Algorand: Scaling Byzantine Agreements for Cryptocurrencies

By Yossi Gilad, Rotem Hemo, Silvia Micali, Georgios Vlachos, Nickolai Zeldovich
At SOSP, 2017.

Presented by Raunak Kumar and Sishan Long

Outline

- Introduction
- Techniques / innovations
    - High level overview
    - In depth on VRF
    - High level on BA*
- Analysis / evaluation
- Research context
- Context in practice
- Personal critique
- Future work

INTRODUCTION

Current Cryptocurrencies

- Cryptocurrencies have a lot of applications.
    - Smart contracts, simplified currency conversions, etc.
- However, most of them suffer from a trade-off.
    - **Trade-off between latency vs confidence in transaction.**
- Bitcoin uses proof-of-work to address the double-spending problem.
    - On average, takes 10 minutes to generate a block.
    - Recommended to wait ~6 blocks before confirming transaction.
    - So on average, takes around an hour to confirm a transaction!
    - Can also have forks in the blockchain.

Algorand

- What is Algorand?
  - New cryptocurrency that aims to provide:
    1. **Low latency.**
    2. **Low probability of forking.**
  - Uses **proof of stake**, not proof of work.
- Why is it interesting?
  - **Use of randomness** - verifiable random functions.
  - **Consensus at scale** - committees within BA* byzantine agreement protocol.

Overview

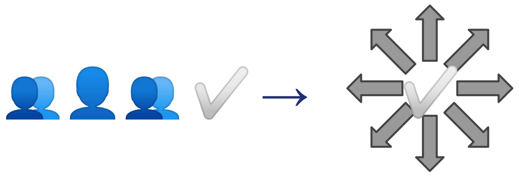
Really Really High Bird’s-eye View

- In every round we add a block to the blockchain.
- Every round has multiple steps.
- In each round:
  - Choose block proposers. They gossip.
  - Each user keeps track of highest priority block received.
  - Run BA* for multiple steps with different committee each time.
  - Run till we agree on a block to add or add empty block.
  - Some issues - final vs tentative consensus, forking.

Algorand's Two-Step Road to Consensus

Step 1: Proposal

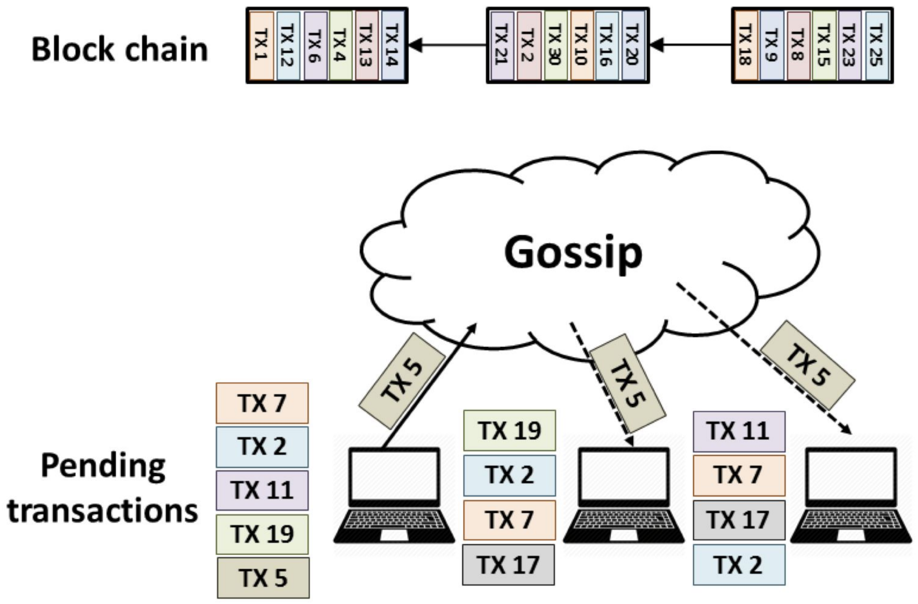
- **One token** is selected **randomly**.
- Its corresponding public key becomes to known to all.
- The corresponding user **proposes**, **signs** and **propagates** a new block.



Step 2: Agreement

- 1000 users are **randomly selected**, their keys become to known to all.
- These users quickly **reach agreement** on and **sign** the proposed blocks.
- **Sufficiently signed** blocks are valid and are propagated.

Gossip



An overview of transaction flow in Algorand.

- Send messages to peers.
- Do not relay same message twice.
- Each user has a block of pending transactions.

Block Proposal

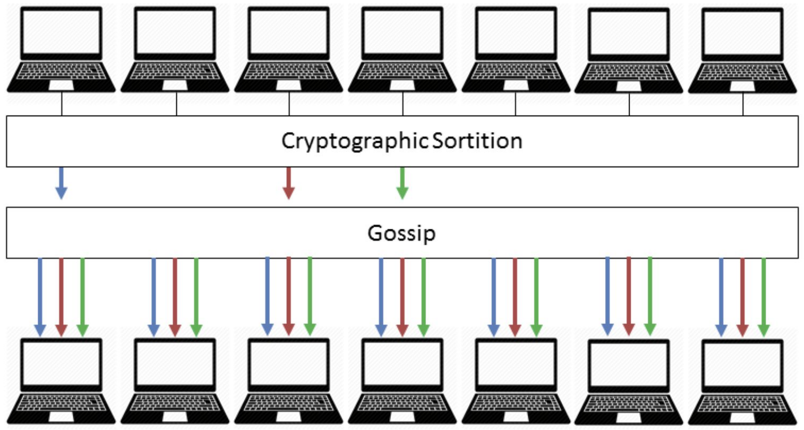
- Small number of users chosen to propose a block.
- Each user proposes a block, its priority and a proof.
- Distribute this information using the gossip protocol.
    - Users wait for a certain amount of time to receive a block.
    - There could be multiple proposers.
    - Priority determines which block everyone should adopt.

👥👤👥👤👥→ 👤→ ⬜⬜✴️

Agreement Using BA*

- Each user initializes BA* with highest priority block that they received.

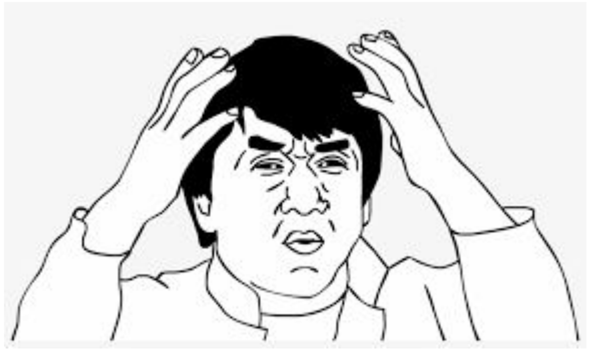
- In each step:
  - Select a committee.
  - Committee members broadcast message.
  - Repeat until enough committee members reach consensus.
  - No private state except private key -> committee can be replaced after each step.
  - Terminates in 4 - 13 (expected) steps.



Cryptographic Sortition

Sortition???

- What does sortition even mean?



Sortition???

- What does sortition even mean?
    - “the action of selecting or determining something by the casting or drawing of lots”

Sortition???

