A Secure Ethereum Voting Platform for Campus Elections

Nour Al-Sheikh
Earlham College
njalshe23@earlham.edu

May 2026

Abstract

Campus elections need a voting workflow that keeps identity private while making the final record inspectable. This report describes a complete capstone system for that setting. The system combines a React voter/admin/audit interface, an off-chain authentication service, a read-only audit API, SQLite-backed event indexing, monitoring, deployment scripts, and a single Solidity smart contract, `ElectionManager`. The delivered system runs as a complete one-node laptop deployment and includes configurable 1 to 4 node Clique/geth deployment support. The end-to-end workflow covers authentication, wallet registration, commit/reveal voting, admin finalization, and public audit. Voter eligibility is verified through an EIP-712 signed proof issued by the authentication service; campus identity never appears on-chain. The automated verification suite contains 69 passing tests across the contract, authentication service, audit API, and shared TypeScript package. An IRB exemption was approved for the mock-election usability study. The report presents the implemented system, validation evidence, and evaluation design for campus-scale usability and trust assessment.

Keywords: blockchain voting, Ethereum, campus elections, smart contracts, commit/reveal, EIP-712, usability evaluation, private blockchain

Contents

1 Introduction

Paper ballots, most of the time, work. The 2000 U.S. presidential election showed how fast that confidence can collapse: ambiguous punch-card ballots in Florida triggered weeks of recounts and a Supreme Court decision that effectively chose the president (National Research Council, 2006). At the campus level the stakes are lower, but the fragility is familiar. Manual recounts drag on, eligibility disputes resurface, and administrative overhead piles up with each contested cycle. Even when outcomes are ultimately accepted, the delay itself chips away at confidence in student governance (Specter et al., 2020; Mannonov & Myeong, 2023).

Digital voting platforms promise efficiency, but centralized solutions bring their own risks. Voters must trust backend infrastructure they cannot inspect. Real-world deployments have shown how quickly security failures or opaque processes can damage institutional credibility (Specter et al., 2020). Trust, once lost, is expensive to rebuild. In response, researchers turned to blockchain-based voting as a way to make elections transparent, verifiable, and resistant to tampering (Kshetri & Voas, 2018; Sharma et al., 2022). Ethereum provides a programmable execution environment: voting logic goes directly into smart contracts, auditable by any third party. Fully public Ethereum networks, however, are rarely a good fit for campus elections. Transaction costs, latency, governance complexity, and privacy concerns quickly outweigh their benefits in small, bounded communities (Jafar et al., 2021). Deploying a *private* Ethereum network governed by campus stakeholders keeps the cryptographic guarantees of blockchain voting while sidestepping the operational drawbacks of open participation. The delivered system uses EIP-712 signed voter registration, commit/reveal balloting, and a read-only audit path over public chain events. It runs as a complete one-node laptop deployment and includes configurable 1 to 4 node Clique/geth deployment support.

The workflow is concrete. A voter authenticates through campus email, binds a wallet, registers on-chain through an EIP-712 eligibility proof, commits a sealed ballot, reveals it after the voting window closes, and receives a receipt. An administrator finalizes the election. The audit API indexes public contract events so voters and observers can inspect the record without exposing campus identity on-chain.

The scope of this report is the implemented capstone system, its validation evidence, and the approved mock-election evaluation design. No binding campus election is part of this capstone. The mock-election study provides controlled conditions for assessing usability and perceived trust in a campus-scale voting workflow.

The rest of this report is organized as follows. Section 2 reviews the relevant literature; Sections 3 and 4 present the data architecture and a graphical abstract of the voter workflow; Section 5 details the evaluation methodology; and Section 6 outlines future work.

2 Literature Review

The platform's design draws on two decades of work in verifiable voting, blockchain architecture, and usability research. This section traces the key ideas and the lessons, including cautionary ones, that shaped the decisions described in Sections 3 through 5.

2.1 Verifiability and Transparency

A central promise of electronic voting is that cryptographic mechanisms can provide *verifiability*: the ability for voters, auditors, or the public to confirm that an election was conducted correctly without trusting a single authority. Helios, one of the earliest web-based voting systems to pursue this goal, introduced homomorphic tallying so that encrypted ballots could be aggregated without ever decrypting individual votes (Adida, 2008). Cortier et al. (2014) later showed that Helios's verifiability guarantees hold even under weaker trust assumptions, broadening the conditions under which the system remains secure.

These contributions established a design principle that motivates the present project: election integrity should not depend on trusting the server. In a blockchain context this maps naturally onto the immutable, publicly auditable ledger that every node maintains. The platform described here enforces verifiability through on-chain event logs (`VoteCommitted`, `VoteRevealed`) that any observer can replay independently.

