Algorand

Algorand School

2022

Open Source: github.com/cusma/algorand-school

The architecture of consensus

1. How to choose the **proposer for the next block** for a **public and permissionless** blockchain?

2. How to ensure that **there is no ambiguity** in the choice of the next block?

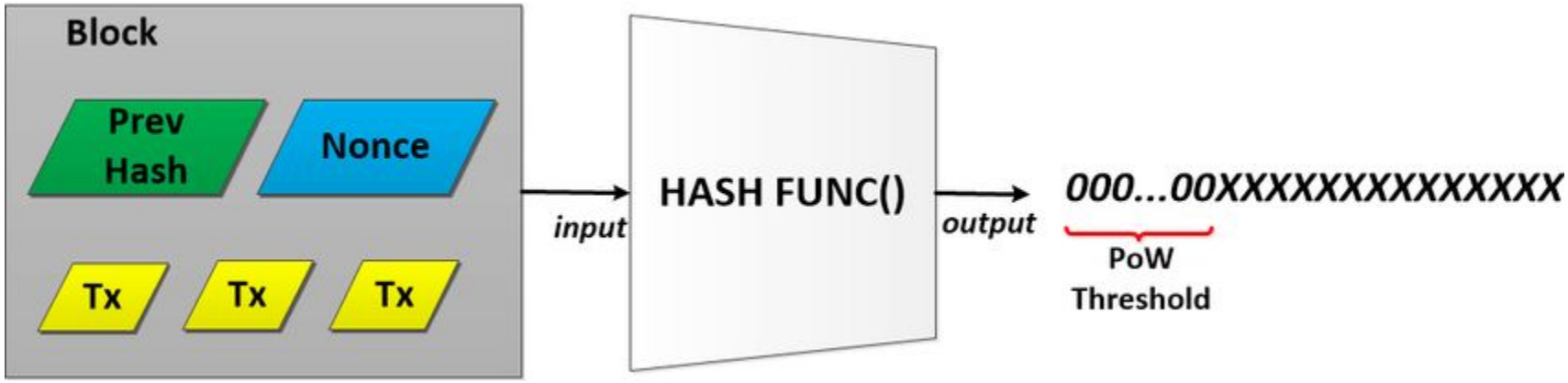
3. How to ensure that the **blockchain stays unique** and has **no forks**?

4. How to ensure that **consensus mechanism itself can evolve** over time while the blockchain is an immutable ledger?

Temple of Concordia

Valley of Temples (Agrigento), 440-430 B.C.

Proof of Work consensus mechanism



Miners **compete with each other** to append the next block and **earn a reward for the effort**, fighting to win an **expensive computational battle**.

The **more computational power**, the **higher the probability** of being elected as block proposer.

Proof of Stake consensus mechanism

Network participants **show their commitment and interest** in keeping the ledger safe and secure **proving the ownership of value stored on the ledger itself**.

The **higher the skin in the game** the **higher the probability** of being elected as block proposer.

Algorand PPoS Consensus

- **Scalable** 6000 TPS, billions of users
- **Fast** < 3.9 s per block
- **Secure** 0 downtime for over 23M blocks
- **Low fees** 0.001 ALGO per txn
- **No Soft Forks** prob. < 10^{-18}
- **Instant Transaction Finality**
- **Carbon neutral**
- **Minimal hardware node requirements**
- **No delegation or binding of the stake**
- **No minimum stake**
- **Secure with respect DDoS**
- **Network Partitioning resilience**