- What does sortition even mean?
  - “the action of selecting or determining something by the casting or drawing of lots”
- Why is this relevant to us?
  - Need to pick block proposers and committee members.

Sortition for Proposal and Committee

- Sortition is great!
- For example, if we want to select committee members:
  - Toss a coin with heads probability proportional to the amount of money a person has.
  - If heads then select person.

Sortition for Proposal and Committee

- Sortition is great!
- For example, if we want to select committee members:
  - Toss a coin with heads probability proportional to the amount of money a person has.
  - If heads then select person.
- What's the problem?
  - Adversary can target committee members! :(

Cryptographic Sortition

- Define:
  - w_i = weight of user i, W = total weight. (weight = money)
  - pk_i = public key of user i, sk_i = private key of user i.
- Goal: select user i proportional to w_i / W in a secure way.

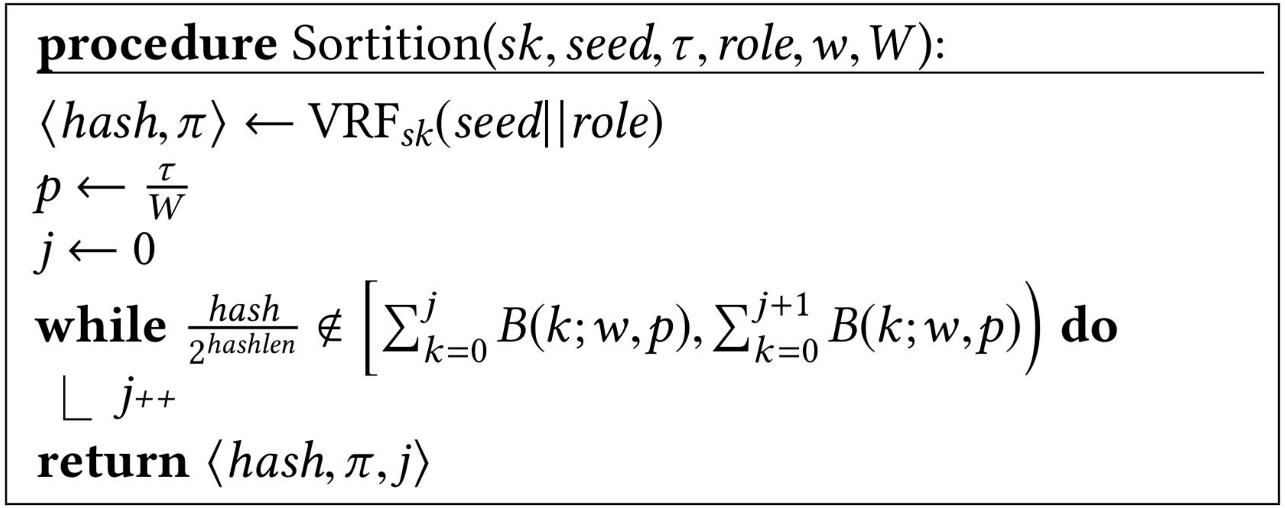
Cryptographic Sortition

- Define:
  - w_i = weight of user i, W = total weight. (weight = money)
  - pk_i = public key of user i, sk_i = private key of user i.
- Goal: select user i proportional to w_i / W in a secure way.
- Key tool: **Verifiable random functions (VRFs)**.

Cryptographic Sortition

- Define:
  - w_i = weight of user i, W = total weight. (weight = money)
  - pk_i = public key of user i, sk_i = private key of user i.
- Goal: select user i proportional to w_i / W in a secure way.
- Key tool: **Verifiable random functions (VRFs)**.
  - I have input string x.
  - VRF_sk(x) = (hash, pi).
  - hash is a hashlen-bit long string determined by sk and x.
  - hash is ~uniformly distributed between 0 and 2^(hashlen) - 1.
  - If you know pk, then using pi you can check hash is valid output for x.

Selection

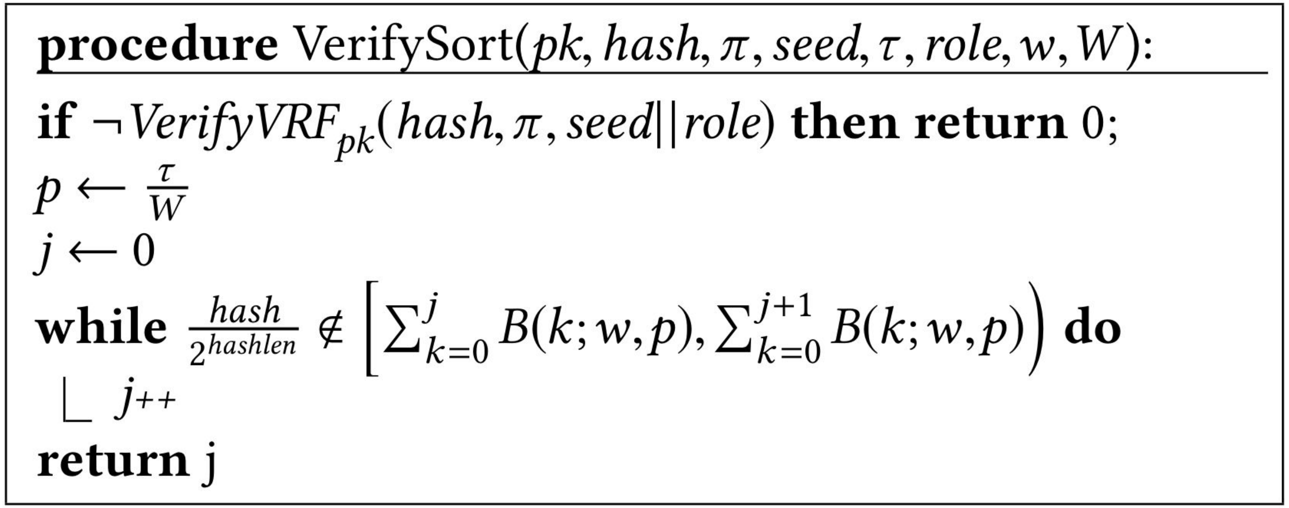


```
procedure Sortition($sk, seed, \tau, role, w, W$):
    $\langle hash, \pi \rangle \leftarrow \mathrm{VRF}_{sk}(seed || role)$
    $p \leftarrow \frac{\tau}{W}$
    $j \leftarrow 0$
    while $\frac{hash}{2^{hashlen}} \notin \left[ \sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p) \right)$ do
        $j$++
    return $\langle hash, \pi, j \rangle$
```

The cryptographic sortition algorithm.

- seed = publicly known (more details coming up)
- tau = expected number of users
- role = proposer, committtte, etc.

Verification



```
procedure VerifySort($pk, hash, \pi, seed, \tau, role, w, W$):
if $\neg VerifyVRF_{pk}(hash, \pi, seed||role)$ then return 0;
$p \leftarrow \frac{\tau}{W}$
$j \leftarrow 0$
while $\frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p)\right)$ do
    $j$++
return j
```

Pseudocode for verifying sortition of a user with public key pk

How to choose the seed?

