# Algorand

# Team Members

IIT2015064 - Jayesh Patil

IIT2015074 - Shubham Padia

IIT2015075 - Tushar Jandial

IIT2015085 - Swapnil Sharma

# Bitcoin’s Problems.
# What Algorand tries to solve?

# Honest Majority of Computational Power

- Assumes that no malicious entity controls the majority of the computational power devoted to block generation.
- Such an entity, in fact, would be able to modify the blockchain
- Due to the ability of this entity to modify the blockchain, it can rewrite the payment history, as it pleases.

# Computational Waste

- Bitcoin’s proof-of-work approach to block generation requires an extraordinary amount of computation.
- Bitcoin mining network uses more electricity in a year than the whole of Ireland.
- The estimated power use of the bitcoin network, which is responsible for verifying transactions made with the cryptocurrency, is 30.14TWh a year, which exceeds that of 19 other European countries.

# Concentration of Power

- Today, due to the exorbitant amount of computation required, a user, trying to generate a new block using an ordinary desktop expects to lose money.
- Indeed, for computing a new block with an ordinary computer, the expected cost of the necessary electricity to power the computation exceeds the expected reward.
- Only using pools of specially built computers (that do nothing other than “mine new blocks”), one might expect to make a profit by generating new blocks.

# Concentration of Power

- Accordingly, today there are, de facto, two disjoint classes of users: ordinary users, who only make payments, and specialized mining pools, that only search for new blocks.

- It should therefore not be a surprise that, as of recently, the total computing power for block generation lies within just five pools. In such conditions, the assumption that a majority of the computational power is honest becomes less credible.

# Ambiguity

○ In Bitcoin, the blockchain is not necessarily unique. Indeed its latest portion often forks: the blockchain may be —say— B1, . . . , Bk, B0 k+1, B0 k+2, according to one user, and B1, . . . , Bk, B00 k+1, B00 k+2, B00 k+3 according another user. Only after several blocks have been added to the chain, can one be reasonably sure that the first k + 3 blocks will be the same for all users.

○ Thus, one cannot rely right away on the payments contained in the last block of the chain. It is more prudent to wait and see whether the block becomes sufficiently deep in the blockchain and thus sufficiently stable.

Algorand is a new cryptocurrency that has properties:

○ No Latency : confirms transactions with latency on the order of a minute while scaling to many users.

○ No Forks : ensures that users never have divergent views of confirmed transactions, even if some of the users are malicious and the network is temporarily partitioned.

○ Uses Byzantine Agreement protocol (called BA*) to reach consensus among users on the next set of transactions

○ Security: Algorand uses a novel mechanism based on Verifiable Random Functions (VRFs) that allow users to privately check whether they are selected to participate in the BA.

# What does the Block contain?

# Block Format

- Algorand's blocks consist of a list of transactions, along with metadata needed by BA⋆.
- The metadata consists of the round number, the proposer's  seed, a hash of the previous block in the ledger, and a timestamp.
- The list of transactions in a block logically translates to a set of weights for each user's public key (based on the balance of currency for that key), along with the total weight of all outstanding currency.

# How it works?

# Phase : 1

- Users/Leaders (i.e., public keys)  are  randomly selected among all current users using **Cryptographic Sortition**.
- Selected users propose their block together with their priority and proof that they are selected for block generation.
- Phase 1 scales because this internal cryptographic lottery is super-fast and independent of how many users there are in the network.

# Phase : 2

- A set of  verifiers is randomly selected among all users using **Cryptographic Sortition**, proportionally to the amount of money that each of them owns in the system.
- The role of these verifiers is to agree on the block proposed by leaders.
- The agreement protocol (BA*) that the verifiers execute is super fast and independent of the total number of users in the network.

# How blocks are validated?

- The user checks if all transactions are valid.
- The user checks if the seed is valid.
- The user checks if the previous block hash is correct.
- The user checks if the block round number is correct.
- The user checks if the timestamp is greater than that of the previous block.