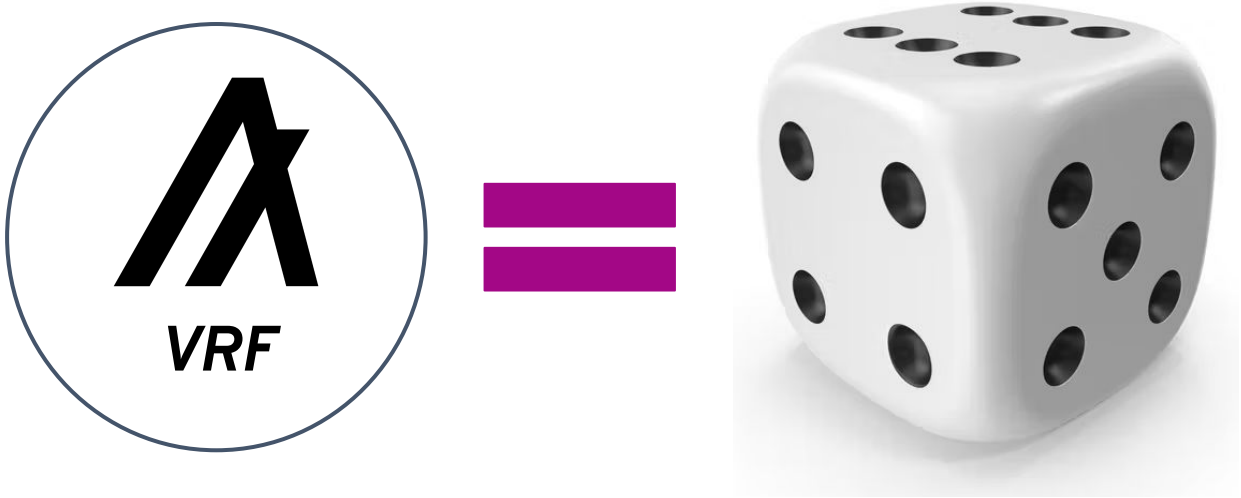
Silvio Micali

Algorand Founder

Professor MIT, Turing Award, Gödel Prize

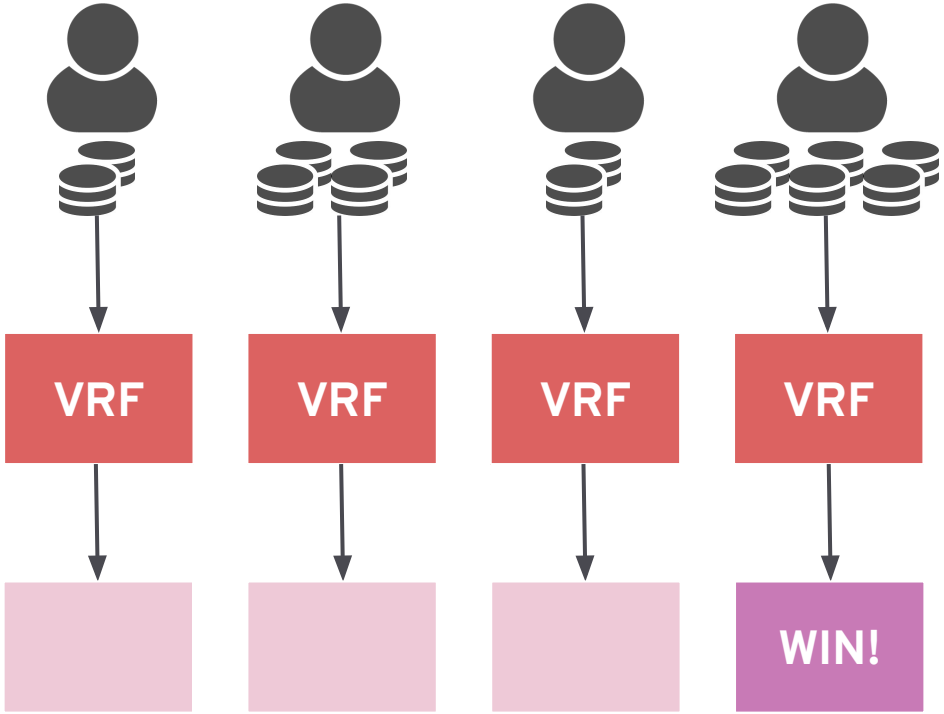
Digital Signatures, Probabilistic Encryption, Zero-Knowledge Proofs, Verifiable Random Functions and other primitives of modern cryptography.

Tamper-proof, unique and verifiable dices



1. Dices are **perfectly balanced** and **equiprobable**, nobody could tamper their result!
2. Keep **observing dice rolls** by no means increases the chance of guessing the next result!
3. Each dice is **uniquely signed** by its owner, nobody can roll someone else dices!
4. Dices are **publicly verifiable**, everybody can read the results of a roll!

Who chose the *next block*?



1. Each **ALGO** could be assimilated to a **tamper-proof dice** participating in a safe and secret **cryptographic dice roll**. More ALGOs more dices to roll.

2. For **each new block**, dice rolls are performed in a **distributed, parallel and secret** and manner, directly on online accounts' hardware (in microseconds).