- Seed should be publicly known, but cannot be controlled by the adversary.
- seed_0: Generate using distributed random number generation.
- seed_r:
  - During block proposal, also compute (seed_r, pi) = VRF_sk(seed_r-1 || r).
  - Include this in every block.
  - Everyone knows seed_r at the start of round r.
  - However, if block does not contain a valid seed or has invalid transactions, use seed_r = H(seed_{r-1} || r), where H is a cryptographic hash function.

How to choose the seed?

- Seed should be publicly known, but cannot be controlled by the adversary.
- seed_0: Generate using distributed random number generation.
- seed_r:
  - During block proposal, also compute (seed_r, pi) = VRF_sk(seed_r-1 || r).
  - Include this in every block.
  - Everyone knows seed_r at the start of round r.
  - However, if block does not contain a valid seed or has invalid transactions, use seed_r = H(seed_{r-1} || r), where H is a cryptographic hash function.
- But the seed is refreshed every R rounds…
  - Compute seed_r in every round.
  - But use seed_{r - 1 - (r % R)} as input to sortition.

How to use the seed?

Round	Compute seed_r	Use seed_{r-1 - (r%R)}
1	seed_1, pi = VRF_sk(seed_0 || 1)	seed_{-1}
2	seed_2, pi = VRF_sk(seed_1 || 2)	seed_{-1}
3	seed_3, pi = VRF_sk(seed_2 || 3)	seed_{-1}
4	seed_4, pi = VRF_sk(seed_3 || 4)	seed_{-1}
5	seed_5, pi = VRF_sk(seed_4 || 5)	seed_4
6	seed_6, pi = VRF_sk(seed_5 || 6)	seed_4
7	seed_7, pi = VRF_sk(seed_6 || 7)	seed_4
8	seed_8, pi = VRF_sk(seed_7 || 8)	seed_4

R = 5

Why?

- Suppose network is not strongly synchronous
    - So adversary has complete control over links.
    - Can drop block proposals and force users to agree on empty blocks.
    - But gets users to compute future selection seeds!

Why?

- Suppose network is not strongly synchronous
  - So adversary has complete control over links.
  - Can drop block proposals and force users to agree on empty blocks.
  - But gets users to compute future selection seeds!
- Instead, in round r
  - Check timestamp of block in round r - 1 - (r % R).
  - **Use keys and weights from last block created b-time before that block.**
    - Lower bound on length of strongly synchronous period should allow for sufficiently many blocks to be created in order to ensure at least one block was honest whp.
    - To ensure prob. of failure <= F, need # blocks O(log(1 / F)).

BA*

BA*

- Execution consists of **2 phases**:
  - First phase: reduce to problem of agreeing on a block to one of 2 options.
  - Second phase: reach agreement on 1 options (either proposed block or empty block).

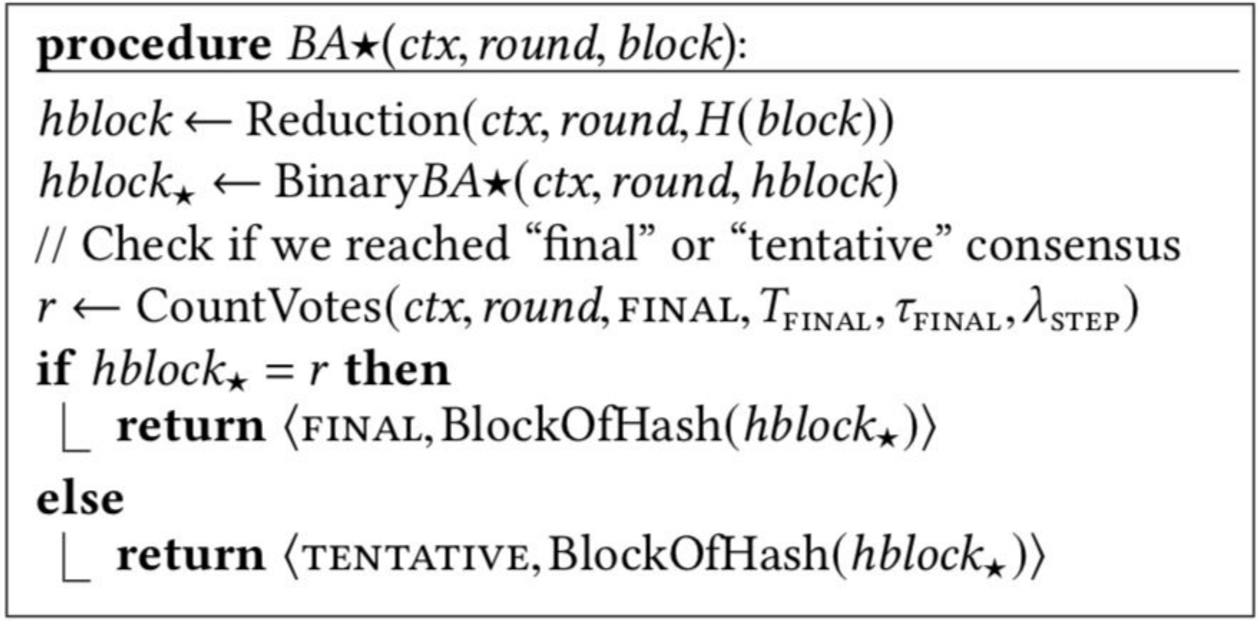
BA*

- Execution consists of **2 phases**:
  - First phase: reduce to problem of agreeing on a block to one of 2 options.
  - Second phase: reach agreement on 1 options (either proposed block or empty block).
- Each phase consists of several steps:
  - First phase takes 2 steps.
  - Second phases takes 2 - 11 (expected) steps.

BA*

- Execution consists of **2 phases**:
  - First phase: reduce to problem of agreeing on a block to one of 2 options.
  - Second phase: reach agreement on 1 options (either proposed block or empty block).
- Each phase consists of several steps:
  - First phase takes 2 steps.
  - Second phases takes 2 - 11 (expected) steps.
- Each step:
  - Committee members cast votes for some value.
  - All users count the votes.
  - If a user receives more than a threshold of votes for a value, vote for that in the next step.

BA*



```
procedure $BA\star(ctx, round, block)$:
    $hblock \leftarrow \text{Reduction}(ctx, round, H(block))$
    $hblock_\star \leftarrow \text{Binary}BA\star(ctx, round, hblock)$
    // Check if we reached “final” or “tentative” consensus
    $r \leftarrow \text{CountVotes}(ctx, round, \text{FINAL}, T_{\text{FINAL}}, \tau_{\text{FINAL}}, \lambda_{\text{STEP}})$
    if $hblock_\star = r$ then
        return $\langle \text{FINAL}, \text{BlockOfHash}(hblock_\star) \rangle$
    else
        return $\langle \text{TENTATIVE}, \text{BlockOfHash}(hblock_\star) \rangle$
```

