Algorand
Blockchain Protocol
–
An Overview

By:
Ayse Karaman, Ph. D.
December 2017

Contents

- Algorand as "*a*" blockchain protocol
  External/environmental characteristics
  - PoS, permissionless, decentralized
  - The Adversary
  - Ledger structure
  - Byzantium consensus
- Algorand as *"the"* blockchain protocol

Algorand As "*A*" Blockchain Protocol

Algorand

Brainchild of ***Silvio Micali***
https://people.csail.mit.edu/silvio/

A fast, binary Byzantium agreement protocol
to reach consensus on transactions

PoS – Proof-of-Stake

Role players in block signing

- A set of Algorand users to certify the blocks
- Which users?
  - Stakeholders
  - Selected randomly – cryptographic sortition
- The stake

  Algorand coins currently at hand
  - Chances of taking a role is directly proportional, *not* exponential to the stake
  - The only means a user can influence his chances of role assignment
- Algorand coins are not on hold during selection
- 2/3+ of stakes in honest hands

Permissionless

- Each digital key is an Algorand user
  - Public-key component, *PK* – the user-ID
  - How to join?

    Appear on ledger as a recipient of Algorand coins
- Private

  No forced mapping between Algorand user and real-life user

Decentralized

Fully decentralized

- Nodes fully homogeneous in their functionality
- P2P gossip – underlying message delivery
- Ledger look-up – the only external reference
- Each user listens to the network and acts on protocol rules using its local functionality
  - No reliance on intermediary nodes or external processes
  - No super- or seed-nodes to organize the Algorand network
  - No servers of any kind, not even a universal time
    - Clocks should tick at same intervals on all nodes – Algorand's only time-wise requirement

The Adversary

A powerful one

- **Can**
  - Corrupt users instantly
  - Coordinate bots towards a specific action
  - See all messages instantly

    Anyone can, but the assumed adversary is least effected by propagation delay

The Ledger

Much like the Bitcoin ledger

- Blockchain
  - Temporal sequence of approved blocks
  - Linked by hash-pointers
- Block structure
  - Set of valid transactions
    - Unordered set
    - Valid by transactions on all prior blocks
  - Hash of previous block – the *chain*
  - Also includes round#
    Algorand operational purposes – no effect on use of Ledger
  - Also includes Q_r – random seed
    ditto
  - Certificate – signatures of 2/3+ of verifiers who worked on it
- Query by user-ID
  The public key
  - PoS – look up stakes for role verification

The Ledger – Cont'd

- Immutable
  In order to change a block, change all blocks after it and forge all signatures certifying them
  - Hash-pointer – block content changed
  - Centuries of computing
- "Posted on the sky"
  - Each new block and its credentials circulated for everyone to see
  - Can be viewed by anyone, anytime
    No restrictions, no control

Algorand As "*The*" Blockchain Protocol

Byzantium Agreement

- Consensus in a decentralized network environment
  Potentially faulty setting, uncontrolled – no central authority/intermediary units, open to powerful adversary, time-constraint
- Honesty of 2/3rd+ of the votes for 1/2+ accuracy of the result
  Theoretical bound of Byzantium Agreement
- Fast – much faster than traditional BA systems
- Binary consensus

Rounds

Round – Logical unit of process
Algorand – Sequence of rounds

- Block approval process
  - i) Propose a block
  - ii) Deliberate on it
  - iii) Reach consensus when/if 2/3+ of verification committee agree
- Byzantium agreement – 4-9 steps
- Result:
  - Either a certified block of approved transactions or an empty block
  - Empty block = consensus can't be reached
  - A third possibility – approval of a fake block?
    - i.e. forks/double spending
    - Negligible
- Algorand keeps executing rounds one after the other
  - Each with round# incremented
    round-0 (the genesis round), *round*-1, *round*-2, ...
  - 1 round – 1 block
    Into the immutable ledger and in that order

Role Players

Selected by cryptographic sortition

- Leader – **propose a block (i)**
  - One leader for each round – the first step
  - Sets "what to deliberate on" for "this" round
    - The block – set of transactions to be approved
    - Consistency for binary consensus
    - Dishonest leader sending out multiple blocks to propose
      - No sufficient majority for YES|NO
        No consensus till timeout

How Long Does It Take To Approve A Block?

i.e., ***How long does a round take***?

