Algorand Ledger State Machine Specification

September 5, 2019

Abstract

Algorand replicates a state and the state's history between protocol participants. This state and its history is called the *Algorand Ledger*.

Contents

1 Overview

1.1 Parameters

The Algorand Ledger is parameterized by the following values:

- t_δ, the maximum difference between successive timestamps.
- $T_{\max}$, the length of the *transaction tail*.
- $B_{\max}$, the maximum number of transaction bytes in a block.
- $b_{\min}$, the minimum balance for any address.
- $f_{\min}$, the minimum processing fee for any transaction.
- $V_{\max}$, the maximum length of protocol version strings.
- $N_{\max}$, the maximum length of a transaction note string.
- τ, the number of votes needed to execute a protocol upgrade.
- δ_d, the number of rounds over with an upgrade proposal is open.
- δ_x, the number of rounds needed to prepare for an upgrade.
- ω_r, the rate at which the reward rate is refreshed.
- A, the size of an earning unit.

1.2 States

A *ledger* is a sequence of states which comprise the common information established by some instantiation of the Algorand protocol. A ledger is identified by a string called the *genesis identifier*. Each state consists of the following components:

- The *round* of the state, which indexes into the ledger's sequence of states.
- The *genesis identifier*, which identifies the ledger to which the state belongs.
- The current *protocol version* and the *upgrade state*.
- A *timestamp*, which is informational and identifies when the state was first proposed.
- A *seed*, which is a source of randomness used to establish consensus on the next state.
- The current *reward state*, which describes the policy at which incentives are distributed to participants.
- The current *account state*, which holds account balances and keys for all stakeholding addresses.
- One component of this state is the *transaction tail*, which caches the *transaction sets* (see below) in the last $T_{\max}$ blocks.

1.3 Blocks

A *block* is a data structure which specifies the transition between states. Each block consists of the following components, which correspond to components of the state:

- The block's *round*, which matches the round of the state it is transitioning into. (The block with round 0 is special in that this block specifies not a transition but rather the entire initial state, which is called the *genesis state*. This block is correspondingly called the *genesis block*.)
- The block's *genesis identifier*, which matches the genesis identifier of the states it transitions between.
- The block's *upgrade vote*, which results in the new upgrade state. The block also duplicates the upgrade state of the state it transitions into.
- The block's *timestamp*, which matches the timestamp of the state it transitions into.
- The block's *seed*, which matches the seed of the state it transitions into.
- The block's *reward updates*, which results in the new reward state. The block also duplicates the reward state of the state it transitions into.
- The block's *transaction set*, which represents a set of updates (i.e., *transactions*) to the account state. The block also duplicates a cryptographic commitment to its transaction set.
- The block's *previous hash*, which is the cryptographic hash of the previous block in the sequence. (The previous hash of the genesis block is 0.)

The data in a block is divided between the *block header* and its *block body*. The block body is the block's transaction set, while the block header contains all other data, including a cryptograhpic commitment to the transaction set.

A block is *valid* if each component is also *valid*. (The genesis block is always valid). *Applying* a valid block to a state produces a new state by updating each of its components. The rest of this document defines block validity and state transitions by describing them for each component.

2 Round

The round or *round number* is a 64-bit unsigned integer which indexes into the sequence of states and blocks. The round r of each block is one greater than the round of the previous block. Given a ledger, the round of a block exclusively identifies it.

The rest of this document describes components of states and blocks with respect to some implicit ledger. Thus, the round exclusively describes some component, and we denote the round of a component with a subscript. For instance, the timestamp of state/block r is denoted t_r.

3 Genesis Identifier

The genesis identifier is a short string that identifies an instance of a ledger.

The genesis identifier of a valid block is the identifier of the block in the previous round. In other words, $\text{GenesisID}_{r+1} = \text{GenesisID}_r$.

4 Genesis Hash

The genesis hash is a cryptographic hash of the genesis configuration, used to unambiguously identify an instance of the ledger.

The genesis hash is set in the genesis block (or the block at which an upgrade to a protocol supporting GenesisHash occurs), and must be preserved identically in all subsequent blocks.

5 Protocol Upgrade State

A protocol version v is a string no more than $V_{\max}$ bytes long. It corresponds to parameters used to execute some version of the Algorand protocol.