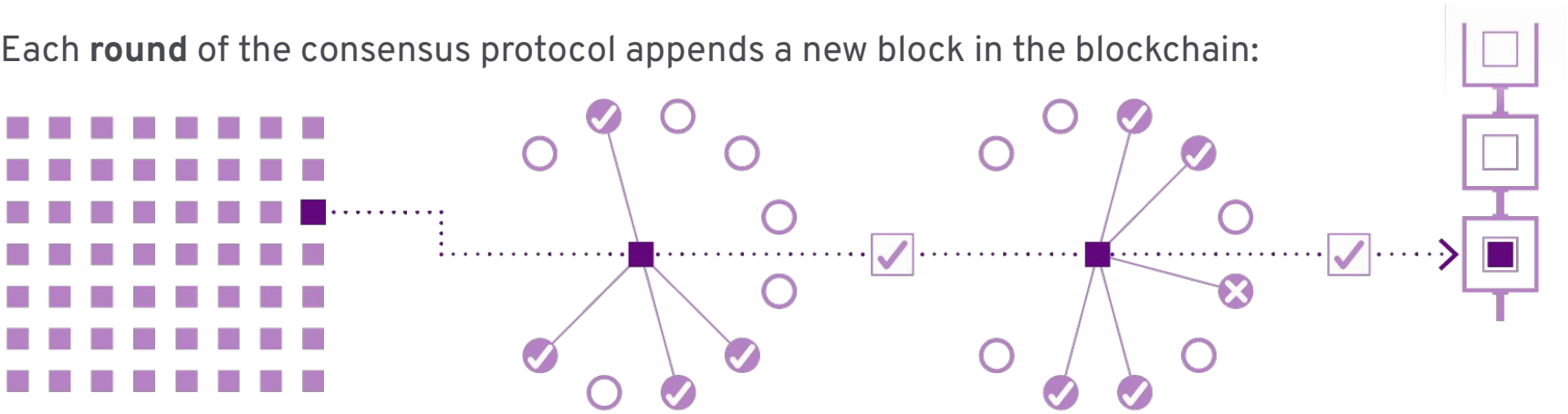
3. The **winner is revealed** in a safe and **verifiable way** only after winning the dice roll, proposing the next block.

A glimpse on “simplified” VRF sortition

1. A **secret key (SK) / public verification key (VK)** pair is associated **with each ALGO in the account**
2. For each new round ***r*** of the **consensus protocol** a **threshold *L(r)*** is defined
3. **Each ALGO** in the account **performs a VRF**, using its own **secret key (SK)**, to generate:
   a. a pseudo-random number: $Y = VRF_{SK}(seed)$
   b. the verifiable associated proof: $\rho_{SK}(seed)$
4. If $Y = VRF_{SK}(seed) < L(r)$, that specific ALGO **“wins the lottery”** and **viraly propagates the proof** of its victory $\rho_{SK}(seed)$ to other **network’s nodes**, through “gossiping” mechanism
5. Others node can use the **public verification key (VK)** to verify, through $\rho_{SK}(seed)$, that the number **Y** was generated by **that specific ALGO, owned by the winner of the lottery**

Pure Proof of Stake, in short

Each **round** of the consensus protocol appends a new block in the blockchain:



An account is elected to **propose** the **next block**	A **committee** is elected to **filter** and **vote** on the **block proposals**	A **new committee** is elected to reach a quorum and **certify** the **block**	The **new block** is **appended** to the **blockchain**

Through the **cryptographic lottery**, an **online account** is elected with probability directly proportional to its stake: **each ALGO corresponds to an attempt** to win the lottery!

Pure Proof of Stake security

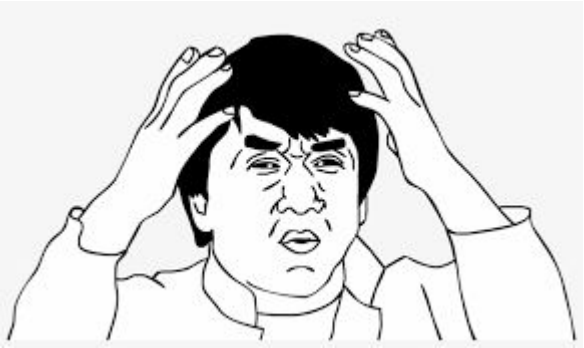
Algorand's decentralized Byzantine consensus protocol can tolerate an **arbitrary number of malicious users** as long as honest users hold a **super majority of the total stake** in the system.

1. The adversary **does not know which users he should corrupt**.
2. The adversary **realizes which users are selected too late** to benefit from attacking them.
3. Each **new set of users will be privately and individually elected**.
4. During a network partition in Algorand, the adversary is **never able to convince two honest users to accept two different blocks** for the same round.
5. Algorand is able to **recover shortly after network partition** is resolved and guarantees that new blocks will be generated at the same speed as before the partition.

Cryptographic Sortition

Sortition???

- What does sortition even mean?