- 4-9 steps [M17]:
  - 1st step: block proposal by round-leader
  - Latter steps: (dis/)approval of the proposed block by SVs
- Worst case – 9 steps

  dishonest round-leader
  - Propose several blocks
  - SVs can't see a consistent block to agree on
  - "Exit" without an agreement – empty-block
- Likeliest case – 4 steps
  - A single block proposed
  - SV agrees right away
- Bound by network propagation delay

  No other time-boundary – process time negligible

How Long To Approve A Block – Time Parameters

2 parameters:

Λ – round setup

λ – step duration

- Upper limits – large enough to cover propagation delay
- Synchronize steps – decentralized environment
- Start timing for round *r*

So – How Long!?

Round duration = $\Lambda + (2k\text{-}1)\lambda + 2\lambda = \Lambda + (2k+1)\lambda$

k: #steps

$+2\lambda$ for round result be seen

- Suggested values [CM17]

  Λ= 1 minute

  λ = 10 seconds

- Worst case – 9 steps

  4 minutes 10 secs

- Expected case – 4 steps

  2.5 minutes

  "*Will see your xn confirmed within 2.5 minutes*"

  Recall: no forks!

An Underlying Security Fact – Time

- Role players are targets of adversary from the time they are known in the network
  - Corrupt the leader – sub-maximal/illegitimate payset, multiple blocks, manipulate the seed, ...
  - Corrupt the SV members – the judges!

Cryptographic Sortition

A role assignment lottery

Raison d'être

2-fold

1) Unbiased assignments
   - Random – nobody is more equal
   - Proportional to stakes – design requirement

2) Secrecy of role players
   - Not unnecessarily exposed
     Out of sight of the adversary
   - Hiding something?
     - Nope. Info no use to anyone till it's relevant
       Except to the adversary!
     - Relevance: "who's certifying it"
       Right on the message!

The lottery

- Winner gets a role in the step
- Ticket#: randomly assembled from your userID, round#, step#
- Result known only to ticket-holder until he comes out

Cryptographic Sortition – The Process

ticket# = **.H(*sign*(*r*,*s*,Q_r))**

where

r: round#

s: step#

Q_r: random seed // known from the block of previous round

Compose the ticket#:

- Compose message to be signed: $m = (r,s,Q_r)$
- Sign *m* using your static key: $sign(r,s,Q_r)$

Cryptographic Sortition – The Stakes

Modify parameters

A suggested implementation*:

- Compute $K' = \frac{n * a_{r-k,i}}{A_{r-k}}$, $K = \lfloor K' \rfloor$
  where
  - A_{r-k} : total coins in the system k rounds before
  - $a_{r-k,i}$: user i's coins in the system k rounds before
  - $n = |SV|$: targeted number of verifiers
- Add K into ticket#: .H($sign((i, K+1), r, s, Q_r)$)

User i has K votes in (r,s)
+ 1 vote if .H($sign((i, K+1), r, s, Q_r)$) < K'-K

Average in target domain: $A_{AVG} = \frac{A_{r-k}}{n}$

One vote for each A_{AVG} + the partial chances

*"*A More Complex Implementation*", Section 8, [CM17]

Cryptographic Sortition – Properties

Specific to (round, step, user)

- Can't be cheated – role player
  - One ticket per player
  - Random ticket#
- Can't be manipulated – the adversary
  - Can't influence "which ticket"

Player Replaceability

Speak only once

- Send out single message
  - Cast your vote
  - Role credentials
  - No other step duty
- Another set of verifiers the next step
  Cryptographic sortition tells "who"
- Role players do not retain any state info necessary/useful for the steps after
  They know what all else know
  - Replaceability
  - Never a useful target to adversary
- Additional overhead?
  - SV members know themselves already – no time-delay
  - Round-leader selected as candidate leader messages propagate
  - Negligible process overhead