The upgrade vote in each block consists of a protocol version v_r and a single bit b indicating whether the block proposer supports the given version.

The upgrade state in each block/state consists of the current protocol version v_r^*, the next proposed protocol version v_r', a 64-bit round number s_r counting the number of votes for the next protocol version, a 64-bit round number d_r specifying the deadline for voting on the next protocol version, and a 64-bit round number x_r specifying when the next proposed protocol version would take effect, if passed.

An upgrade vote (v_r, b) is valid given the upgrade state $(v_r^*, v_r', s_r, d_r, x_r)$ if v_r is the empty string or v_r' is the empty string, and either

- $b = 0$ or
- $b = 1$ with $r < d_r$ and either
  - v_r' is not the empty string or
  - v_r is not the empty string.

If the vote is valid, then the new upgrade state is $(v^*_{r+1}, v'_{r+1}, s_{r+1}, d_{r+1}, x_{r+1})$ where

- v^*_{r+1} is v'_r if $r = x_r$ and v^*_r otherwise.
- v'_{r+1} is
  - the empty string if $r = x_r$ or both $r = s_r$ and $s_r + b < \tau$,
  - v_r if v'_r is the empty string, and
  - v'_r otherwise.
- s_{r+1} is
  - 0 if $r = x_r$ or both $r = s_r$ and $s_r + b < \tau$, and
  - $s_r + b$ otherwise
- d_{r+1} is
  - 0 if $r = x_r$ or both $r = s_r$ and $s_r + b < \tau$,
  - $r + \delta_d$ if v'_r is the empty string and v_r is not the empty string, and
  - d_r otherwise.
- x_{r+1} is
  - 0 if $r = x_r$ or both $r = s_r$ and $s_r + b < \tau$,
  - $r + \delta_d + \delta_x$ if v'_r is the empty string and v_r is not the empty string, and
  - x_r otherwise.

6 Timestamp

The timestamp is a 64-bit signed integer. The timestamp is purely informational and states when a block was first proposed, expressed in the number of seconds since 00:00:00 Thursday, 1 January 1970 (UTC).

The timestamp t_{r+1} of a block in round r is valid if:

- $t_r = 0$ or
- $t_{r+1} > t_r$ and $t_{r+1} < t_r + t_\delta$.

7 Cryptographic Seed

The seed is a 256-bit integer. Seeds are validated and updated according to the specification of the Algorand Byzantine Fault Tolerance protocol. The Seed procedure specified there returns the seed from the desired round.

8 Reward State

The reward state consists of three 64-bit unsigned integers: the total amount of money distributed to each earning unit since the genesis state T_r, the amount

of money to be distributed to each earning unit at the next round R_r, and the amount of money left over after distribution B^*_r.

The reward state depends on the I_{pool}, the address of the *incentive pool*, and the functions $\text{Stake}(r, I_{pool})$ and $\text{RewardUnits}(r)$. These are defined as part of the Account State below.

Informally, every ω_r rounds, the rate R_r is updated such that rewards given over the next ω_r rounds will drain the incentive pool, leaving it with the minimum balance b_{min}. The *rewards residue* B^*_r is the amount of leftover rewards that should have been given in the previous round but could not be evenly divided among all reward units. The residue carries over into the rewards to be given in the next round. The actual draining of the incentive pool account is described in the Validity and State Changes section further below.

More formally, let $Z = \text{RewardUnits}(r)$. Given a reward state (T_r, R_r, B^*_r), the new reward state is $(T_{r+1}, R_{r+1}, B^*_{r+1})$ where

- $R_{r+1} = \left\lfloor \frac{\text{Stake}(r, I_{pool}) - B^*_r - b_{min}}{\omega_r} \right\rfloor$ if $R_r \equiv 0 \bmod \omega_r$; $R_{r+1} = R_r$ otherwise,
- $T_{r+1} = T_r + \left\lfloor \frac{R_r}{Z} \right\rfloor$ if $Z \neq 0$; $T_{r+1} = T_r$ otherwise, and
- $B^*_{r+1} = (B^*_r + R_r) \bmod Z$ if $Z \neq 0$; $B^*_{r+1} = B^*_r$ otherwise.