2.2 Blockchain Voting Architectures and Deployment Lessons

Blockchain-based voting systems have been built on several distinct architectures, each trading off privacy, scalability, and implementation complexity differently.

Commit/reveal schemes. Voters submit a cryptographic hash of their ballot during a commit phase, then reveal the plaintext vote after the voting window closes. BroncoVote, an Ethereum-based campus election system piloted at Santa Clara University, adopted this approach for its straightforward implementation and low computational overhead (Dagher & et al., 2018). Commit/reveal is scalable but offers only moderate privacy: individual votes become visible after the reveal phase (McCorry et al., 2017). This project uses

a commit/reveal protocol and accepts that trade-off as appropriate for the campus threat model, where the primary concern is preventing premature disclosure rather than guaranteeing unconditional ballot secrecy. For a campus election, this is enough.

Homomorphic tallying. Homomorphic encryption allows ballots to be aggregated directly in encrypted form, so individual votes are never revealed. Helios pioneered this approach in electronic voting (Adida, 2008; Cortier et al., 2014), but the computational overhead limits scalability in large elections. This project does not use homomorphic tallying; it is noted here as a candidate for future enhancement.

Zero-knowledge proof schemes. Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) provide the strongest available privacy guarantees, allowing voters to prove ballot validity without revealing their choices. Academic prototypes such as OzKoin (Tsang et al., 2016) and Hawk (Kosba et al., 2016) showed theoretical feasibility but also exposed substantial practical challenges in proof generation time and trusted-setup requirements. Like homomorphic tallying, zk-SNARKs are scoped as future work.

Table 1 summarizes the key trade-offs across these three approaches.

Table 1: Comparison of blockchain voting tallying methods.

Method	Priv.	Scale	Complex.
Commit/reveal	Mod.	High	Low
Homomorphic tally	High	Mod.	High
zk-SNARK	Very high	Low	Very high

Several institutional pilots offer concrete lessons. BroncoVote handled roughly 1,200 student votes at Santa Clara University, showing that blockchain can work at campus scale despite initial friction with wallet management and voter onboarding (Dagher & et al., 2018). The city of Tsukuba, Japan, ran a blockchain-based election for social-program funding on Ethereum and AWS infrastructure; turnout was comparable to traditional methods, extending the evidence for blockchain voting beyond university pilots (Tsukuba City & AWS, 2018). Both deployments showed that user education and streamlined authentication workflows matter at least as much as the underlying cryptography.

2.3 Privacy Approaches and Usability Challenges

Ethereum-based decentralized applications (DApps) typically require voters to interact through wallet interfaces such as MetaMask. Studies report significant drop-off rates during the onboarding process, especially during wallet installation and private-key management. In a multi-university usability study, Mannonov & Myeong (2023) recorded a 30 to 40% drop-off rate during initial wallet setup, driven primarily by complexity and uncertainty about security practices, with nearly half of potential voters never casting a ballot.

To reduce this friction, streamlined UI/UX patterns have emerged. Polys, a blockchain voting platform from Kaspersky Lab, introduced wizard-style step-by-step guidance that cut setup time and improved completion rates (Kaspersky Lab, 2019). Empirical data suggest that user adoption increases when voters perceive transparency and trustworthiness, even if the initial onboarding requires extra effort (Mannonov & Myeong, 2023; Schiarelli & Dupuis, 2023).

The present platform adopts several of these mitigations: a magic-link authentication flow that eliminates passwords entirely, a wizard-style DApp interface, and contextual help throughout the voting process. The planned evaluation (Section 5) explicitly measures onboarding friction through task-completion time and post-task interviews.

2.4 Failure Modes and Trust Assumptions

Security audits of early blockchain voting platforms have surfaced vulnerabilities that caution against over-reliance on any single layer of defense. The Voatz mobile voting pilot, deployed during the 2018 West Virginia elections, underwent a rigorous security analysis that uncovered flaws in the mobile client allowing vote tampering and potential privacy breaches (Specter et al., 2020; West Virginia Secretary of State, 2018). Endpoint security and client-side code quality matter as much as the blockchain layer.

Static and dynamic analysis of Ethereum smart contracts has also identified common vulnerability classes, including reentrancy, improper access control, and integer overflow, that specialized audit tools can catch before deployment (Moreno-Sánchez et al., 2018; Koch et al., 2018).

