS_s: The expected committee size for that step.
- CT_t: The committee threshold. A set of votes of total weight CT_s constitutes a *quorum* for step (r, p, s)

For each s, the following list gives the step name and the associated parameters:

Step number s	Step Name	CS_t	CT_s
0	propose	20	N/A
1	soft	2990	2267
2	cert	1550	2267
$3\leq s \leq 252$	next$_{s-3}$	5000	3838
253	late	500	320
254	redo	2400	1768
255	down	6000	4560

7.2 Voting Notation

- $StakeOn_{r-322}$: Total “online” stake at round $r - 322$.
- $q_s^r = \dfrac{CS_s}{StakeOn_{r-322}}$.
- u's (r, p, s)-credential: $VRF_u(Q^{r-2}, r, p, s)$ If $n \geq 1$ then u is elected to vote.
- u's (r, p, s)-credential weight: The unique integer n that satisfies

  $$\sum_{k=n+1}^{\infty} Binomial(b_u^{r-322}; q_s^r) < VRF_u(Q^{r-2}, r, p, s) \leq \sum_{k=n}^{\infty} Binomial(b_u^{r-322}; q_s^r),$$

  If $n \geq 1$ then u is elected to vote.
- u's $(r, p, 0)$-credential value: If n is the credential weight, h_i is the hash of $(credential, i)$, the value is $min_{0\leq i<n}\{h_i\}$
- u's (r, p, s)-vote: $ESIG_u^r(H(B), r, p, s)$; that is, the signature generated by $ESIG_u^r$. $H(B)$ is a string in the range of the hash function H, and intuitively is the hash of a block B. When the round and period are clear by context, we may use the step name to refer to a vote; for example, $(r, p, 2)$-vote may be referred to as a cert-vote. $\perp$ denotes a specific string of the same length as H's output but is not in the range of H. u's vote for $\perp$ is as above but with $H(B)$ replaced by $\perp$.
- (r, p, s)-quorum: A set of CT_s (r, p, s)-votes for the same value.

7.3 Timing Parameters

- λ: intuitively corresponds to the time that it takes a small message (e.g. a vote) to propagate in good network conditions.
- Λ: intuitively corresponds to the time it takes a big message (e.g. a block) to propagate in good network conditions.
- λ_f: the frequency at which the fast partition recovery steps are repeated.

7.4 The Protocol Instructions

User u starts round r period 0 when he first sees a cert-quorum from round $r-1$ for some $H(B^{r-1})$ and the block B^{r-1}. While in period (r,p), if u sees a next-quorum for period $p' \geq p$, then u starts period $p'+1$.

Whenever u starts a new period (including when he starts a new round), he resets $timer_u$, which is used to make decisions on when to vote for each step.

7.4.1 Period (r,p) voting instructions

When user u starts period (r,p), he finishes all previous periods and resets $timer_u$ to 0.

In the step instructions, any message u propagates is accompanied by u's signature using his ephemeral key ESK_u^r and u's credential for that step.

We describe the protocol steps. To simplify the exposition, certain implementation details are omitted.

Step 0: [Proposal] When $timer_u = 0$:

- If $p = 0$ or if $p > 0$ and u has received a next-quorum for $\perp$ from period $(r, p-1)$], then u assembles a new block proposal B_u, then propagates B_u and $H(B_u)$ as separate messages.
- Otherwise, $p > 0$ and u has received a next-quorum for $v = H(B') \neq \perp$ from period $(r, p-1)$, and u propagates v;

Step 1: [Filtering] When $timer_u = 2\lambda$:

- If $p = 0$ or if $p > 0$ and u has received a next-quorum for $\perp$ from period $(r, p-1)$, then u selects from the proposals it has received for this period (if any) the proposal v whose proposer has the minimum credential, and soft-votes v [unless v was a proposal from an earlier period and u has not received a next-quorum for v];
- Otherwise, $p > 0$ and u has received a next-quorum for $v = H(B') \neq \perp$ from period $(r, p-1)$, and u soft-votes v.

Step 2: [Certifying] While $timer_u \in (2\lambda, \max\{4\lambda, \Lambda\})$:

- If u receives a period (r,p) soft-quorum for some value v and a valid block B with $H(B) = v$, then i cert-votes v.