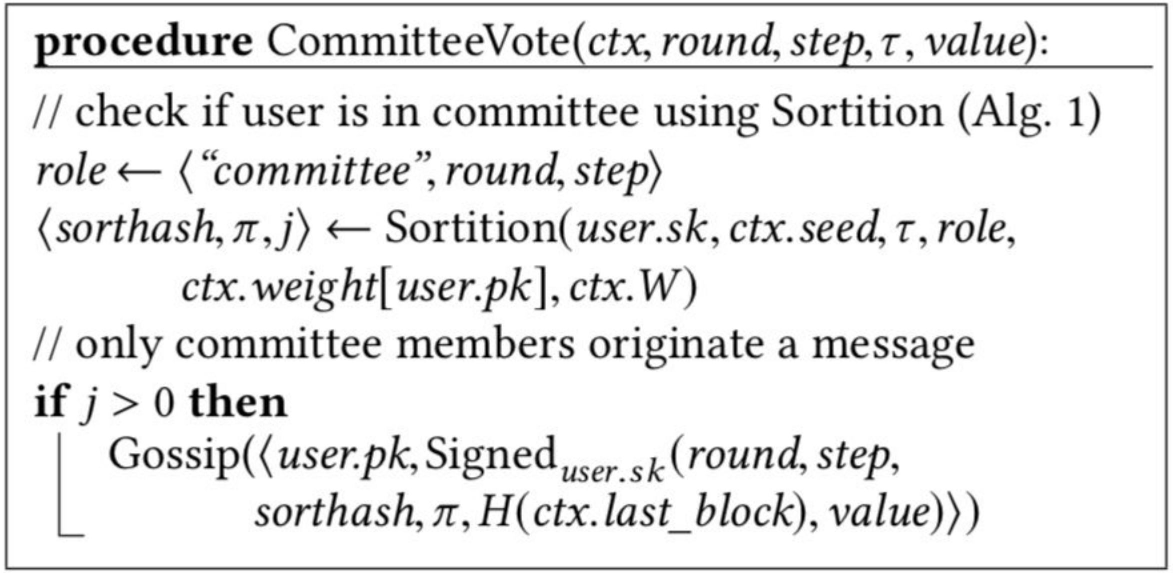
Algorithm 3: Running *BA⋆* for the next *round*, with a proposed *block*. H is a cryptographic hash function.

Voting

Each user

- Checks if part of committee (sortition).
- If chosen, gossip signed message containing a value.
    - Value is typically the hash of some block.
    - Also include the hash of previous block.

Voting



```
procedure CommitteeVote($ctx, round, step, \tau, value$):
// check if user is in committee using Sortition (Alg. 1)
$role \leftarrow \langle$"committee"$, round, step\rangle$
$\langle sorthash, \pi, j\rangle \leftarrow$ Sortition($user.sk, ctx.seed, \tau, role,$
        $ctx.weight[user.pk], ctx.W$)
// only committee members originate a message
if $j > 0$ then
    Gossip($\langle user.pk$, Signed$_{user.sk}(round, step,$
            $sorthash, \pi, H(ctx.last\_block), value)\rangle$)
```

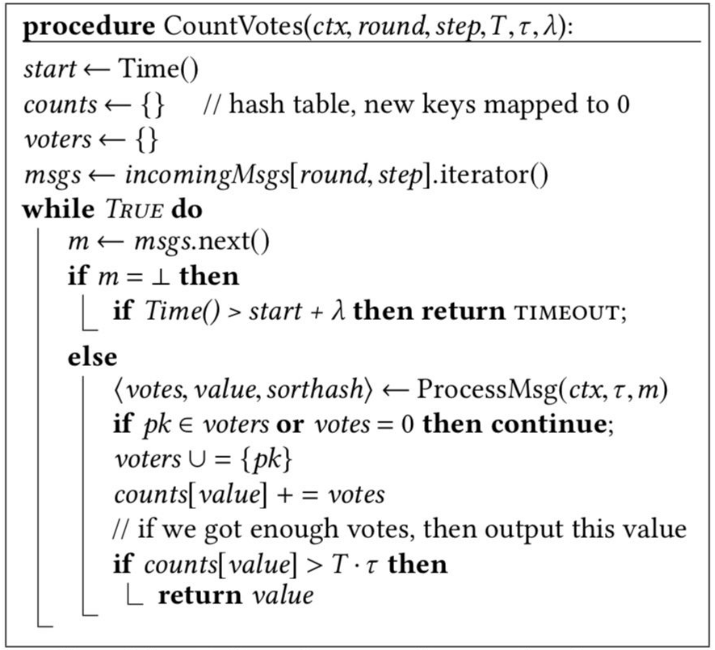
Algorithm 4: Voting for *value* by committee members. *user.sk* and *user.pk* are the user's private and public keys.

Counting Votes

Each user

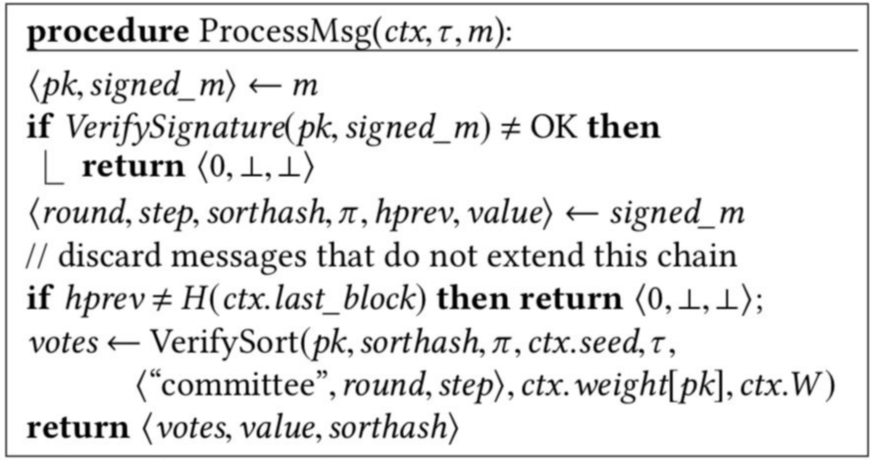
- Reads messages from current round and step.
- Process votes for every message.
    - Get number of votes for message.
    - If it exceeds a certain threshold then vote for this value.
    - Otherwise return TIMEOUT.

Counting Votes



```
**procedure** CountVotes($ctx, round, step, T, \tau, \lambda$):
$start \leftarrow$ Time()
$counts \leftarrow \{\}$    // hash table, new keys mapped to 0
$voters \leftarrow \{\}$
$msgs \leftarrow incomingMsgs[round, step]$.iterator()
**while** TRUE **do**
    $m \leftarrow msgs$.next()
    **if** $m = \perp$ **then**
        **if** $Time() > start + \lambda$ **then return** TIMEOUT;
    **else**
        $\langle votes, value, sorthash \rangle \leftarrow$ ProcessMsg($ctx, \tau, m$)
        **if** $pk \in voters$ **or** $votes = 0$ **then continue**;
        $voters \cup = \{pk\}$
        $counts[value] += votes$
        // if we got enough votes, then output this value
        **if** $counts[value] > T \cdot \tau$ **then**
            **return** $value$
```

Algorithm 5: Counting votes for *round* and *step*.



```
**procedure** ProcessMsg($ctx, \tau, m$):
$\langle pk, signed\_m \rangle \leftarrow m$
**if** $VerifySignature(pk, signed\_m) \neq$ OK **then**
    **return** $\langle 0, \perp, \perp \rangle$
$\langle round, step, sorthash, \pi, hprev, value \rangle \leftarrow signed\_m$
// discard messages that do not extend this chain
**if** $hprev \neq H(ctx.last\_block)$ **then return** $\langle 0, \perp, \perp \rangle$;
$votes \leftarrow$ VerifySort($pk, sorthash, \pi, ctx.seed, \tau,$
        $\langle$"committee"$, round, step \rangle, ctx.weight[pk], ctx.W$)
**return** $\langle votes, value, sorthash \rangle$
```

Algorithm 6: Validating incoming vote message *m*.