# Sortition and Proof of Sortition

$$.Hash(DigiSign(Q_r)) <= p$$

# Cryptographic Sortition

**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** $\langle hash, \pi, j \rangle$

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

# Cryptographic Sortition

**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

**Algorithm 2:** Pseudocode for verifying sortition of a user with public key *pk*.

# Block Proposal

- Minimizes unnecessary block transmissions.
  - Users discard messages about blocks that do not have the highest priority seen by that user so far.
- If some block proposers are malicious then they can trick different Algorand users into verifying different blocks. (but this scenario is relatively unlikely)
  - If the adversary is not the highest priority proposer in a round, then the highest priority proposer will gossip a consistent version of their block to all users.
  - If the adversary is the highest priority proposer in a round, they can propose the empty block, and thus prevent any real transactions from being confirmed.
  - Probability of this is at most $1-h$, by Algorand's assumption where $h > 2/3$ of the weighted user are honest.

# Byzantine Agreement *

**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 $BA\star$ for the next *round*, with a proposed *block*. H is a cryptographic hash function.

# Sybil Attack

1. Every user in algorand has weights associated with it.  These weights are proportional to the amount of money a user holds in the network.

2. The probability that a user is selected as a leader or verifier depend on its weights

3. If a malicious user create million of accounts , the probabilty of selection of that user
will still depend on the total money.

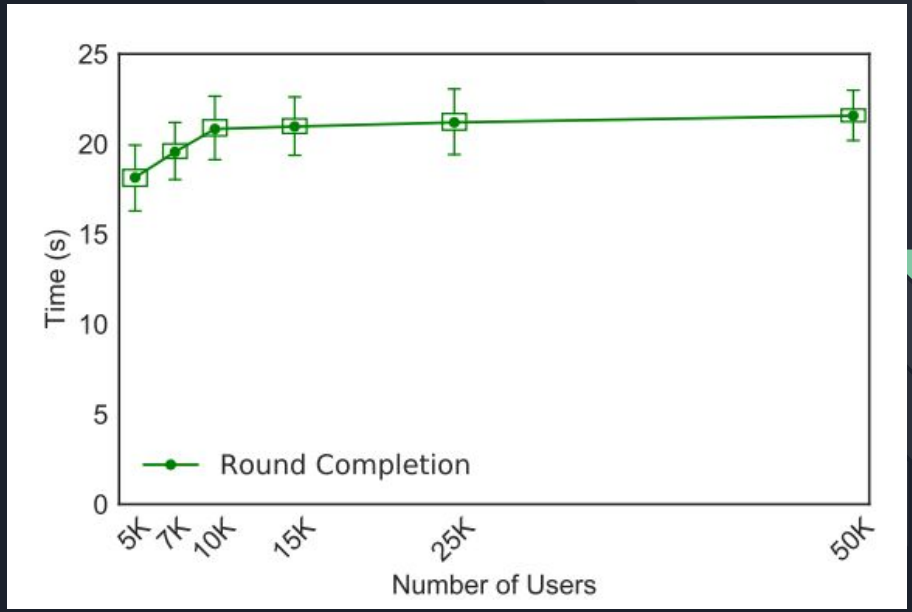
# Nothing at stake

1. This problem occurs in proof of stake where blocks can be build on each fork and can cause both forks to grow. And consensus is never reached.

2. In algorand, consensus takes place in many steps and new verifiers selected after each step vote previous block with highest votes, and other blocks are rejected.. Thus reaching consensus for every block without any forks.