These findings shape the security posture of the present system in two ways. First, the platform restricts the attack surface by using a private network where only allowlisted addresses can submit transactions. Second, the smart contract is kept deliberately simple: a single `ElectionManager` with clearly

scoped functions. Additional static analysis, fuzzing, and threat modeling are scoped as future hardening steps (Section 6).

3 Data Architecture Diagram

Figure 1 presents the data architecture of the voting platform. The diagram is organized into zones that map to the system's trust boundaries and operational phases. The one-node laptop deployment exercises the same contract, web application, authentication service, audit service, monitoring, and event-indexing boundaries used by the configurable 1 to 4 node Clique/geth deployment support.
All voter registration, ballot commitment, and tally logic reside in a single Solidity smart contract, `ElectionManager`, deployed once on the private network. The contract uses `electionId` namespaces to scope each election independently, so multiple elections can run on the same contract without interference. The architecture spans three protocol phases plus a read-only audit path.

Register. The voter submits their campus email address to the authentication service, which sends a one-time magic link containing a signed JSON Web Token (JWT). Clicking the link opens the DApp, which prompts the voter's MetaMask wallet to sign a challenge. The authentication service then produces an EIP-712 typed-data signature binding the voter's wallet address, election identifier, expiration timestamp, and a unique nonce (a one-time random value used to prevent replay attacks) to the campus issuer key. The voter's wallet submits this signature to `ElectionManager.register()`, which recovers the signer via ECDSA and checks it against the stored issuer address. On success, the contract marks the wallet as eligible and records the digest to prevent reuse. No personally identifiable information (PII) reaches the chain; identity stays entirely within the off-chain Identity Zone. The current local deployment uses a development email-verification mode; SMTP delivery is a future hardening step for a broader pilot.

Commit. The eligible wallet submits a single commit transaction to `ElectionManager.commitVote()`. The commit payload is a cryptographic hash computed client-side:

$$commitHash = \texttt{keccak256}(elId,\ sender,\ optId,\ nonce)$$

Including the election identifier and the sender's address in the preimage prevents cross-election replay and vote-swapping attacks. The contract enforces a one-commit-per-address constraint: a second call from the same wallet reverts with `AlreadyCommitted`. A receipt containing the commit hash is returned to the voter for later verification.
To mitigate timing-correlation attacks, where a watcher could observe a voter sign a MetaMask transaction and then see a `VoteCommitted` event moments later, the DApp introduces a randomized delay between the client-side signing action and the on-chain commit submission. This weakens the temporal link between a known voter and a specific commit event.

Reveal & Tally. After the commit window closes and the reveal window opens, voters call `ElectionManager.revealVote()` with their plaintext `optionId` and the nonce they used during the commit phase. The contract recomputes the hash, compares it to the stored commit, and, if the values match, increments the tally for the chosen option and emits a `VoteRevealed` event. Results are gated: `getTally()` and `getAllTallies()` revert until the reveal window has closed, preventing partial-result leakage from influencing remaining voters. The configurable Clique/geth deployment supports 1 to 4 local validator nodes for multi-node operation. During one-node laptop validation, the same contract behavior was tested through the complete product workflow.
Elections close in one of two ways: the reveal window expires (time-based), or an administrator manually closes the election after confirming that all eligible voters have participated (threshold-based).

Audit Outputs. On-chain event logs (`VoterRegistered`, `VoteCommitted`, `VoteRevealed`) are immutable and publicly readable. Voter-held receipts (commit hashes) let individuals verify their own votes against the ledger. The implemented audit API backfills and tails these logs into a read-only SQLite index for the public audit view. An auditor or researcher can use that index, together with exported metrics, as the basis for later evaluation analysis.

4 Graphical Abstract

Figure 2 gives a high-level summary of the voter experience in four steps: authentication via a campus email link, sealed (committed) voting on-chain, reveal and tally through the private blockchain layer, and a public record that the voter can verify at any time using their receipt.
The graphical abstract is aimed at a non-specialist audience. Two properties are emphasized throughout: (1) voter identity never appears on-chain, and (2) the commit/reveal mechanism keeps vote content private until the designated reveal window.