Step $s \in [3, 252]$: [Recovery] When $timer_u = \max\{4\lambda, \Lambda\}$ (when $s = 3$) or $\max\{4\lambda, \Lambda\} + 2^{s-3}\lambda + r$ (when $4 \leq s \leq 252$), where $r \in [0, 2^{s-3}\lambda]$ is sampled uniformly at random:

- If u has seen a valid block B and a (r,p)-soft-quorum for $H(B)$ then u next-votes $H(B)$;
- Otherwise, if $p > 0$ and u has received a next-quorum for $\perp$ from period $(r, p-1)$, then u next-votes $\perp$.
- Otherwise, u has received a next-quorum for $v = H(B') \neq \perp$ from period $(r, p-1)$, and u next-votes v.

Step $s \in [253, 255]$: [Fast recovery] When $timer_u = k\lambda_f + t$ for any positive integer k and $t \in [0, \lambda_f]$, where t is sampled uniformly at random:

- If u has seen a valid block B and a (r, p)-soft-quorum for $H(B)$ then u late-votes $H(B)$ [step 253];
- Otherwise, if $p > 0$ and u has received a next-quorum for $\perp$ from period $(r, p-1)$, then u down-votes $\perp$ [step 255];
- Otherwise, u has received a next-quorum for $v = H(B') \neq \perp$ from period $(r, p-1)$, and u redo-votes v [step 254].

7.5 The seed

At B^0, $Q^0 = H(S^0)$.
Recall that $S^{-r} = S^0$ and $Q^{-r} = Q^0$. A valid block proposal for period (r, p) and proposer u, has a seed Q^r defined as follows for $r > 0$:

```
If $r \equiv 0$ mod 160 or $r \equiv 1$ mod 160:
    If $p = 0$:
        $Q^r = VRF_u(Q^{r-2}, H(B^{r-160}))$
    Else:
        $Q^r = H(Q^{r-2}, H(B^{r-160}))$
Else:
    If $p = 0$:
        $Q^r = VRF_u(Q^{r-2})$
    Else:
        $Q^r = H(Q^{r-2})$
```

7.6 Safety Properties

Here we list a set of properties of the protocol, which imply safety. Each property below refers to quorums and steps within a single period; we list the associated steps for each period. Conditions 4 and 5 are similar, we list them separately to decouple their associated steps.

1. Malicious users cannot, by themselves, generate a quorum for any step.
   Associated steps: $[1, 255]$

2. No two distinct soft-quorums are generated.
   Associated steps: 1

3. If there is a cert-quorum, then there is no down-quorum or next-quorum for $\perp$.
   Associated steps: $[2, 252], 255$

4. If there is a soft-quorum on v, there is no next-quorum for $v' \notin \{v, \perp\}$.
   Associated steps: $1, [3, 252]$

5. If there is a soft-quorum on v, there is no redo-quorum for $v' \neq v$.
   Associated steps: $1, 254$

8 Protocol updates

Algorand Foundation governs the development of the blockchain. Updates to the core blockchain protocol are done in a few steps:

1. Algorand Foundation announces and posts a specification of the protocol in a public git repository. The URL of the repository commit is used as a "protocol version" identifier. This URL must contain a hash that corresponds to the git commit.

2. Any update to the protocol must be supported by the community *on chain* via voting by the block proposers. A block proposer "supports" a proposal by including it as the next protocol version. An update is accepted if for an interval of 10000 rounds, at least 8000 of the finalized blocks support the same next protocol version.

3. After the end of the 10000-round voting interval, users are given another 10000 rounds to update their software. The new specification then takes effect and, from that point on, blocks are produced based on the updated protocol rules.

A Parameters

A.1 Parameter Table

Committee 0 size (CS_0)	20
Committee 1 size/threshold (CS_1/CT_1)	2990/2267
Committee 2 size/threshold (CS_2/CT_2)	1500/1112
Committee $3 \leq s \leq 252$ size/threshold (CS_s/CT_s)	5000/3838
Committee 253 size/threshold (CS_{253}/CT_{253})	500/320
Committee 254 size/threshold (CS_{254}/CT_{254})	2400/1768
Committee 255 size/threshold (CS_{255}/CT_{255})	6000/4560
Minimum account balance	0.1
Minimum transaction fee	0.001
Smallest Algo amount	10^{-6}
Seed lookback	2
Parameter refresh interval	160
Seed lookback	320
$\lambda (seconds)$	2
$\Lambda (seconds)$	17
$\lambda_f (seconds)$	300