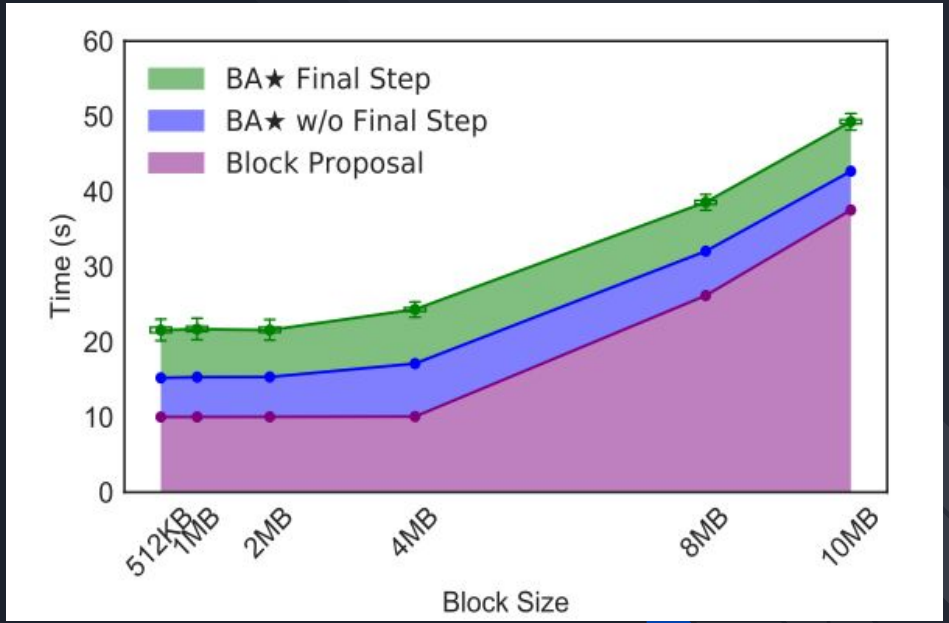
3. Double spending is also prevented if there are no forks.

# Latency:


25
20
15
10
5
0
Time (s)
Round Completion
5K
7K
10K
15K
25K
50K
Number of Users

# Throughput

1. Constant agreement time.
2. Linear growth wrt block size.
3. If we use 10MB blocks, we have 750 MB of transactions per hour which is 125 times of Bitcoin.



# Costs of running Algorand

1. Significant use of CPU is only for calculating hashes and verifying signatures which does not increase as network grows.

2. Communication cost is also independent of the number of users in network.

3. More storage space is required in algorand as it uses certificates in addition to block data for verification of block.

# Thanks

**procedure** CommitteeVote($ctx, round, step, \tau, value$):

// check if user is in committee using Sortition (Alg. 1)

$role \leftarrow \langle \text{“committee”}, round, step \rangle$

$\langle sorthash, \pi, j \rangle \leftarrow \text{Sortition}(user.sk, ctx.seed, \tau, role,$
$ctx.weight[user.pk], ctx.W)$

// only committee members originate a message

**if** $j > 0$ **then**

  Gossip($\langle user.pk, \text{Signed}_{user.sk}(round, step,$
  $sorthash, \pi, H(ctx.last\_block), value) \rangle$)

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

```
procedure CountVotes(ctx, round, step, T, τ, λ):
start ← Time()
counts ← {}     // hash table, new keys mapped to 0
voters ← {}
msgs ← incomingMsgs[round, step].iterator()
while TRUE do
    m ← msgs.next()
    if m = ⊥ then
        if Time() > start + λ then return TIMEOUT;
    else
        ⟨votes, value, sorthash⟩ ← ProcessMsg(ctx, τ, m)
        if pk ∈ voters or votes = 0 then continue;
        voters ∪ = {pk}
        counts[value] + = votes
        // if we got enough votes, then output this value
        if counts[value] > T · τ then
            return value
```

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

**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, T_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 = 1 then
            CommitteeVote(ctx, round, FINAL, τ_FINAL, r)
        return r
    step++

    CommitteeVote(ctx, round, step, τ_STEP, r)
    r ← CountVotes(ctx, round, step, T_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, T_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.
```

**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
hblock_1 ← 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 hblock_1 = TIMEOUT then
    CommitteeVote(ctx, round, REDUCTION_TWO,
            τ_STEP, empty_hash)
else
    CommitteeVote(ctx, round, REDUCTION_TWO,
            τ_STEP, hblock_1)
hblock_2 ← CountVotes(ctx, round, REDUCTION_TWO,
        T_STEP, τ_STEP, λ_STEP)
if hblock_2 = TIMEOUT then return empty_hash ;
else return hblock_2 ;
```

**Algorithm 7:** The two-step reduction.