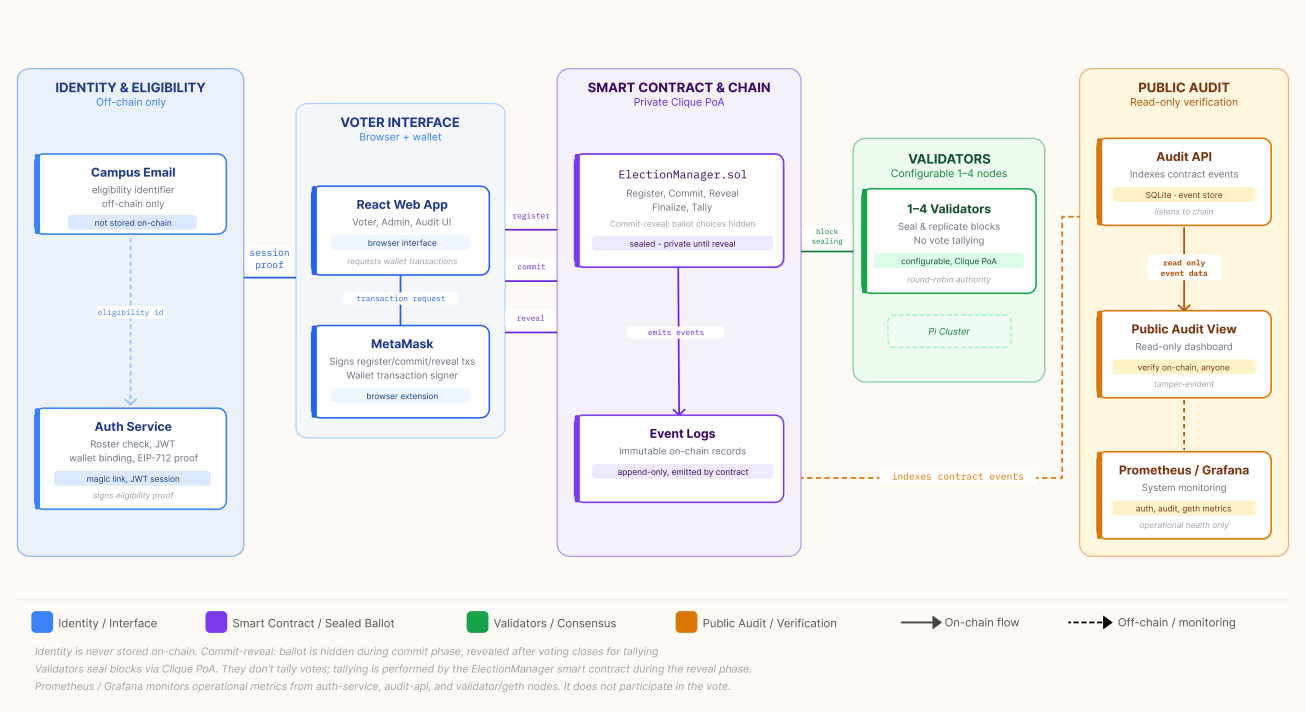

Figure 1: Data architecture diagram. Solid arrows represent on-chain writes or protocol messages; dashed arrows represent asynchronous triggers or read operations. The five zones are: Identity Zone (off-chain, blue), Voting Zone (client-side, blue), Private Blockchain with the `ElectionManager` contract (purple), Clique/geth Validators (configurable 1 to 4 node support, green), and Outputs (orange).

5 Methods

This section separates the implemented system from the evaluation protocol. The capstone deliverable runs as a complete one-node laptop deployment. It also includes configurable 1 to 4 node Clique/geth deployment support for multi-node operation. The validation work checks the product path end to end: authentication, wallet registration, commit, reveal, admin finalization, receipt display, and public audit.

5.1 Study Design Overview

The technical validation confirms that each component runs together: deployed `ElectionManager` contract, React web application, authentication service, audit API, SQLite indexes, and Prometheus/Grafana monitoring. The approved participant evaluation follows a single-group, within-subjects design. Recruited participants complete a mock voting task on the deployed platform, then fill out a post-task survey and an optional short interview. Quantitative metrics (task-completion time, SUS score, trust rating, operational metrics) and qualitative observations (interview excerpts, open-ended survey responses) are collected in a single session per participant.

5.2 Participant Recruitment

Participants will be recruited from the Earlham College student body through campus mailing lists and in-class announcements. The target sample is 15 to 20 participants, a range grounded in the usability-testing literature: Faulkner (2003) showed that samples of 15 or more users reliably uncover at least 90% of usability problems in formative evaluations, and Lewis (1994) found that problem-discovery rates plateau rapidly beyond moderate sample sizes. Inclusion criteria are: (a) current enrollment at Earlham College and (b) a valid `@earlham.edu` email address. No prior blockchain or cryptocurrency experience is required; participants with varied technical backgrounds are welcome, to better reflect the intended user population.

Participation is voluntary. All participants will provide informed consent before the session begins.

