Blockchain-Based Voting System for Secure Distributed Elections

Waseem Ahmad
Harvard College
waseemahmad@college.harvard.edu

Max Peng
Harvard College
mpeng@college.harvard.edu

05/04/25

Abstract

We present a prototype blockchain-based voting system that combines classical distributed-systems principles with modern Web3 tools to deliver secure, transparent, and decentralized elections. Our design builds on an in-class exploration of consensus algorithms,extending our Paxos/Raft/BFT exercises to a Proof-of-Authority (PoA) scheme,and integrates off-chain storage via IPFS alongside Ethereum-style wallet authentication. The system comprises a Next.js client for election management and vote casting, a Node.js/Express server exposing REST endpoints, a custom PoA consensus module in JavaScript, and IPFS for candidate metadata persistence. We validated our approach with end-to-end tests across multiple nodes. Our evaluation demonstrates that PoA reduces computational overhead while ensuring tamper-resistant vote ordering, IPFS guarantees data availability without prohibitive gas costs, and Web3 wallets enable strong voter authentication. This work illustrates how distributed-systems theory can be translated into a real-world application for trustworthy elections, and provides a foundation for further enhancements in scalability, privacy, and more complicated models.

The GitHub repository for this project is available at: `https://github.com/waseemahmad1/cs2620_final`, which also includes presentation slides.

1 Introduction

In an era where digital technology has taken over aspects of daily life, the persistence of largely physical and centralized voting systems stands in stark contrast. Most modern democracies continue to rely on in-person polling places and rudimentary practices. While these traditional approaches offer familiarity, they also present persistent concerns regarding accessibility, scalability, transparency, and public trust in electoral outcomes.

Recent debates surrounding the integrity and inclusivity of elections (ie. the past few presidential elections) demonstrate the need to reimagine how voting systems can leverage modern computational advances. Distributed systems, and blockchain technologies in particular, offer the potential to fundamentally reshape how elections are conducted by introducing tamper-resistance, decentralization, and verifiable auditability. These technical breakthroughs, alongside the potential social implications, inspired our project: as discussed in the motivation subsection below.

1.1 Motivation

The primary motivation behind our project is to demonstrate the feasibility and value of a blockchain-based voting system that addresses the challenges in traditional election infrastructure. Having enjoyed CS 2620's deep dive into consensus and replication, we were particularly excited by Design Exercise 4 (`cs2620_hw4`), where we built a leadership election logic and learned about algorithms like Paxos, Raft, etc, alongside

simplified Byzantine-fault-tolerant replicas across multiple nodes. That exercise taught us how subtle protocol choices affect safety and liveness under node failures.
For our final project, we wanted to take those lessons further:

- **Explore alternative consensus schemes**
- **Leverage IPFS storage** for further data persistence
- **Integrate Web3 wallets** to decentralize voter authentication, and explore blockchain technologies

By building a (very nontrivial) distributed application that combines both academic concepts from class and real-world blockchain technologies, we hope to solidify our understanding of how distributed systems theory scales into practice, a key theme throughout the semester.

1.2 Problem Formulation

More explicitly, for this project, we aim to design and implement a secure, transparent, and distributed election platform that meets the following goals originally in our proposal:

1. Ensures **security** via vote integrity. Each vote, once cast, is immutable and tamper-proof through decentralized ledger storage. We will evaluate against edge cases (ie. adversarial attempts and simulated node failures)
2. Promotes **transparency** while preserving anonymity. All recorded votes are publicly verifiable on the chain and through an independent audit mechanism
3. Eliminates central dependence via **fault tolerance**. The system does not rely on a single trusted authority for validation or storage. Instead, vote blocks are pinned to IPFS across multiple peers, and block inclusion is controlled by a distributed consensus protocol rather than by a central server.

Our approach centers on building a prototype that mirrors the core components of a real-world election system:

- A **web-based voting client** interacting with the voter's Web3 wallet.
- A **distributed backend** based on Ethereum smart contracts (or a private PoA chain) to record vote transactions.
- A **consensus protocol module** inspired by both classical algorithms (Paxos/Raft) and blockchain PoA/PoS variants to validate and order votes.
- **IPFS-based storage** for candidate metadata and ballot definitions, ensuring decentralization.
- An **audit mechanism**, exposing a read-only endpoint that anyone can use to reconstruct and verify the vote history.