Reduction

- Reduce reaching consensus on arbitrary value to **one of 2 options**:
  - specific value (proposed block hash)
  - hash of empty block.

Reduction

- Reduce reaching consensus on arbitrary value to **one of 2 options**:
  - specific value (proposed block hash),
  - hash of empty block.
- First step: each committee member votes for hash of block.

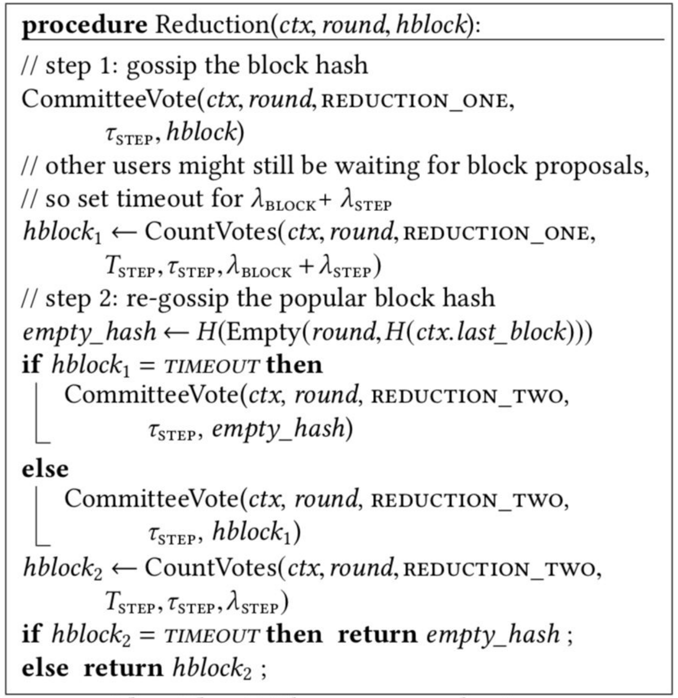
Reduction

- Reduce reaching consensus on arbitrary value to **one of 2 options**:
  - specific value (proposed block hash),
  - hash of empty block.
- First step: each committee member votes for hash of block.
- Second step: vote for hash that received at least some threshold of votes.
  - If no such block, then vote for empty hash.

Reduction

- Reduce reaching consensus on arbitrary value to **one of 2 options**:
  - specific value (proposed block hash),
  - hash of empty block.
- First step: each committee member votes for hash of block.
- Second step: vote for hash that received at least some threshold of votes.
  - If no such block, then vote for empty hash.
- **Reduction ensures that there is at most 1 non-empty block that can be returned for all honest users.**

Reduction



```
procedure Reduction($ctx$, $round$, $hblock$):
// step 1: gossip the block hash
CommitteeVote($ctx$, $round$, REDUCTION_ONE,
        $\tau_{\text{STEP}}$, $hblock$)
// other users might still be waiting for block proposals,
// so set timeout for $\lambda_{\text{BLOCK}} + \lambda_{\text{STEP}}$
$hblock_1 \leftarrow$ CountVotes($ctx$, $round$, REDUCTION_ONE,
        $T_{\text{STEP}}, \tau_{\text{STEP}}, \lambda_{\text{BLOCK}} + \lambda_{\text{STEP}}$)
// step 2: re-gossip the popular block hash
$empty\_hash \leftarrow H(\text{Empty}(round, H(ctx.last\_block)))$
if $hblock_1 = $ TIMEOUT then
    CommitteeVote($ctx$, $round$, REDUCTION_TWO,
            $\tau_{\text{STEP}}$, $empty\_hash$)
else
    CommitteeVote($ctx$, $round$, REDUCTION_TWO,
            $\tau_{\text{STEP}}$, $hblock_1$)
$hblock_2 \leftarrow$ CountVotes($ctx$, $round$, REDUCTION_TWO,
        $T_{\text{STEP}}, \tau_{\text{STEP}}, \lambda_{\text{STEP}}$)
if $hblock_2 = $ TIMEOUT then return $empty\_hash$ ;
else return $hblock_2$ ;
```

Algorithm 7: The two-step reduction.

Binary BA*

Not enough time to cover Binary BA* - please refer to the paper.

Experiments

Implementation Parameters

Parameter	Meaning	Value
h	assumed fraction of honest weighted users	80%
R	seed refresh interval (# of rounds)	1,000 (§5.2)
τ_{PROPOSER}	expected # of block proposers	26 (§B.1)
τ_{STEP}	expected # of committee members	2,000 (§B.2)
T_{STEP}	threshold of τ_{STEP} for $BA\star$	68.5% (§B.2)
τ_{FINAL}	expected # of final committee members	10,000 (§C.1)
T_{FINAL}	threshold of τ_{FINAL} for $BA\star$	74% (§C.1)
MaxSteps	maximum number of steps in Binary$BA\star$	150 (§C.1)
$\lambda_{\text{PRIORITY}}$	time to gossip sortition proofs	5 seconds
λ_{BLOCK}	timeout for receiving a block	1 minute
λ_{STEP}	timeout for $BA\star$ step	20 seconds
λ_{STEPVAR}	estimate of $BA\star$ completion time variance	5 seconds

