Algorand: Scaling Byzantine Agreements for Cryptocurrencies

Speaker: Yinlong Qian

Authors: Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios VlACHOS, Nickolai Zeldovich
Problem Statement - Motivation

• Algorand - cryptocurrency
  ✦ low latency
  ✦ high scalability
  ✦ no forks/branches

• Key components
  ✦ Verifiable random function/VRF
  ✦ cryptographic sortition
  ✦ byzantine agreement protocol/BA★
Challenge

• Sybil attacks
• Scale to millions of users
• Resilient to denial-of-service attacks

• Techniques used by Algorand
  ✷ Weighted users
  ✷ Consensus by committee
  ✷ Cryptographic sortition
  ✷ Participant replacement
Related Work - Comparison

• Proof of work: Bitcoin
• Byzantine fault tolerance: Honey Badger Et al.
• Byzantine consensus: Bitcoin-NG, Pass and Shi’s paper, Stellar
• Proof of stake: Ouroboros
• Trees and DAGs: GHOST, SPECTRE, Meshcash
Goals and Assumptions

• Safety
• Liveness

• Assumptions
  ✷ standard cryptographic algorithms
  ✷ liveness - “strong synchrony”
  ✷ safety - “weak synchrony”
  ✷ loosely synchronized clocks across users (e.g. NTP)
Cryptographic sortition

• Requirements
  ✷ Verifiable random functions/VRF: \(<hash, pi>\)
  ✷ random seed
  ✷ public/private key pair \((pk_i, sk_i)\)

• Selection procedure
  • Choosing the seed
  • Choosing \(sk_u\) well in advance of the seed
Cryptographic sortition - selection

- Binomial distribution
  - \( B(k; w, p) = \binom{w}{k} p^k (1-p)^{w-k} \)
- hash
  - uniformly distributed [0, 2^{\text{hashlen}-1}]
- axis:
  - \( j = 5 \)

**Algorithm 1:** The cryptographic sortition algorithm.

```plaintext
procedure Sortion(sk, seed, \tau, role, w, W):

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

  return \( \langle \text{hash}, \pi, j \rangle \)
```

0

B(0-5; 10, p)  B(0-6; 10, p)
Cryptographic sortition - seed

- Seed at round $r$
  - VRF with seed at round $r-1$
    - $<seed_r, pi> = VRF_{sku}(seed_{r-1} \mathbin\| r)$
  - Cryptographic hash function $H$
    - $seed = H(seed_{r-1} \mathbin\| r)$
  - refreshed every $R$ rounds
    - $seed_{r-1-(r \mod R)}$
Cryptographic sortition - private key

- Weak synchrony
  - drop block proposals
  - force empty blocks
  - compute future selection seeds

- Solution
  - every period of length $b$
  - a strongly synchronous period of length $s < b$
Block proposal

- Minimizing unnecessary block transmissions
  - priority of block proposals
    - hashing of hash output of VRF
- Waiting for block proposals
  - $\lambda_{\text{STEPVAR}}$
  - $\lambda_{\text{PRIORITY}}$
  - $\lambda_{\text{BLOCK}}$
- Malicious proposers
  - initialize different blocks
    - result in empty block
BA★ - Top-level procedure

• phase 1
  ✦ reduction
  ✦ agreement on 1 of 2 options

• phase 2
  ✦ binaryBA★
  ✦ reach agreement
    ▪ a proposed block
    ▪ an empty block

```
procedure BA★(ctx, round, block):
  hblock ← Reduction(ctx, round, H(block))
  hblock★ ← BinaryBA★(ctx, round, hblock)
  // Check if we reached “final” or “tentative” consensus
  r ← CountVotes(ctx, round, FINAL, TFINAL, τFINAL, λSTEP)
  if hblock★ = r then
    return ⟨FINAL, BlockOfHash(hblock★)⟩
  else
    return ⟨TENTATIVE, BlockOfHash(hblock★)⟩
```