Through this project, we will explore how the foundational principles of distributed computing we've learned throughout the semester can be applied to one of society's most foundational processes: ensuring secure, transparent, and fair elections.

2 Methodology

We will provide a very brief overview of our system, then discuss specific mechanisms of certain meaningful aspects, namely the, consensus algorithm, use of IPFS, transactions of ethereum blockchain, and audit mechanism.

2.1 Brief Overview of System

Let us first provide a brief overview of our system, then dig deeper into specific components. Our system consists of two primary components: a JavaScript web client and a Node.js/Express server. The client connects to users' Web3 wallets (via MetaMask), presents a voting interface, and submits ballots to the backend. The server validates and orders incoming votes using a consensus protocol, pins each new block to IPFS for decentralized storage, and records vote transactions on an Ethereum-based blockchain. Finally, an audit API allows anyone to fetch the full chain of IPFS hashes and smart-contract logs to independently verify election integrity.

Now we will delve into the four key mechanisms of the system: the consensus algorithm, IPFS storage, ethereum blockchain, and audit mechanism

2.2 Consensus Algorithm

To coordinate agreement among validator nodes and maintain the integrity of the blockchain ledger, we implement a simplified Proof-of-Authority (PoA) consensus algorithm. Unlike Proof-of-Work (PoW) or Proof-of-Stake (PoS), which require either significant computational effort or financial collateral, PoA relies on a set of pre-designated authority nodes to validate and produce blocks. This approach is particularly well-suited for our use case, where scalability, speed, and fault tolerance are essential, and the threat model assumes partially trusted validators (e.g., campus servers or verified election officials).

In our implementation, a validator is deterministically selected in round-robin fashion from a predefined list of authority keys. Each block includes a transaction, with the voter's election ID, vote data, and a digital signature. The consensus algorithm first verifies a transaction by checking that the voter's signature matches the content of their vote. Once validated, the block is accepted only if its previous hash matches the last block in the chain, its own hash is consistent with its contents, and it was produced by an authorized validator. This lightweight, deterministic approach ensures that vote inclusion is both efficient and tamper-resistant, without incurring the high resource costs of traditional blockchain consensus mechanisms.

2.3 Decentralized Storage with IPFS

In our blockchain-based voting system, we employ the InterPlanetary File System (IPFS) to achieve decentralized and immutable storage of election-related data. IPFS is a peer-to-peer distributed file system that allows for the storage and sharing of data in a decentralized manner, eliminating reliance on centralized servers. This approach enhances the system's resilience against data tampering and single points of failure.

Specifically, IPFS is utilized to store static election data such as candidate information, election metadata, and other relevant documents. By uploading these files to IPFS, each piece of data is assigned a unique content identifier (CID) based on its cryptographic hash. These CIDs are then referenced within our Ethereum smart contracts, ensuring that the data remains tamper-proof and verifiable. This integration allows voters and auditors to retrieve and verify election data directly from the IPFS network, promoting transparency and trust in the electoral process.

The use of IPFS in our system not only decentralizes data storage but also complements the blockchain's immutable ledger by providing a scalable solution for storing larger files that are impractical to store directly on-chain. This hybrid approach leverages the strengths of both technologies to create a robust and secure voting infrastructure.

2.4 Ethereum Blockchain

We set up a simple Ethereum-style smart contract to serve as an off-chain ledger for our vote blocks, but we never run real mining or proof-of-work. Whenever a validator is ready to seal a block, it gathers all of the IPFS content identifiers for that block's votes and calls the contract's `appendBlock` method. That single transaction records the previous block's CID, the list of new CIDs, and a fresh signature from the validator's election-specific wallet key. Because each block header is stored in one lightweight transaction, we avoid gas-heavy mining altogether. Clients and auditors can still retrieve the full history by querying the contract's event logs with standard JSON-RPC calls, giving us the security of an Ethereum-compatible ledger without any of the mining overhead.

2.5 Audit Mechanism

We built a one-stop `/auditProof` endpoint so that anyone can verify an election from start to finish. A request to `/auditProof?electionId=XYZ` makes the server pull all the `appendBlock` events for that election from the smart contract, then for each event fetches the matching JSON blobs from IPFS to rebuild the vote lists. Using the same Merkle-tree code from our blockchain module, the server generates inclusion proofs for every vote and packages everything into a single JSON response. With that data in hand, you can walk through each block's parent–child relationships, check that each validator signature matches the known key, and confirm that every vote really belongs in its block. It's a straightforward way to get full transparency and trust in the results without needing any centralized authority.