Implementation Parameters

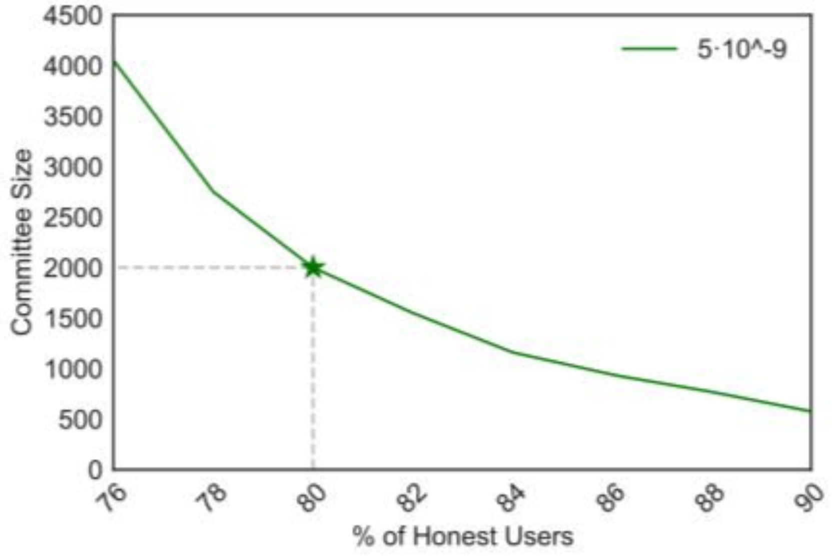
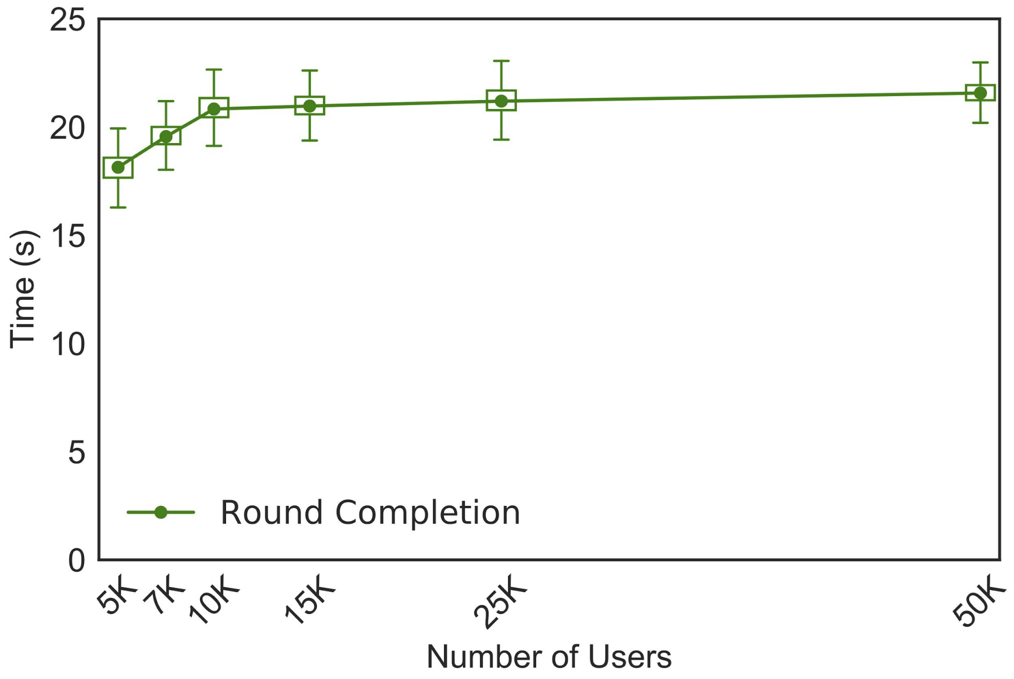


Figure 3: The committee size, τ, sufficient to limit the probability of violating safety to 5×10^{-9}. The x-axis specifies h, the weighted fraction of honest users. $\star$ marks the parameters selected in our implementation.

Experimental Setup

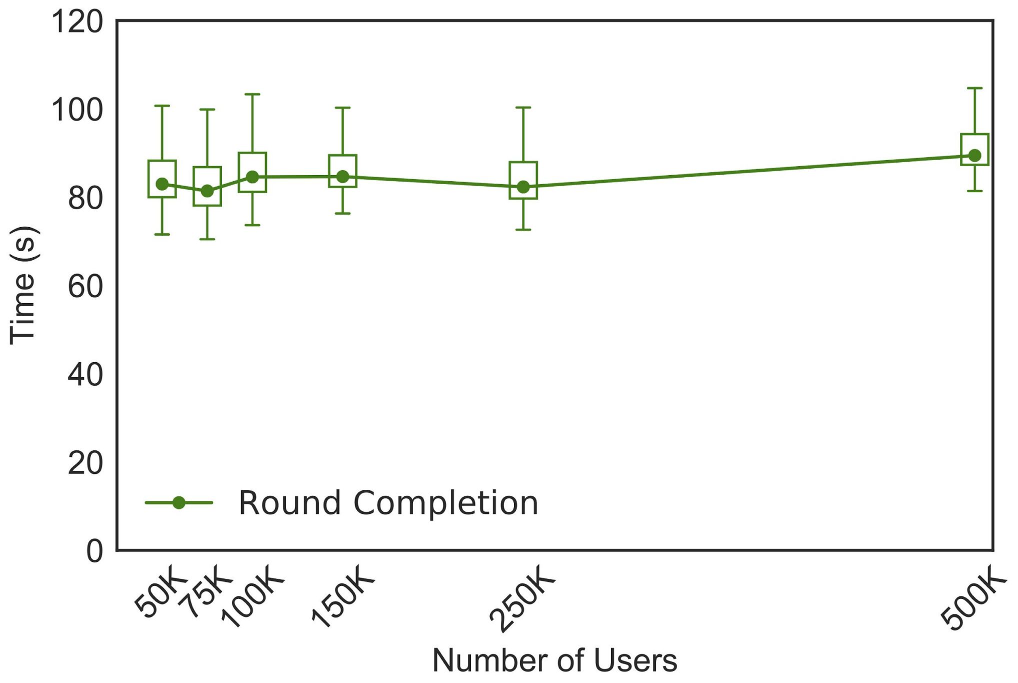
- 1000 m4.2xlarge VMs on Amazon EC2
  - 8 cores, 1Gbps network throughput.
  - Distributed across 20 cities around the world.
- Usually 50, sometimes 500, users per VM.
  - Each user can use <= 20 Mbps bandwidth.
  - Equal share of money.
- 1 MB block.

Latency



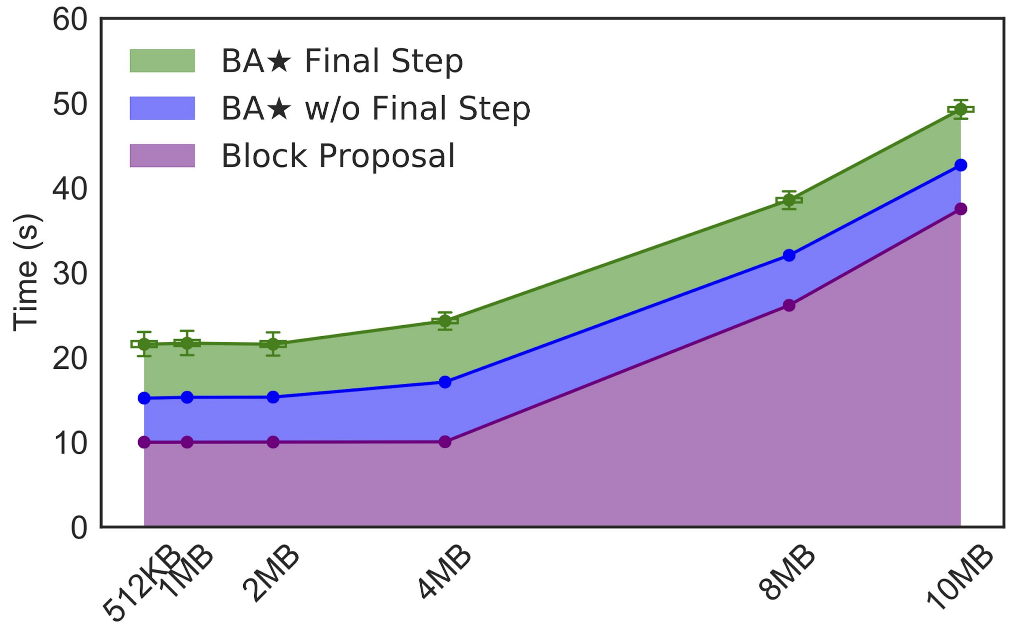
Latency for one round of Algorand, with 5,000 to 50,000 users.