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

- Consensus on 1 of 2 values
  - Step 1
    - committee member
      - vote for hash of block
  - 1 block hash gets supermajority votes
  - Step 2
    - committee member
      - the specific proposed block hash
      - the default empty block hash

```plaintext
procedure Reduction(ctx, round, hblock):
  // step 1: gossip the block hash
  CommitteeVote(ctx, round, REDUCTION_ONE, 
               τ_STEP, hblock)
  // other users might still be waiting for block proposals,
  // so set timeout for λ_BLOCK + λ_STEP
  hblock₁ ← CountVotes(ctx, round, REDUCTION_ONE, 
                     T_STEP, τ_STEP, λ_BLOCK + λ_STEP)
  // step 2: re-gossip the popular block hash
  empty_hash ← H(Empty(round, H(ctx.last_block)))
  if hblock₁ = TIMEOUT then
    CommitteeVote(ctx, round, REDUCTION_TWO, 
                  τ_STEP, empty_hash)
  else
    hblock₂ ← CountVotes(ctx, round, REDUCTION_TWO, 
                         T_STEP, τ_STEP, λ_STEP)
    if hblock₂ = TIMEOUT then return empty_hash ;
    else return hblock₂ ;

Algorithm 7: The two-step reduction.
```
BA★ - Binary agreement

• All users reach consensus on a block (hash)
• Case 1: strongly synchronous (common)
  ✡ Reach consensus in the FIRST step
  ✡ Go to FINAL step and terminate
• Case 2: not strongly synchronous (partition)
  ✡ Can NOT Reach consensus in the FIRST step
    ▪ final consensus
    ▪ tentative consensus

MAXSTEPS
Case 3: two groups of honest users (getting stuck)

block_hash vs empty_hash

Common Coin:
- binary value predominantly same for all users
- least significant bit of the lowest hash of VRF-based committee member hashes attached to messages received

```
procedure CommonCoin(ctx, round, step, τ):
    minhash ← 2^{hashlen}
    for m ∈ incomingMsgs[round, step] do
        ⟨votes, value, sorthash⟩ ← ProcessMsg(ctx, τ, m)
        for 1 ≤ j < votes do
            h ← H(sorthash||j)
            if h < minhash then minhash ← h;
    return minhash mod 2

Algorithm 9: Computing a coin common to all users.
```


\textbf{Algorithm 8}: BinaryBA★ executes until consensus is reached on either \texttt{block\_hash} or \texttt{empty\_hash}. 

// No consensus after MAXSTEPS; assume network // problem, and rely on §8.2 to recover liveness.

\texttt{HangForever()}

---

\texttt{procedure BinaryBA★(ctx, round, block\_hash):}
\begin{verbatim}
step ← 1
r ← block\_hash
empty\_hash ← H(Empty(round, H(ctx.last_block)))
while step < MAXSTEPS do
    CommitteeVote(ctx, round, step, \tau_{STEP}, r)
    r ← CountVotes(ctx, round, step, T_{STEP}, \tau_{STEP}, \lambda_{STEP})
    if r = TIMEOUT then
        r ← block\_hash
    else if r ≠ empty\_hash then
        for step' ≤ step + 3 do
            CommitteeVote(ctx, round, step', \tau_{STEP}, r)
        if step = 1 then
            return r
        step++
    CommitteeVote(ctx, round, step, \tau_{STEP}, r)
    r ← CountVotes(ctx, round, step, T_{STEP}, \tau_{STEP}, \lambda_{STEP})
    if r = TIMEOUT then
        r ← empty\_hash
    step++
\end{verbatim}

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

Algorithm 4: Voting for value by committee members.
user.sk and user.pk are the user’s private and public keys.
procedure CountVotes($ctx, round, step, T, \tau, \lambda)$:

$\text{start} \leftarrow \text{Time()}$

$\text{counts} \leftarrow \emptyset$ // hash table, new keys mapped to 0

$\text{voters} \leftarrow \emptyset$

$\text{msgs} \leftarrow \text{incomingMsgs[round, step].iterator()}$

\textbf{while} TRUE \textbf{do}

$\ m \leftarrow \text{msgs.next()}$

\textbf{if} $m = \bot$ \textbf{then}

\hspace{1em} \textbf{if} $\text{Time()} > \text{start} + \lambda$ \textbf{then return} \text{TIMEOUT};

\textbf{else}

\hspace{1em} $\langle \text{votes, value, sorthash} \rangle \leftarrow \text{ProcessMsg}(ctx, \tau, m)$

\hspace{1em} \textbf{if} $pk \in \text{voters or votes} = 0$ \textbf{then continue;}