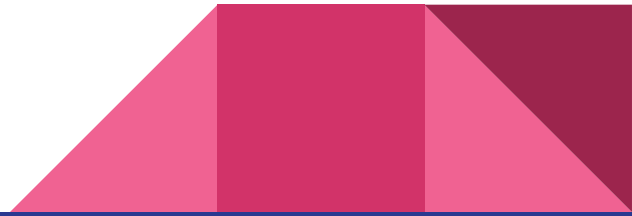


Sortition???

- 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.

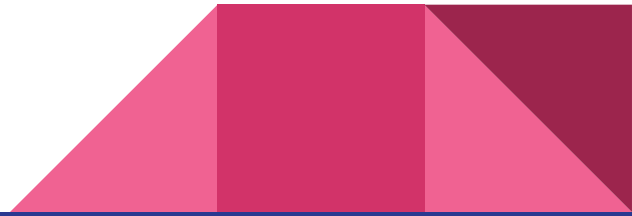


Sortition for Proposal and Committee

- 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.

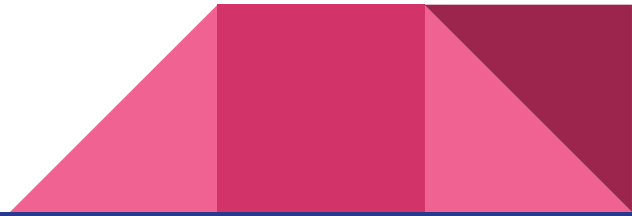
Sortition for Proposal and Committee

- 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$(sk, seed, \tau, role, w, W)$:

$\langle hash, \pi \rangle \leftarrow \text{VRF}_{sk}(seed||role)$

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

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, committte, 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**

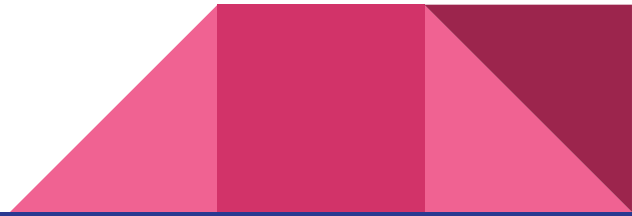
  j++

return j

Pseudocode for verifying sortition of a user with public key pk

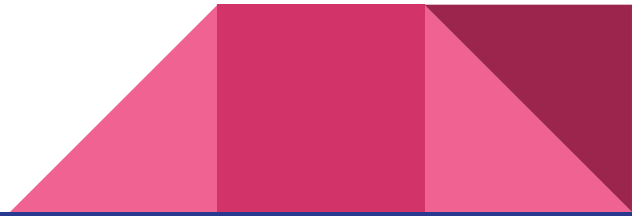
How to choose the seed?

- 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.



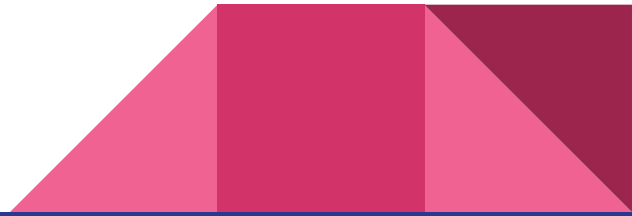
How to use the seed?

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)).

Proof-of-Stake
“Virtual Mining”

Algorand: Main Highlights

- Proof of Stake based Cryptocurrency
- High throughput: ~1 min to confirm transactions vs an hour in Bitcoin
- Public ledger with low probability of forks
- Assumes 2/3-honest stake majority
- Uses a gossip communication protocol

Algorand: Main Highlights

- Adaptive adversary: May corrupt dynamically, as long as 2/3 majority assumption holds
- Achieves **Consistency** assuming “weak synchrony”
  - Network can be asynchronous for long bounded time period b, but then must have strong synchrony for short period s < b
- Achieves **Liveness** assuming “strong synchrony”
  - Most honest users (e.g., 95%) can send messages that will reach within a known time bound

Main Design Ingredients

- Users weighted by stake (to prevent sybil attacks)
- Builds on byzantine agreement (BA) protocol of Micali [ITCS'17] for consensus
- BA protocol executed between a small committee of users for scalability
- Committee chosen at random, using cryptographic techniques

Algorand Consensus: Main Highlights

- BA protocol in expectation terminates in only 4 steps (in “honest” case) or 13 steps (in “dishonest” case)
- Player replaceability: Players across different steps of BA protocol may not be the same
  - Possible because protocol does not require “private state”
- For each step, players chosen at random, non-interactively, in a “publicly verifiable” manner