Algorand: Scaling Byzantine Agreements for Cryptocurrencies

Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, Nickolai Zeldovich

Presented by:
Preet Patel and Umang Lathia
Outline

- Overview of Distributed Ledgers
- Algorand - Introduction and Motivation
- Cryptographic Sortition
- BA* - Byzantine Agreement
- Implementation details
- Evaluation
- Current status and open questions
A Distributed Ledger is simply a sequence of data organized in blocks.

Blockchains are a form of distributed ledgers.

Readable/Writable by all

Tamperproof
Distributed Ledger - Bitcoin

- The blockchain has seen spectacular success in Bitcoin
- Main Idea: Consensus via **Proof-of-Work**
- High level idea:
  - Transactions are signed and gossiped across the network
  - Miners collect transactions and solve a cryptographic riddle to successfully propagate their block
  - This proof of work is implemented such that there is one block generated every ~10 minutes.
Distributed Ledger - Bitcoin

- Assumption: Honest majority of computing power

- Weaknesses of Bitcoin:
  - Long latency for confirmation and block creation (~10 minutes)
  - Waste of computational resources
  - Forks!
Introducing ALGORAND
Why Algorand?

- Computational Complexity
- Weighted by money vs. number of users
- Cryptographic Sortition (avoids Sybil attacks)
- New Byzantine Agreement (BA*)
Algorand Assumptions

- Network
- Adversary
- Fraction of Honest Money ($\frac{2}{3}$)
- Synchronized Global Clock (e.g., NTP)
Cryptographic Sortition

- Goal: Select a subset of users

- Idea: Get the users to interact and talk amongst themselves to elect a committee. Simple!
Cryptographic Sortition

- Goal: Select a subset of users

- Idea: Get the users to interact and talk amongst themselves to elect a committee. Simple!

However, the adversary could be listening to the election process!

Choosing the users in an interactive way can lead to them being attacked by the adversary before they can participate in the protocol.
Cryptographic Sortition

Goal: **Local and non-interactive** way to select subset of users

Implemented using **Verifiable Random Functions (VRF):**

- **Secret Key**
- **Seed**
- **VRF**
- **Hash**
- **Proof**

Hash is **pseudo-random** and is **distributed uniformly between** \([0, 2^{\text{hashlen}} - 1]\)
Cryptographic Sortition

- Since each user is weighted by their money, we can let one unit of currency be one “sub-user”. A user with \(x\) units of currency has \(x\) “sub-users”.
- The probability of any unit of currency being chosen is \(p = \frac{\tau}{W}\)
  - \(\tau\) - Expected number of users that we want to be chosen
  - \(W\) - Total amount of currency units

For a user with \(x\) units of currency, we divide the interval \([0,1)\) into \(x+1\) sub-intervals:

Calculate \(c = \frac{\text{hash}}{2^\text{hashlen}}\) and choose the number of sub-users based on which sub-interval \(c\) falls in.
Cryptographic Sortition - Key ideas

Properties:

- Chooses a random subset of users by their weight
- Implemented using Verifiable Random Function, including proof of selection
- Each user knows whether they have been selected or not for that round

Security:

- The chosen user needs to communicate only once, and then they are irrelevant
- Adversary cannot know how many times (or if) a user is chosen
- Splitting currency across users does not increase chance of being selected!

Sortition will be used at each step of the Algorand Byzantine Agreement protocol
Gossip

Gossip -> sortition -> Block Proposal -> many -> Reduction

Final or Tentative <- ≥threshold <- CountVotes <- one <- Binary BA

Reduction <- two
Gossip

1. Gossip
2. Block Proposal
   - sortition
3. Reduction
   - many
4. Final or Tentative
   - ≥threshold
5. CountVotes
   - one
6. Binary BA
    - two

BA*
Gossip

- Select a small random pool of peers weighted by money to send messages to
- Signed by private key
- Check for forgery
- Avoid forwarding loops
- Each peer forwards messages that it hears to its random pool
Block Proposal

Gossip → sortition → Block Proposal → many → Reduction

Final or Tentative → ≥threshold → CountVotes → one → Binary BA
Block Proposal

- Select committee
- Send blocks, priority, proof to ALL users
- Select second committee
- Send *highest priority* block, proof to ALL users
Reduction

- **Gossip** → **sortition** → **Block Proposal** → **many** → **Reduction**
- **Final or Tentative** → **≥threshold** → **CountVotes** → **one** → **Binary BA** → **BA* [two]**
Reduction

- Select committee
- Receive **highest priority** block, proof
- **CountVotes()**
  - More than threshold, ensures that no two distinct values can be returned
- Reduce to one block to consider
- Send block, proof to ALL users
- 2 steps
Binary BA

Gossip → sortition → Block Proposal → many → Reduction

Final or Tentative → ≥threshold → CountVotes → one → Binary BA

BA*
procedure BinaryBA*(ctx, round, block_hash):
    step ← 1
    r ← block_hash
    empty_hash ← H(Empty(round, H(ctx.last_block)))
    while step < MaxSteps do
        CommitteeVote(ctx, round, step, τ_STEP, r)
        r ← CountVotes(ctx, round, step, τ_STEP, τ_STEP, λ_STEP)
        if r = TIMEOUT then
            r ← block_hash
        else if r ≠ empty_hash then
            for step < s' ≤ step + 3 do
                CommitteeVote(ctx, round, s', τ_STEP, r)
            if step = τ then
                CommitteeVote(ctx, round, final, τ_FINAL, r)
            return r
        step++
        CommitteeVote(ctx, round, step, τ_STEP, r)
        r ← CountVotes(ctx, round, step, τ_STEP, τ_STEP, λ_STEP)
        if r = TIMEOUT then
            r ← empty_hash
        else if r = empty_hash then
            for step < s' ≤ step + 3 do
                CommitteeVote(ctx, round, s', τ_STEP, r)
            return r
        step++
        CommitteeVote(ctx, round, step, τ_STEP, r)
        r ← CountVotes(ctx, round, step, τ_STEP, τ_STEP, λ_STEP)
        if r = TIMEOUT then
            if CommonCoin(ctx, round, step, τ_STEP) = 0 then
                r ← block_hash
            else
                r ← empty_hash
        step++