\hspace{1em} $\text{voters} \cup = \{pk\}$

\hspace{1em} $\text{counts[value]} += \text{votes}$

\hspace{1em} // if we got enough votes, then output this value

\hspace{1em} \textbf{if} $\text{counts[value]} > T \cdot \tau$ \textbf{then}

\hspace{2em} \textbf{return} \text{value}

\textbf{Algorithm 5:} Counting votes for round and step.
procedure ProcessMsg(ctx, τ, m):

⟨pk, signed_m⟩ ← m
if VerifySignature(pk, signed_m) ≠ OK then
    return ⟨0, ⊥, ⊥⟩
⟨round, step, sorthash, π, hprev, value⟩ ← signed_m
// discard messages that do not extend this chain
if hprev ≠ H(ctx.last_block) then return ⟨0, ⊥, ⊥⟩;
votes ← VerifySort(pk, sorthash, π, ctx.seed, τ,
    ⟨“committee”, round, step⟩, ctx.weight[pk], ctx.W)
return ⟨votes, value, sorthash⟩

Algorithm 6: Validating incoming vote message m.
Higher-level issues

• Block format
• Safety and liveness
• Bootstrapping
  ✷ Bootstrapping the system
  ✷ Bootstrapping new users
  ✷ Storage
• Communication
  ✷ Gossiping blocks and relaying messages
  ✷ scalability
Implementation

- Create fork ✧ 1-h controlled
- $\lambda_{\text{PRIORITY}}$ ✧ 5 seconds
- $\lambda_{\text{BLOCK}}$ ✧ 1 minute
- $\lambda_{\text{STEP}}$ ✧ 20 seconds
- $\lambda_{\text{STEPVAR}}$ ✧ 10 seconds

<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 ($$B.2$)</td>
</tr>
<tr>
<td>$T_{\text{STEP}}$</td>
<td>threshold of $\tau_{\text{STEP}}$ for $BA\star$</td>
<td>68.5% ($$B.2$)</td>
</tr>
<tr>
<td>$\tau_{\text{FINAL}}$</td>
<td>expected # of final committee members</td>
<td>10,000 ($$C.1$)</td>
</tr>
<tr>
<td>$T_{\text{FINAL}}$</td>
<td>threshold of $\tau_{\text{FINAL}}$ for $BA\star$</td>
<td>74% ($$C.1$)</td>
</tr>
<tr>
<td>$\text{MAXSTEPS}$</td>
<td>maximum number of steps in Binary$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 $BA\star$ step</td>
<td>20 seconds</td>
</tr>
<tr>
<td>$\lambda_{\text{STEPVAR}}$</td>
<td>estimate of $BA\star$ completion time variance</td>
<td>5 seconds</td>
</tr>
</tbody>
</table>

Figure 4: Implementation parameters.
Evaluation - questions

- Latency vs scale (number of users)
- Throughput: latency vs block size
- CPU, bandwidth, and storage costs
- Latency vs percentage of malicious users
- Reasonable timeout parameters
Evaluation - 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

Figure 7: Latency for one round of Algorand as a function of the block size.
Figure 8: Latency for one round of Algorand with a varying fraction of malicious users, out of a total of 50,000 users.
Extensions / Improvements

• Incentives
  ✷ encourage users to participate

• Cost of joining
  ✷ fetching all existing blocks

• Forward security
  ✷ fake certificate -> fork
Conclusions. Thanks!

- Algorand: a new cryptocurrency
  - features
    - transactions confirmed in a minute
    - negligible probability of forking
  - design
    - cryptographic sortition
    - BA★ - Byzantine agreement protocol
Background - POW and Double spend attack

• Possibility of forks

Block 1

Block 2

Block 3?

Block 3?
Background - POW and Double spend attack

• Possibility of forks

More POW fork gets accepted.

Block 1

Block 2

Block 3?

Rejected

Block 1

Block 2

Block 3

Block 4

Block 5
Background - POW and Double spend attack

• Alice and Bob’s transaction
  ✷ Alice pays Bob 25 Bitcoins for an iPad
  ✷ Transaction recorded in Block 2
Background - POW and Double spend attack

- Alice and Bob’s transaction
  - Bob waits for 6 blocks/confirmations
  - Bob ships the iPad to Alice
Background - POW and Double spend attack

- Alice and Bob’s transaction
  - Eve, with 51% hash power, mines another fork with more POW
  - In ‘new’ Block 2, Alice pays herself 25 Bitcoins