# Anonymous voting using zero-knowledge proofs

CS 291D: Blockchain and Cryptocurrencies

# Motivation

- Traditional voting requires trust
  - Correctness of votes cast, counting and result declaration
- Electronic voting
  - Identities of voters revealed
  - Correctness of vote aggregation
- Voters maybe influenced
  - Knowledge of distribution of cast votes
  - Contestants know identities of voters
- Verifiable and anonymous voting
  - Can be used in e-voting scenarios or in consensus protocols for leader election
  - Fair and unbiased
  - Existing approaches either assume a trusted broadcast channel do not provide verifiability

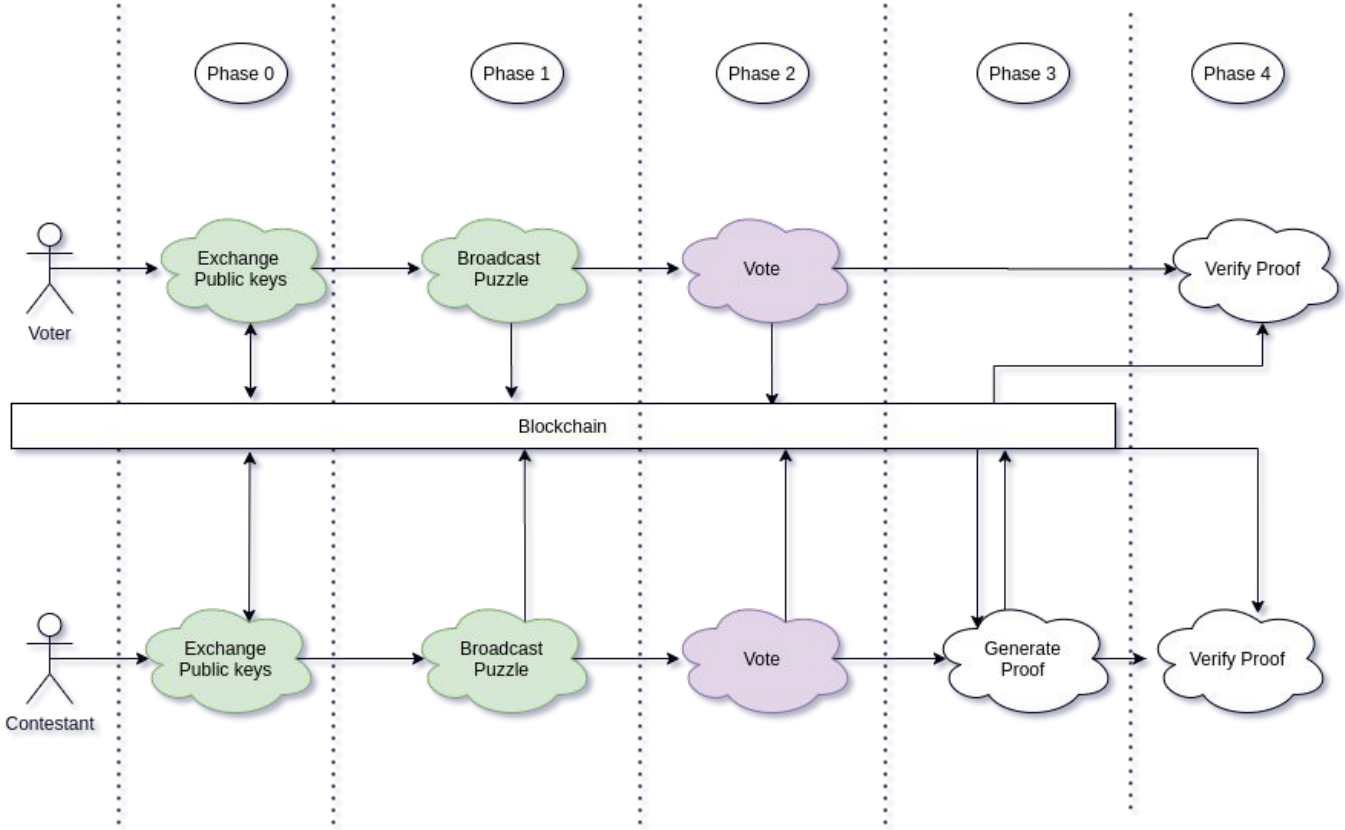
# Problem Definition

- Anonymous leader election protocol in distributed consensus
  - Conceal the vote of each participant from all other participants except the contestant of choice
- Setup:
  - Set of participants $V = \{V_1, V_2, \ldots, V_n\}$
  - Set of contestants $P \subseteq V$
- Honest majority
  - For t malicious participants, system has at least 2t+1 participants
- Constraints: Anonymity, Verifiability