// No consensus after MaxSteps; assume network
// problem, and rely on §8.2 to recover liveness.
HangForever()
procedure BinaryBA*(ctx, round, block_hash):
    step ← 1
    r ← block_hash
    empty_hash ← H(Empty(round, H(\text{ctx.last block})))
    while step < MaxSteps do
        CommitteeVote(ctx, round, step, τSTEP, r)
        r ← CountVotes(ctx, round, step, τSTEP, rSTEP, λSTEP)
        if r = TIMEOUT then
            return 0
        else if r = empty_hash then
            for step < s' ≤ step + 3 do
                CommitteeVote(ctx, round, s', τSTEP)
            return 1
        CommitteeVote(ctx, round, step, τSTEP, r)
        r ← CountVotes(ctx, round, step, τSTEP, rSTEP, λSTEP)
        if r = TIMEOUT then
            return 0
        else if r = empty_hash then
            for step < s' ≤ step + 3 do
                CommitteeVote(ctx, round, s', τSTEP)
            return 1
        CommitteeVote(ctx, round, step, τSTEP, r)
        r ← CountVotes(ctx, round, step, τSTEP, rSTEP, λSTEP)
        if r = TIMEOUT then
            return 0
        else if CommitteeVote(ctx, round, step) = 0 then
            r ← block_hash
        else
            r ← empty_hash
        step++
    // No consensus after MaxSteps; assume network
    // problem, and rely on §8.2 to recover liveness.
    HangForever()
Binary BA

- Select committee
- Receive block, proof
- Goal: choose block or empty block
Binary BA

- Select committee
- Receive block, proof
- Goal: choose block or empty block
- Rounds of voting
  - Adversary could push threshold for some users, but not others
  - → Timeout sets the next step vote to that which could have been returned
- Min: 2 steps Max: 11 steps (to prevent incorrectness)
  - Common Coin (getting unstuck, lowest hash)
  - Recovery Protocol
CountVotes

Gossip \(\xrightarrow{\text{sortition}}\) Block Proposal \(\xrightarrow{\text{many}}\) Reduction

Final or Tentative \(\xleftarrow{\geq \text{threshold}}\) CountVotes \(\xrightarrow{\text{one}}\) Binary BA

Reduction \(\xrightarrow{\text{two}}\)
CountVotes

- Select committee
- CountVotes() - Final
  - Number of Users who decided in the first round
Final or Tentative

- Select committee
- CountVotes() - Final
  - Number of Users who decided in the first round
  - If proposer was honest, this will happen
- If greater than a threshold, it is final
- Otherwise tentative
  - Fixed during recovery process or final block after
  - No more consensus can be achieved if there is a fork

Final or Tentative \( \geq \text{threshold} \) CountVotes
How To Make This Feasible?
**Evaluation**

- Prototype deployed on EC2 using 1000 VMs, each with 8 cores
- Latency:

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

**Figure 6:** Latency for one round of Algorand in a configuration with 500 users per VM, using 100 to 1,000 VMs.
Evaluation

- Throughput:

\[ \sim 125 \times \text{Bitcoin's throughput (750 MB/hour vs. 6 MB/hour)} \]

*Figure 7*: Latency for one round of Algorand as a function of the block size.
Evaluation

- Malicious users:

**Figure 8**: Latency for one round of Algorand with a varying fraction of malicious users, out of a total of 50,000 users.
thank you!
Discussion Questions

- Recently raised $4 million in seed funding
- How should incentives be handled?
- How is currency generated (bootstrapping)?
- Your weight (money) is public
Backup Slides

- Seed for sortition - comes from the previous round - proposed while proposing blocks
- BA* steps are NOT synchronized across users - wait for some time to receive a block, else use empty block
how many sub-users are selected, as follows. The probability that exactly \( k \) out of the \( w \) (the user’s weight) sub-users are selected follows the binomial distribution, 
\[
B(k; w, p) = \binom{w}{k} p^k (1-p)^{w-k},
\]
where \( \sum_{k=0}^{w} B(k; w, p) = 1 \). Since

are selected, the sortition algorithm divides the interval \([0, 1]\) into consecutive intervals of the form
\[
I_j = \left[ \sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p) \right]
\]
for \( j \in \{0, 1, \ldots, w\} \). If

Visual: [https://homepage.divms.uiowa.edu/~mbognar/applets/bin.html](https://homepage.divms.uiowa.edu/~mbognar/applets/bin.html)
## Backup Slides

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

![Graph showing committee size vs. % of honest users with a line at 5 \times 10^{-9}.

Committee Size vs. % of Honest Users

- Committee Size on the y-axis ranging from 0 to 4500
- % of Honest Users on the x-axis ranging from 16 to 90
- A line at 5 \times 10^{-9} indicating a specific point on the graph.}