3 Development Log

We completed the core implementation over a nine-day period, which we divide into three consecutive three-day phases.

Days 1–3: Blockchain Core and Consensus
Day 1. We first read up to review the blockchain lecture (the 3blue1brown video on youtube was quite helpful as well). We initialized the repository and sketched out our blockchain data structures in `Block.js` and `Blockchain.js`. After defining the block format (prev-hash pointer, vote list) and in-memory chain operations, we committed foundational tests for block linking.

Day 2. With the data structures in place, we turned to consensus. In `PoAConsensus.js` we implemented a simple Proof-of-Authority scheme: a fixed list of validator addresses in `config/validators.json` take turns in round-robin order to sign and seal new blocks. We wrote unit tests to simulate validator failures and message reordering, to help us verify that only the designated authority at each height could produce a valid seal.

Day 3. Next, we built the Express API in `index.js`, exposing endpoints like `/castVote` and `/getChain`. We wired up vote submission to the consensus module and block-sealing logic, ensuring that new blocks were created when vote batches reached a threshold or after a timeout. By the end of Day 3, clients could post signed ballots and receive back block identifiers.

Days 4–6: Decentralized Storage and Authentication
Day 4. In terms of storage, we turned to usind IPFS. We added helper functions in `ipfs.js` to pin JSON blobs (candidate lists, ballots) and retrieve CIDs. We refactored the API so that each new block's payload was pinned to IPFS before being sealed on the blockchain.

Day 5. We shifted focus to voter authentication. In the client code, we scaffolded a Next.js interface and wired in `ethers.js` calls to MetaMask. Voters now sign their vote objects locally; the server validates signatures against the public key before accepting ballots.

Day 6. With most of our core functionality correct, we polished the frontend UI. We implemented the "Create Election," "Quick Vote," "End Election," and "View Ledger" buttons, connecting each to the appropriate API calls. We also added basic error handling for invalid election IDs and signature failures.

Days 7–9: Testing, Audit, and Writing Paper
Day 7. We designed our audit mechanism: an `/auditProof` endpoint that returns the list of block CIDs and on-chain events. We then wrote a small client script to fetch and locally verify each block's parent pointer

generated in `Blockchain.js`.

Day 8. This day was primarily for more testing and starting the notebook. We conducted end-to-end tests: spinning up three PoA nodes, creating elections, casting votes, and simulating node crashes. We wrote up the skeleton of our draft, thinking about how we wanted to split our notebook.

Day 9. In our final day (mostly), we updated the README with setup instructions and double-checked that all tests pass. By the end of Day 9, our prototype was stable and met the goals of integrity, transparency, and decentralization that we had outlined in the beginning.

3.1 Challenges

During development, we encountered numerous challenges. Here we recorded three main ones that took us some time to debug or discuss system design.

- **Validator Rotation Race Conditions.** Early on, our PoAConsensus implementation assumed that validators would always take perfect turns. In practice, slight network delays or concurrent block requests caused two validators to seal blocks at the same height, resulting in divergent chains. We spent a day instrumenting detailed logs and writing unit tests that simulated staggered "heartbeat" messages. Ultimately, we introduced a small grace period and stricter timestamp checks, which eliminated the duplicate-seal problem without sacrificing liveness.
- **IPFS Pinning Latency.** Pinning JSON blobs to IPFS was conceptually straightforward, but during heavy voting periods we saw 2–3 second delays before a CID became retrievable. This caused the client to time out or display broken links. To work around this, we added a preliminary "pending" state for new blocks, retried failed pinning operations in the background, and cached recent CIDs in memory for instant lookups. While this added complexity, it greatly smoothed the user experience.
- **User Authentication UX Trade-Offs.** Integrating MetaMask for vote-signing ensured strong cryptographic guarantees, but it also meant users faced pop-up prompts for every operation. During testing, we noticed frustration when voters had to confirm multiple small transactions in quick succession. To mitigate this, we batched signature requests where possible, grouping candidate selection metadata and ballot confirmation into a single signing step.

4 Results

4.1 General System Design

Previously, in the methodology section, we gave a brief introduction to our system design. Now, we discuss it more deeply. Our voting platform is organized into a five-layer architecture. At the highest level, a *Web client*—implemented in Next.js—provides voters with an intuitive interface for selecting candidates and submitting ballots. This client connects to the voter's Web3 wallet (e.g., MetaMask) to prompt signing and forwards signed vote transactions to our *Server/API layer*. The server, built on Node.js and Express (in `index.js`), exposes REST endpoints such as `/castVote` and `/auditProof` to handle vote submission, chain retrieval, and proof-of-inclusion queries.