# Threat Model

- Active setting
  - Malicious adversary can arbitrarily deviate from the protocol
  - Eg: vote multiple times, impersonate other participants
- Public broadcast channel is tamper free
- No collusion between any parties
- Each participant can assume one (or both) of the following roles
  - Voter - vote, verify proofs
  - Contestant - voter actions + count votes, generate proofs

# Our Protocol - Overview



# Our Protocol - Details

- Each phase runs for a predefined time window
- **Phase 0:**
  - Public key cryptosystem to send private messages on the blockchain
  - Participant is identified by its pk
  - Broadcast without signatures at the start
- **Phase 1:**
  - We use factorization problem (RSA) as a puzzle
  - Participants publish a product from the sampled prime pair with a signed message*
  - Contestants broadcast candidacy

# Our Protocol - Details

- **Phase 2:**
  - Voter encrypts their factors with the public key of contestant they vote for
  - Votes are broadcast on blockchain
- **Phase 3:**
  - Each contestant retrieves all of the vote messages from the blockchain and tries to decrypt them to check if the vote was for them, if so, retrieve the factors (x)
  - The contestant with enough number of votes (determined by a threshold) claims victory
  - Contestants compute public input (product of all puzzles) and produce a proof showing enough number of distinct factors

# Our Protocol - Details

- **Phase 4:**
  - Voter computes the public inputs namely the number of voters and the product of all the factors published by the voters on the blockchain
  - Verify the zkSNARK using these two pieces as public inputs
  - Additionally the following are checked:
    - Number of factor pairs is more than half the number of voters
    - No factor is repeated
    - No factor is trivial

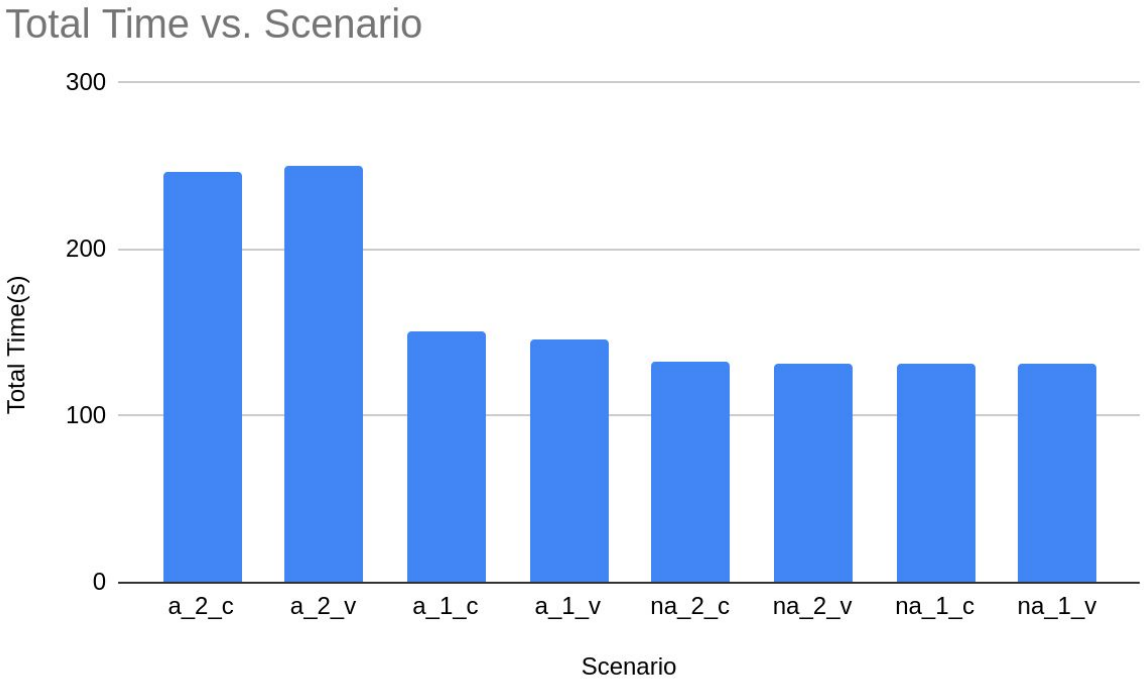
# Malicious Participants

- Voter
  - Vote/broadcast product multiple times
  - Encrypt incorrect factors
  - Broadcast unencrypted factors
  - Verify false proof
    - Compute N incorrectly
  - Do not vote (Node failure)
- Contestant
  - Broadcast false proof
    - Compute N incorrectly