A valid block's reward state matches the expected reward state.

9 Account State

The *balances* are a set of mappings from *addresses*, 256-bit integers, to *balance records*. A *balance record* contains the following fields: the account *raw balance*, the account *status*, the account *rewards base* and *total awarded amount*, and the account *participation keys*.

The account raw balance a_I is a 64-bit unsigned integer which determines how much money the address has.

The account rewards base a'_I and total awarded amount a^*_I are 64-bit unsigned integers.

Combined with the account balance, the reward base and total awarded amount are used to distribute rewards to accounts lazily. The *account stake* is a function which maps a given account and round to the account's balance in that round and is defined as follows:

$$\text{Stake}(r, I) = a_I + (T_r - a'_I) \left\lfloor \frac{a_I}{A} \right\rfloor$$

unless $p_i = 2$ (see below), in which case:

$$\text{Stake}(r, I) = a_I.$$

RewardUnits(r) is a function that computes the total number of whole *earning units* present in a system at round r. A user owns $\left\lfloor \frac{a_I}{A} \right\rfloor$ whole earning units, so the total number of earning units in the system is RewardUnits(r) $= \sum_I \left\lfloor \frac{a_I}{A} \right\rfloor$ for the a_I corresponding to round r.

TODO define how RewardUnits updates.

The account status p_I is an 8-bit unsigned integer which is either 0, 1, or 2. An account status of 0 corresponds to an *offline* account, a status of 1 corresponds to an *online* account, and a status of 2 corresponds to a *non-participating* account.

Combined with the account stake, the account status determines how much *voting stake* an account has, which is a 64-bit unsigned integer defined as follows:

- The account balance, if the account is online.
- 0 otherwise.

The account's participation keys pk are defined in Algorand's specification of participation keys.

An account's participation keys and voting stake from a recent round is returned by the Record procedure in the Byzantine Agreement Protocol.

There exist two special addresses: I_{pool}, the address of the *incentive pool*, and I_f, the address of the *fee sink*. For both of these accounts, $p_I = 2$.

9.1 Transactions

Just as a block represents a transition between two ledger states, a *transaction* Tx represents a transition between two account states. A transaction contains the following fields:

- The *transaction type* TxType, which is a short string that indicates the type of a transaction. The transaction type is either "pay" or "keyreg". A transaction type of "pay" corresponds to a *payment* transaction, while a transaction type of "keyreg" corresponds to a *key registration* transaction.
- The *sender* I, which is an address which identifies the account that authorized the transaction.
- The *fee* f, which is a 64-bit integer that specifies the processing fee the sender pays to execute the transaction.
- The first round r_1 and last round r_2 for which the transaction may be executed.
- The *genesis identifier* GenesisID of the ledger for which this transaction is valid. The GenesisID is optional.
- The *genesis hash* GenesisHash of the ledger for which this transaction is valid. The GenesisHash is required.

- The *note* N, a sequence of bytes with length at most $N_{\max}$ which contains arbitrary data.

A payment transaction additionally has the following fields:

- The *amount* a, which is a 64-bit number that indicates the amount of money to be transferred.
- The *receiver* I', which is an optional address which specifies the receiver of the funds in the transaction.
- The *closing address* I_0, which is an optional address that collects all remaining funds in the account after the transfer to the receiver.

A key registration transaction additionally has the following fields:

- The new participation keys pk, which is an optional set of account participation keys to be registered to the account. If unset, the transaction deregisters the account's keys.
- An optional (boolean) flag nonpart which, when deregistering keys, specifies whether to mark the account offline (if nonpart is false) or nonparticipatory (if nonpart is true).

The cryptographic hash of the fields above is called the *transaction identifier*. This is written as Hash(Tx).

A valid transaction can either be a *signed* transaction or a *multi-signed* transaction. This is determined by the *signature* of a transaction:

- A valid signed transaction's signature is a 64-byte sequence which validates under the sender of the transaction.
- (TODO specify multisignatures)

9.2 ApplyData

Each transaction is associated with some information about how it is applied to the account state. This information is called ApplyData, and contains the following fields:

- The closing amount, ClosingAmount, which specifies how many microalgos were transferred to the closing address.
- The amount of rewards distributed to each of the accounts touched by this transaction. There are three fields ("rs", "rr", and "rc" keys in msgpack encoding), representing the amount of rewards distributed to the sender, receiver, and closing addresses respectively. The fields have integer values representing microalgos. If any of the accounts are the same (e.g., the sender and recipient are the same), then that account received the sum of

the respective reward distributions (i.e., "rs" plus "rr"); in the reference implementation, one of these two fields will be zero in that case.