Beneath the API layer lies our custom *Blockchain Layer*, realized in `Block.js` and `Blockchain.js`. Each block encapsulates a batch of one or more vote transactions, links to its predecessor via a hash pointer, and stores a timestamp. The *Consensus Layer*, implemented in `PoAConsensus.js`, employs a deterministic proof-of-authority (PoA) scheme: a fixed set of validator identities take turns in round-robin fashion to sign and seal new blocks without requiring energy-intensive mining.

To minimize on-chain storage costs and increase data availability, we integrate an *IPFS Storage Layer*. Candidate metadata, ballot definitions, and election parameters are serialized to JSON, pinned to a local or remote IPFS daemon, and referenced by the blockchain via their content identifiers (CIDs). Finally, an *Audit* *and Verification Layer* allows any participant to fetch the full chain of IPFS CIDs and blockchain events, reconstruct the vote ledger, and therefore verify integrity of the results.

4.2 System Design Choices

Our design choices were guided by the three core goals of vote integrity, transparency, and decentralization (noted in our proposal). Specifically, here are some choices we made:

- **Proof-of-Authority Consensus.** We originally prototyped a Proof-of-Work variant, but found this consensus algorithm was simpler. PoA (in `PoAConsensus.js`) offers fast confirmation and was quite efficient.
- **IPFS for Off-Chain Data.** Storing large JSON blobs directly on Ethereum or a similar chain would be prohibitively expensive. By pinning candidate data and ballots to IPFS—then embedding only the resulting CIDs on-chain—we achieve tamper-proof storage without incurring high gas fees.
- **Ethereum-based Authentication.** Leveraging users' existing Web3 wallets (via the `ethers.js` library) enables strong voter authentication: each ballot is signed with the voter's private key, preventing forgery and enabling non-repudiation.
- **Express API Layer.** A thin REST layer (`index.js`) cleanly separates client concerns from blockchain operations, simplifying error handling, request validation, and the audit-proof endpoint.

Together, these choices balance security, performance, and usability. The PoA consensus ensures rapid block production; IPFS guarantees data availability; and Web3-based vote signing preserves voter sovereignty.

4.3 Example Election Walkthrough

Below is a step-by-step example of the user experience:

1. **Create an Election.** Any user can initiate a new election by entering a unique `Election ID` (e.g., `spring2025_senate`) and clicking "Create Election." This action registers the election ID on the back-end, initializes an empty vote ledger, and pins the election metadata (e.g., candidate list) to IPFS.
2. **Cast a Vote.** Voters navigate to the "Quick Vote" panel, enter the same `Election ID`, enter their vote (as text, and confirm via their Web3 wallet signature. The signed ballot is submitted to the server, where it is validated, added to the current PoA block, and persisted on IPFS and the blockchain.
3. **View Ledger (Ongoing).** At any point, anyone may click "View Ledger," enter the `Election ID`, and fetch the anonymized vote list. The ledger shows each vote alongside the voter address (the unique key which comes from metamask, and is private).
4. **End the Election.** Only the original creator (identified by their wallet address) can terminate the election. By clicking "End Election" and confirming the signature, the creator signals the PoA validators to seal the final block and mark the election as closed.
5. **View Results.** After the election is ended, "View Ledger" additionally displays the vote tallies and highlights the winner at the top of the list. From this point, no further votes can be cast under that election ID.

5 Conclusion

We have designed and implemented a functional blockchain-backed voting platform that meets our core proposal goals of vote integrity, transparency, and decentralization. By combining a lightweight Proof-of-Authority consensus protocol with IPFS-based off-chain storage and Ethereum-style wallet authentication, our system achieves publicly verifiable vote data without relying on a single trusted authority and is distributed. Throughout development, we navigated challenges in dealing with IPFS and making system choices, ultimately delivering a prototype that demonstrates the practical viability of distributed-ledger technology for elections. We hope our work inspires further research into scaling blockchain voting to larger electorates, integrating advanced privacy techniques, and exploring hybrid consensus approaches. Thank you to the CS 2620 Professor Waldo and all the teaching staff for helping make this class an incredible experience.