# Implementation Details

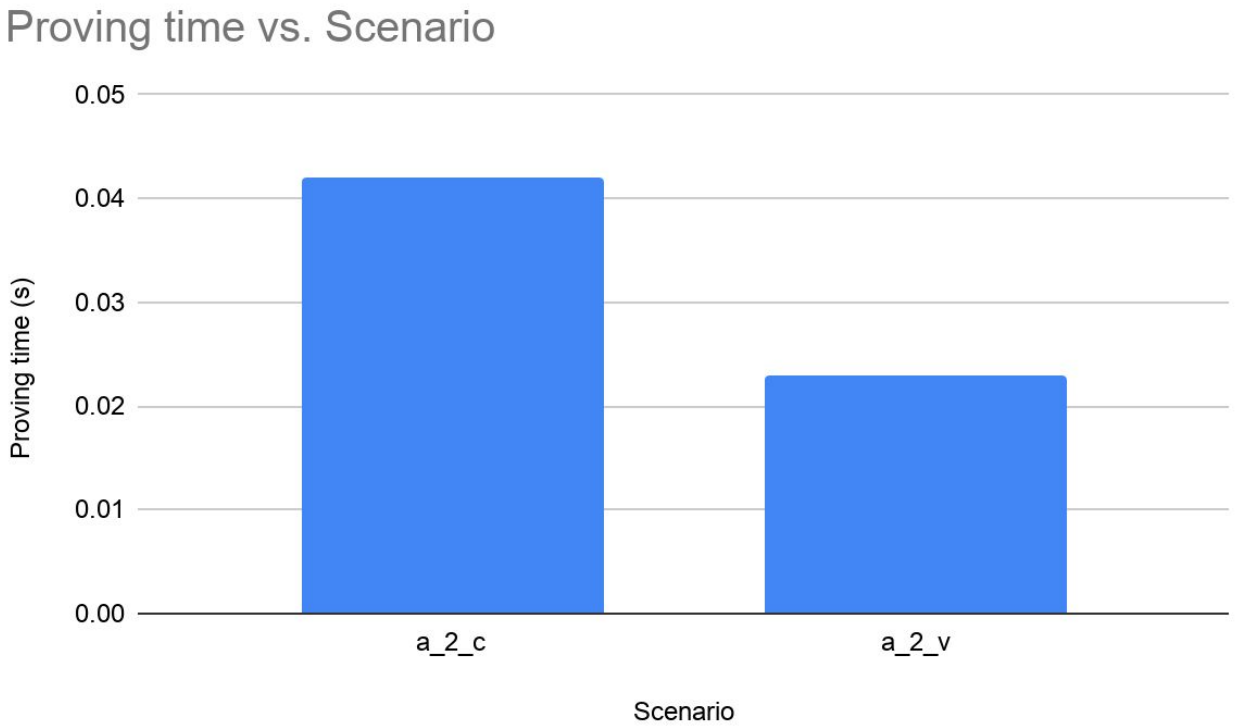
- cryptography for encryption, decryption, signature and verification
  - RSA 2048 bit keys, RSA-OAEP padding for encryption, RSA-PSS padding for signatures
- pysnark for zero knowledge proofs
  - Converts high level python into R1CS and uses libsnark in the backend for zero knowledge
- Ethereum (geth) for tamper-free public broadcast channel