The Keys

3 uses of cryptographic keys

(1) Authentication of payments
Sign that you made that payment
- Static – long-term key

(2) Role credentials – block signing
Get your lottery ticket, prove that you won

Unique Keys

Keys that generate exactly one verifiable signature for a given message

- Static user key (PK, SK)
  PK: public-key component, SK: secret-key component
  - Randomizes for cryptographic sortition – along with hash function
    - the "ticket#" – also hides
    - Q_r – the seed to go into sortition of next round
  - Also serves as user ID
- Verifier $V(s, m, PK)$ knows that "that" signature $s=sign_{PK}(m)$ of "this" message m signed by "this" user PK is unique
  - No other $s'=sign_{PK}(m)$ that can pass $V(s', m, PK)$
- Unique signature
  - Every time $sign_{PK}(m)$ is invoked
  - Every time $sign_{PK\text{-}sub}(m)$ is invoked on a subkey of PK
    - PK can verify $sign_{PK\text{-}sub}(m)$ without having to know $PK\text{-}sub$
- A feasible implementation of *Verifiable Random Functions*[*]

[*]Micali, S., Rabin M. and Vadhan, S., (1999) Verifiable Random Functions, FOCS, New York

Unique Keys – Otherwise ..

Keep generating "ticket#"s till you obtain a winning one

- Keep signing till you get the signature to ..
  - .. get you a role in the round cryptographic sortition
  - .. get you/your bots roles the next round

Look-back Parameter

*Users that are **k**+ rounds old in the system*

- *k*: Look-back parameter
- Role players of current round
  - Restricted to users that existed in the system past *k* rounds
  - Adversary/bots newly injected into system
  - Set aside possibility of adversary manipulating "who"
    - Can influence only if closer to round
- Suggested value minimum *k*=40 [CM17]

Ephemeral Keys

*Single-use keys – Destroy right after **sign()***

- Role players sending out messages during round
  - Have to authenticate their messages – *sign*()
  - Revealing themselves

    Messages tell who the signing role players are
- Destroy key right after *sign*() – can't *sign*() again
  - Even if corrupted by the adversary by then
  - Unique for each round/step – no other key can be issued

Forks

- Can not fork
  - Not while there is 2/3+ honesty
    Legitimate verifiers with valid credentials out of cryptographic sortition process
  - Well, can – with probability 10^{-18}
    Once every few million years!
  - Each block is safely final as soon as it's certified and circulated in the network
- Tie-breakers, just in case
  i) Longest chain
  ii) Non-empty block at most recent round
  iii) Round leader with smaller credential
    The lottery ticket# in cryptographic sortition
  iv) Block with smallest hash value
    Collision-resilient hash – tie should be broken by now !!!

A Fork Scenario – The Setting

No sufficient majority, adversary can swing the votes
The network got partitioned, the adversary got lucky

- Honest votes do not constitute sufficient majority
  against Algorand assumptions!
- The adversary controls a certain amount of votes, say V_{adv}
- Honest votes + V_{adv} constitute sufficient majority

A Fork Scenario – The Act

- Make V_{adv} hold back votes in *Coin-Fixed-To-0* step
- Let Algorand proceed to *Coin-Fixed-To-1* step
- Make V_{adv} vote for empty-block
- Make V_{adv} release votes on valid block

Result: a temporary fork

- Both empty-block and the valid block are certified
- Temporary

  Branch with empty-block discarded the round after
  - Legitimate verifiers on both branches
  - The tie-breaker

Details: Section 10.1 of [CM17]

References

- [CM17] Chen, J., Micali, S., (2017), *ALGORAND*, https://arxiv.org/pdf/1607.01341.pdf

- [G+17] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N., (2017), *Algorand: Scaling Byzantine Agreements for Cryptocurrencies*, https://people.csail.mit.edu/nickolai/papers/gilad-algorand-eprint.pdf