9.3 Transaction Sequences, Sets, and Tails

Each block contains a *transaction sequence*, an ordered sequence of transactions in that block. The transaction sequence of block r is denoted TxSeq_r. Each valid block contains a *transaction commitment* TxCommit_r which is the commitment to this sequence (a hash of the msgpack encoding of the sequence).

There are two differences between how a standalone transaction is encoded, and how it appears in the block:

- Transactions in a block are encoded in a slightly different way than standalone transactions, for efficiency:

If a transaction contains a GenesisID value, then (1) it must match the block's GenesisID, (2) the transaction's GenesisID value must be omitted from the transaction's msgpack encoding in the block, and (3) the transaction's msgpack encoding in the block must indicate the GenesisID value was omitted by including a key "hgi" with the boolean value true.

Since transactions must include a GenesisHash value, the GenesisHash value of each transaction in a block must match the block's GenesisHash, and the GenesisHash value is omitted from the transaction as encoded in a block.

- Transactions in a block are also augmented with the ApplyData that reflect how that transaction applied to the account state.

The transaction commitment for a block covers the transaction encodings with the changes described above. Individual transaction signatures cover the original encoding of transactions as standalone.

A valid transaction sequence contains no duplicates: each transaction in the transaction sequence appears exactly once. We can call the set of these transactions the *transaction set*. (For convenience, we may also write TxSeq_r to refer unambiguously to this set.) For a block to be valid, its transaction sequence must be valid (i.e., no duplicate transactions may appear there).

All transactions have a *size* in bytes. The size of the transaction Tx is denoted $|\text{Tx}|$. For a block to be valid, the sum of the sizes of each transaction in a transaction sequence must be at most $B_{\max}$; in other words,

$$\sum_{\text{Tx} \in \text{TxSeq}_r} |\text{Tx}| \leq B_{\max}.$$

The transaction tail for a given round r is a set produced from the union of the transaction identifiers of each transaction in the last $T_{\max}$ transaction sets and

is used to detect duplicate transactions. In other words,

$$\text{TxTail}_r = \bigcup_{r-T_{\max} \leq s \leq r-1} \{\text{Hash}(\text{Tx}) | \text{Tx} \in \text{TxSeq}_s\}.$$

As a result, the transaction tail for round $r+1$ is computed as follows:

$$\text{TxTail}_{r+1} = \text{TxTail}_r \backslash \{\text{Hash}(\text{Tx}) | \text{Tx} \in \text{TxSeq}_{r-T_{\max}}\} \cup \{\text{Hash}(\text{Tx}) | \text{Tx} \in \text{TxSeq}_r\}.$$

The transaction tail is part of the ledger state but is distinct from the account state and is not committed to in the block.

9.4 Validity and State Changes

The new account state which results from applying a block is the account state which results from applying each transaction in that block, in sequence. For a block to be valid, each transaction in its transaction sequence must be valid at the block's round r and for the block's genesis identifier GenesisID_B.

For a transaction

$$\text{Tx} = (\text{GenesisID}, \text{TxType}, r_1, r_2, I, I', I_0, f, a, N, \text{pk}, \text{nonpart})$$

to be valid at the intermediate state ρ in round r for the genesis identifier GenesisID_B, the following conditions must all hold:

- It must represent a transition between two valid account states.
- Either $\text{GenesisID} = \text{GenesisID}_B$ or GenesisID is the empty string.
- TxType is either "pay" or "keyreg".
- There are no extra fields that do not correspond to TxType.
- $0 \leq r_2 - r_1 \leq T_{\max}$.
- $r_1 \leq r \leq r_2$.
- $|N| \leq N_{\max}$.
- $I \neq I_{pool}$ and $I \neq 0$.
- $\text{Stake}(r+1, I) \geq f \geq f_{\min}$.
- Exactly one of the signature or the multisignature is present and verifies for Hash(Tx) under I.
- $\text{Hash}(\text{Tx}) \notin \text{TxTail}_r$.
- If TxType is "pay",
  - $I \neq I_k$ or both $I' \neq I_{pool}$ and $I_0 \neq 0$.
  - $\text{Stake}(r+1, I) - f > a$ if $I' \neq I$ and $I' \neq 0$.
  - If $I_0 \neq 0$, then $I_0 \neq I$.
- If TxType is "keyreg",
  - $p_{\rho,I} \neq 2$ (i.e., nonparticipatory accounts may not issue keyreg transactions)
  - If nonpart is true then $\text{pk} = 0$

Given that a transaction is valid, it produces the following updated account state for intermediate state $\rho + 1$:

- For I:
  - If $I_0 \neq 0$ then $a_{\rho+1,I} = a'_{\rho+1,I} = a^*_{\rho+1,I} = p_{\rho+1,I} = \text{pk}_{\rho+1,I} = 0$;
  - otherwise,
    - $a_{\rho+1,I} = \text{Stake}(\rho + 1, I) - a - f$ if $I' \neq I$ and $a_{\rho+1,I} = \text{Stake}(\rho + 1, I) - f$ otherwise.
    - $a'_{\rho+1,I} = T_{r+1}$.
    - $a^*_{\rho+1,I} = a^*_{\rho,I} + (T_{r+1} - a'_{\rho,I}) \left\lfloor \frac{a_{\rho,I}}{A} \right\rfloor$.
    - If TxType is "pay", then $\text{pk}_{\rho+1,I} = \text{pk}_{\rho,I}$ and $p_{\rho+1,I} = p_{\rho,I}$
    - Otherwise (i.e., if TxType is "keyreg"),
      - $\text{pk}_{\rho+1,I} = \text{pk}$
      - $p_{\rho+1,I} = 0$ if pk $= 0$ and nonpart = false
      - $p_{\rho+1,I} = 2$ if pk $= 0$ and nonpart = true
      - $p_{\rho+1,I} = 1$ if pk $\neq 0$.
- For I' if $I \neq I'$ and either $I' \neq 0$ or $a \neq 0$:
  - $a_{\rho+1,I'} = \text{Stake}(\rho + 1, I') + a$.
  - $a'_{\rho+1,I'} = T_{r+1}$.
  - $a^*_{\rho+1,I'} = a^*_{\rho,I'} + (T_{r+1} - a'_{\rho,I'}) \left\lfloor \frac{a_{\rho,I'}}{A} \right\rfloor$.
- For I_0 if $I_0 \neq 0$:
  - $a_{\rho+1,I_0} = \text{Stake}(\rho + 1, I_0) + \text{Stake}(\rho + 1, I) - a - f$.
  - $a'_{\rho+1,I_0} = T_{r+1}$.
  - $a^*_{\rho+1,I_0} = a^*_{\rho,I_0} + (T_{r+1} - a'_{\rho,I_0}) \left\lfloor \frac{a_{\rho,I_0}}{A} \right\rfloor$.
- For all other $I^* \neq I$, the account state is identical to that in view ρ.

TODO define the sequence of intermediate states

The final intermediate account ρ_k state changes the balance of the incentive pool as follows:

$$a_{\rho_k,I_{pool}} = a_{\rho_{k-1},I_{pool}} - R_r(\text{RewardUnits}(r))$$

An account state in the intermediate state $\rho + 1$ and at round r is valid if all following conditions hold:

- For all addresses $I \notin \{I_{pool}, I_f\}$, either $\text{Stake}(\rho + 1, I) = 0$ or $\text{Stake}(\rho + 1, I) \geq b_{\min}$.
- $\sum_I \text{Stake}(\rho + 1, I) = \sum_I \text{Stake}(\rho, I)$.

10 Previous Hash

The previous hash is a cryptographic hash of the previous block header in the sequence of blocks. The sequence of previous hashes in each block header forms an authenticated, linked-list of the reversed sequence.

Let B_r represent the block header in round r, and let H be some cryptographic function. Then the previous hash Prev_{r+1} in the block for round $r+1$ is $\text{Prev}_{r+1} = H(B_r)$.