# Evaluation (time)



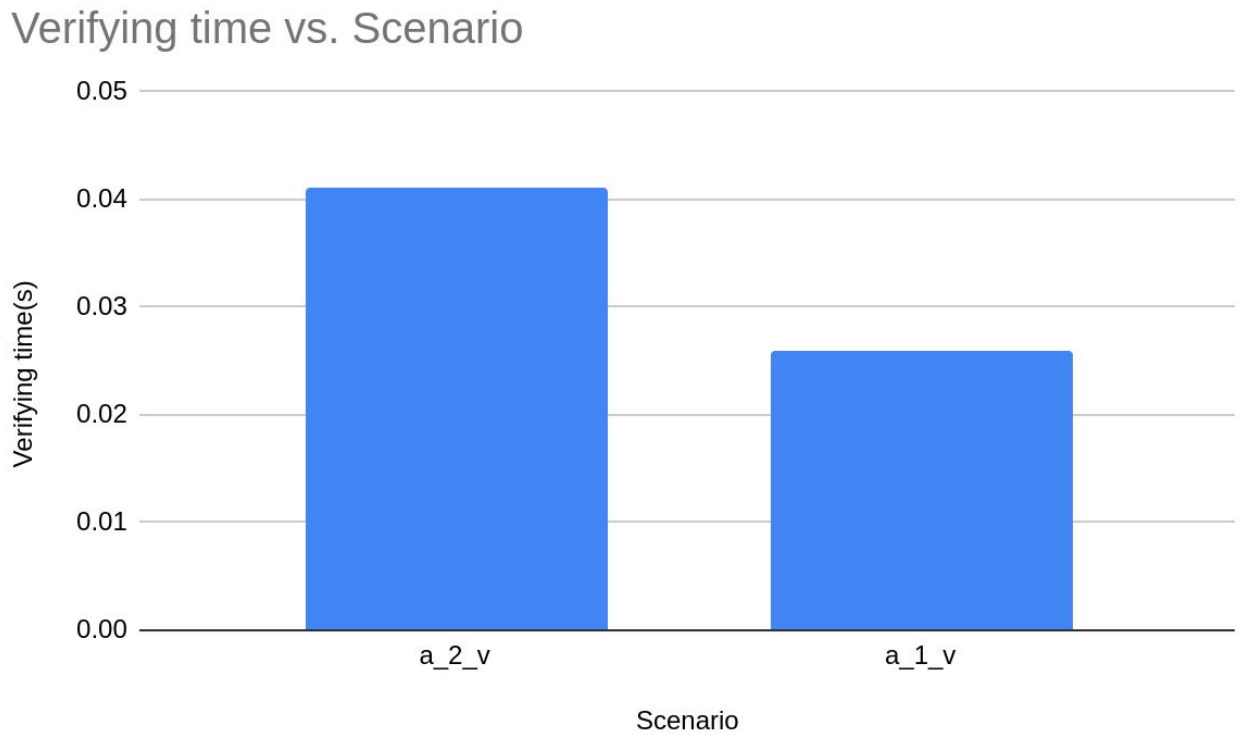

a/na: anonymous/non-anonymous ½: no. of contestants c/v: contestant vs voter

# Evaluation (time)




a/na: anonymous/non-anonymous ½: no. of contestants c/v: contestant vs voter

# Evaluation (time)




a/na: anonymous/non-anonymous ½: no. of contestants c/v: contestant vs voter

# Evaluation (data)

| Scenario | Data sent (bytes) | Data received (bytes) |
|---|---|---|
| a_2_c | 5694 | 7199 |
| a_2_v | 1505 | 7199 |
| a_1_c | 5690 | 6933 |
| a_1_v | 1243 | 6933 |
| na_2_c | 24 | 48 |
| na_2_v | 24 | 48 |
| na_1_c | 24 | 37 |
| na_1_v | 13 | 37 |

a/na: anonymous/non-anonymous ½: no. of contestants c/v: contestant vs voter

# An Alternate Approach

- Multi-key FHE
  - Common FHE public key
  - Each participant generates their private key
- Voter
  - Vector (of length number of contestants) of votes
  - Encrypts the vote vector under the common FHE public key and broadcasts
- Aggregator
  - Participant chosen in a round robin manner
  - Performs homomorphic addition of all encrypted vote vectors
- Decryption
  - Threshold decryption by an honest majority
- Verifiability in zero knowledge
  - Correct vote vectors
  - Correct aggregation of vote vectors

# Conclusions

- Approach 2 is more appealing with complete anonymity but hard to encode FHE evaluation in a zero-knowledge proof
- Implementation limitations
  - pysnark uses libsnark as a backend using SWIG
  - Cannot handle more than 32-bit inputs
  - Small experiments due to time and infrastructure limitations
- Going forward
  - Implementation with libsnark bypassing pysnark