Latency



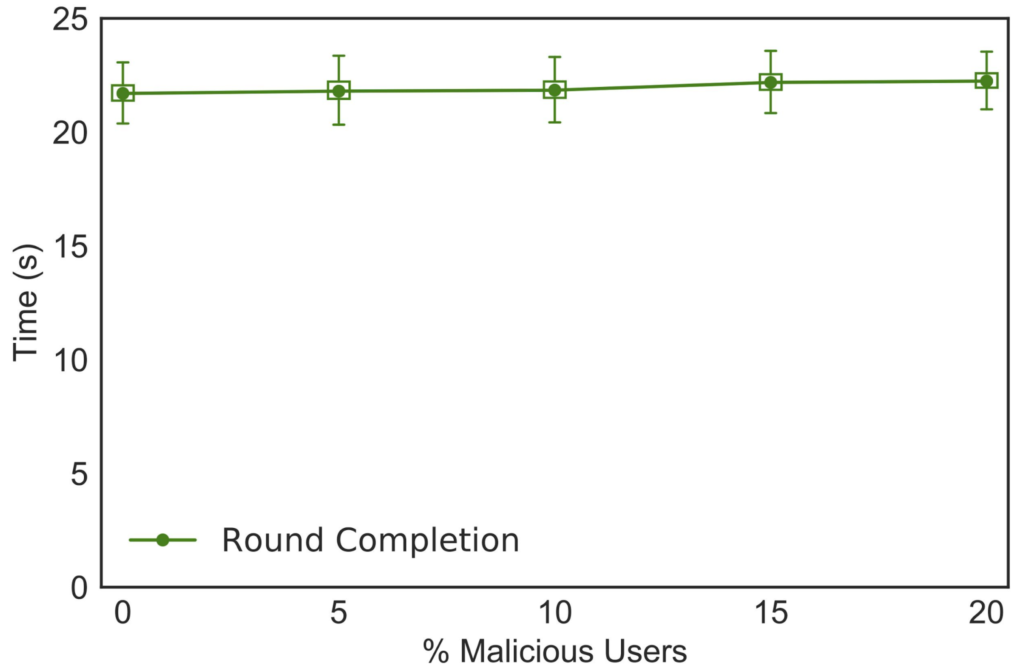
Latency for one round of Algorand in a configuration with 500 users per VM, using 100 to 1,000 VMs.

Throughput



Latency for one round of Algorand as a function of the block size.

Malicious Users



Latency for one round of Algorand with a varying fraction of malicious users, out of a total of 50,000 users.

Related Work

Ouroboros

- Some users are elected to propose blocks.
- Each round is subdivided into slots, each slot has one block proposer.

Ouroboros

- Some users are elected to propose blocks.
- Each round is subdivided into slots, each slot has one block proposer.
- Use of randomness
  - A random seed is generated in the previous round r-1.
  - Follow-the-Satoshi algorithm chooses some coins out of all coins.
  - The new block proposers are the owners of these coins.
- No committee election.

Ouroboros

- Use of randomness
  - A random seed is generated in the previous round r-1.
  - Follow-the-Satoshi algorithm chooses some coins out of all coins, their owners are the new block proposers.

Algorand

- All users participate in the lottery and can win the ticket.
- Use of randomness
  - Every user runs VRF with a public random seed.
  - They produce uniformly distributed random values, others can verify.

Ouroboros

- No committee election.
- Longest chain rule.
- Slot leaders are known during the round.

Algorand

- The user may be selected to propose a block or be selected to be a member of the committee at a certain step of BA*.
- Committees are constantly changing.

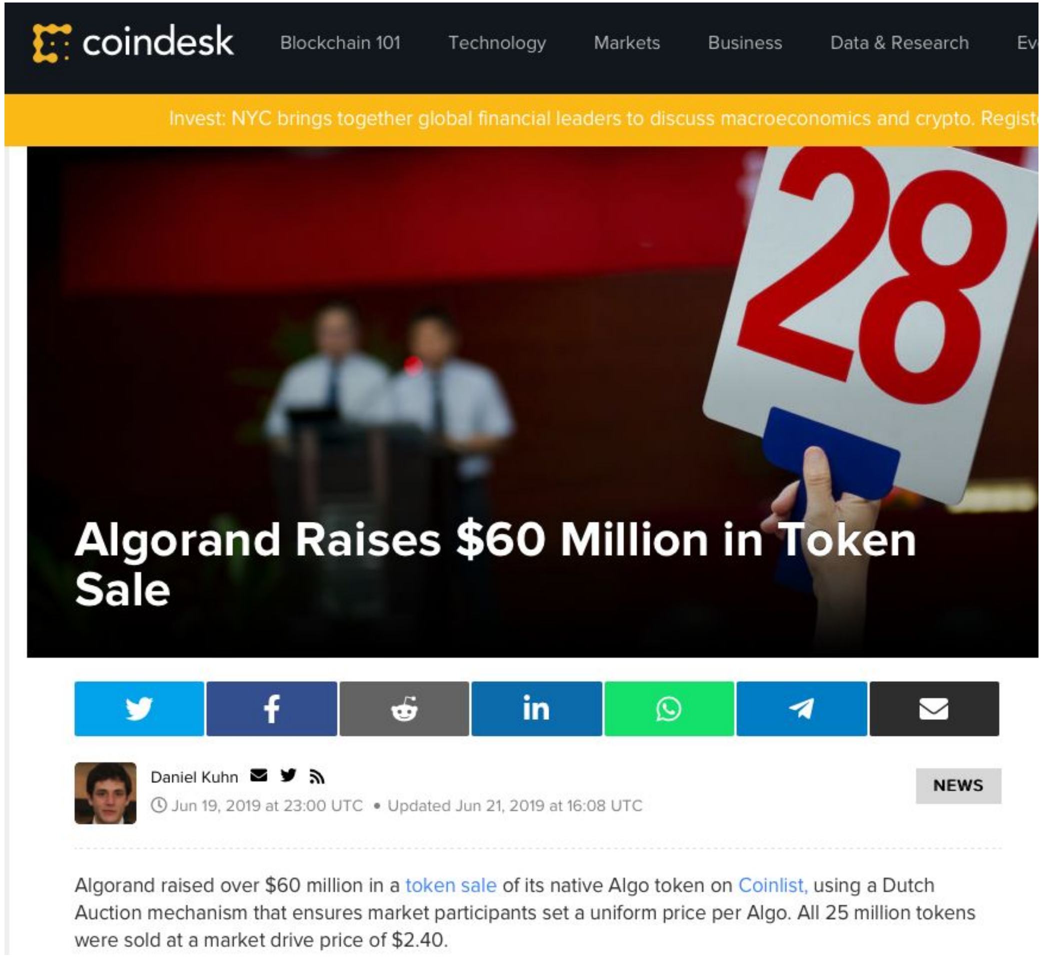
Ouroboros vs Algorand

- Security
    - Ouroboros assumes that honest users can communicate within some bounded delay.
    - Algorand assumes that the adversary may temporarily fully control the network and immediately corrupt users in targeted attacks.
- Rate of Producing Blocks
- Time Synchronization

Related Work

- Bitcoin:
  - Possibility of forks → long confirmation time
  - Network partition
- Delegated PoS:
  - The committee is not random and will not change in a long time.
  - DOS
- Bonded PoS:
  - They are willing to give up their money and become adversary to gain more money.
  - DOS
- Hybrid consensus:
  - Secure only with respect to a “mildly adaptive” adversary.

Algorand in Practice



Invest: NYC brings together global financial leaders to discuss macroeconomics and crypto. Regist

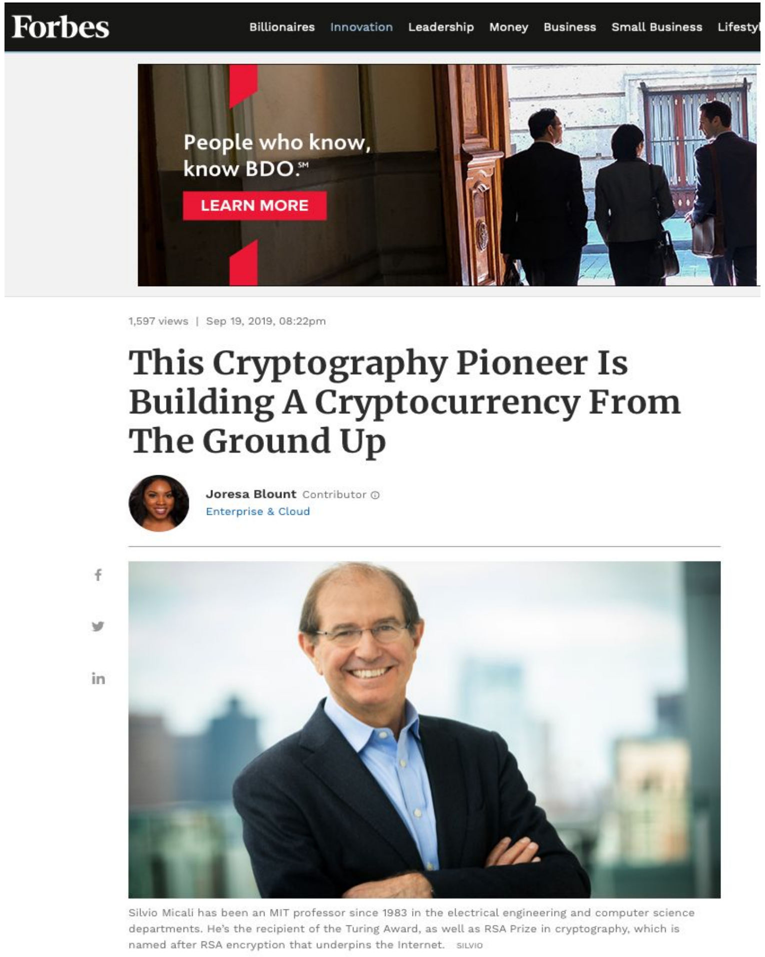
Algorand Raises $60 Million in Token Sale

Daniel Kuhn

Jun 19, 2019 at 23:00 UTC • Updated Jun 21, 2019 at 16:08 UTC

NEWS

Algorand raised over $60 million in a token sale of its native Algo token on Coinlist, using a Dutch Auction mechanism that ensures market participants set a uniform price per Algo. All 25 million tokens were sold at a market drive price of $2.40.



Forbes Billionaires Innovation Leadership Money Business Small Business Lifesty

People who know, know BDO.℠

LEARN MORE

1,597 views | Sep 19, 2019, 08:22pm

This Cryptography Pioneer Is Building A Cryptocurrency From The Ground Up

Joresa Blount Contributor

Enterprise & Cloud

Silvio Micali has been an MIT professor since 1983 in the electrical engineering and computer science departments. He's the recipient of the Turing Award, as well as RSA Prize in cryptography, which is named after RSA encryption that underpins the Internet. SILVIO

Critique

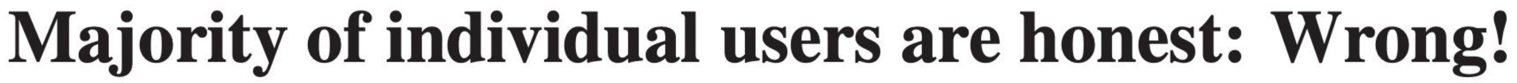
Trust Model

- Algorand assumes that majority of nodes remain honest.

Trust Model

- Algorand assumes that majority of nodes remain honest.

Not Rational !



Majority of individual users are honest: Wrong!

-- Another Look at ALGORAND

Trust Model

- Algorand assumes that majority of nodes remain honest
    - The adversary can provide sufficient incentives to ask selected leaders and verifiers to publish their roles.

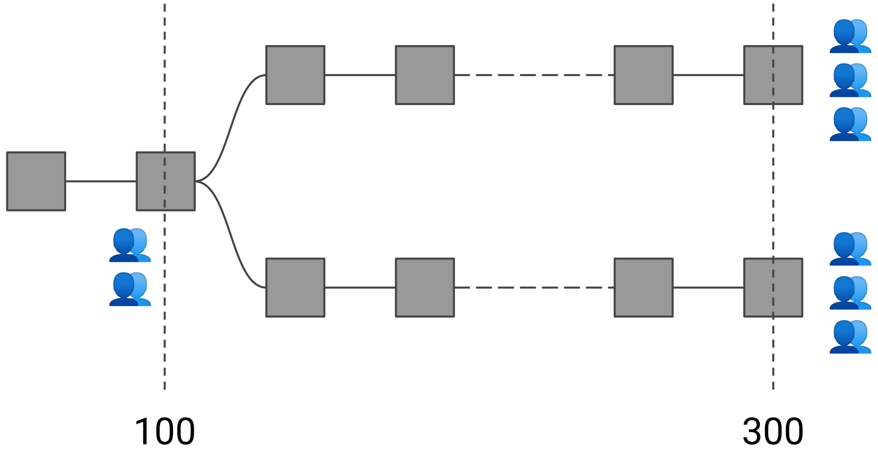
Trust Model

- Algorand assumes that majority of nodes remain honest
  - The adversary can provide sufficient incentives to ask selected leaders and verifiers to publish their roles.
  - Users need to forget their secrets, so they cannot be corrupted later.

    Obviously there is no incentive.

No Forks

costless simulation attack → possible forks



Writing

The technical details are explained very well but

- Organization is not great - takes some time to figure out how the pieces fit together, where the overview is, where the details are, etc.
- Male pronouns - the authors always use male pronouns for users (e.g., “...which the user can include in **his** messages…”).

Future Work and Conclusion

Future Work

- Incentives
  - Encourage users to be online when selected and pay network costs.
- Cost of joining
  - New users fetch all existing blocks and their certificates.
  - Lot of data + high throughput of Algorand => scalability challenge.
- Forward security
  - Identities of committee members are revealed after they send a message.
  - Attackers may attempt to corrupt users over time
    - Gather enough user keys, construct fake certificate, create fork

Conclusion

- Algorand is a cryptocurrency that
  - Provides low latency.
  - Provides high throughput.
  - Small probability of forking.
- Design based on
  - Cryptographic sortition
  - BA* byzantine agreement protocol.
- Need to solve
  - Creation of incentives.
  - High cost of joining.

END