5.3 Mock Election Setup

The mock election simulates a campus ballot with two to three candidate choices on a single question (e.g., "vote for your favorite fruit"). The one-node laptop deployment exercises this flow locally. The system includes the following infrastructure:

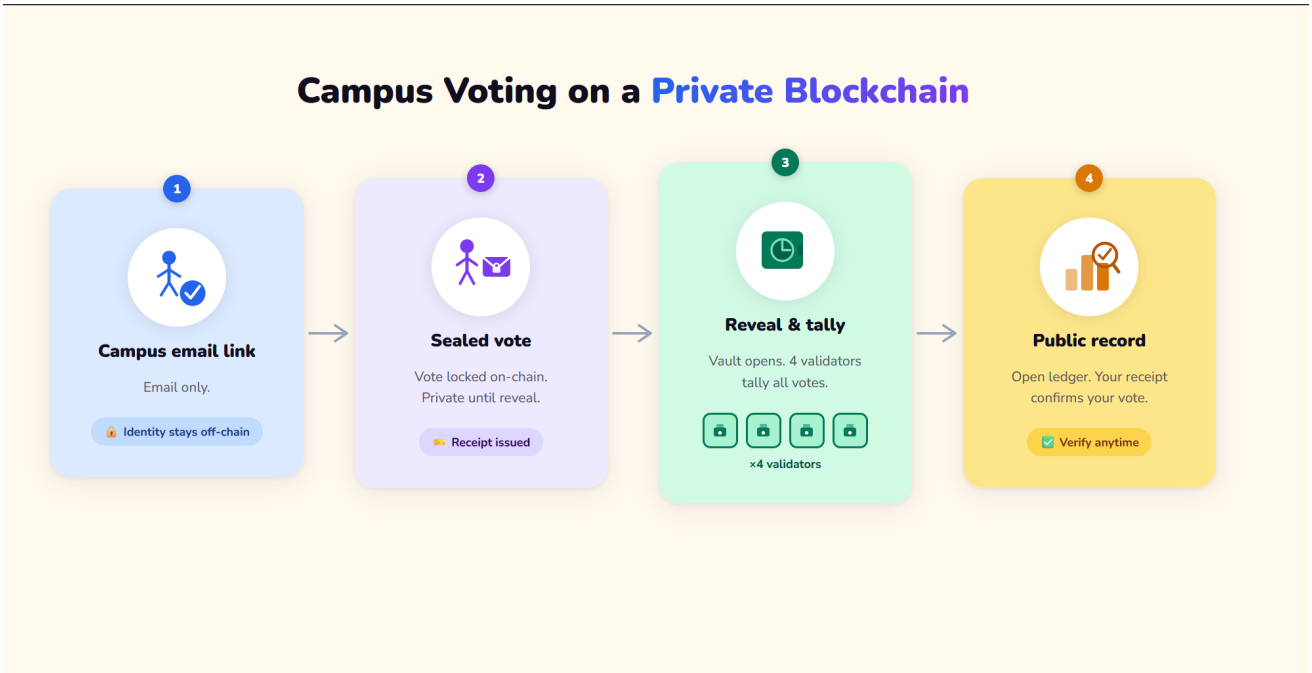

Figure 2: Graphical abstract: Campus Voting on a Private Blockchain. The four color-coded panels correspond to Identity (off-chain, blue), Private Vote (on-chain and sealed, purple), Private Blockchain consensus (green), and Public Audit (amber). The validator panel represents configurable 1 to 4 node Clique/geth deployment support, while the one-node laptop deployment exercises the same voter-facing protocol and receipt model.

Deployment modes.
A complete one-node laptop deployment workflow and configurable 1 to 4 node geth/Clique support through `NODE_COUNT=1..4`. The multi-node configuration provides the deployable consensus shape for later physical hosts with operator-provided host and key material.

Authentication service.
A Node.js/Express backend that manages roster-based sign-in, JWT issuance, wallet binding, server-side EIP-712 proof signing, health checks, and Prometheus metrics. The same service holds the EIP-712 issuer private key in the local deployment. Production SMTP delivery and stronger key-management integration remain future hardening steps.

Smart contract.
A single `ElectionManager` contract (Solidity 0.8.24), deployed via the Hardhat framework and compiled with the optimizer enabled at 200 runs. The contract manages voter registration (EIP-712 signature verification, anti-replay via consumed digests), commit/reveal voting (one commit per address, hash-verified reveal), time-gated tally reads, and role-based access control for election administration. OpenZeppelin libraries provide `AccessControl`, `EIP712`, and `ECDSA`. TypeChain generates TypeScript bindings consumed by the test suite and front-end DApp.

Front-end DApp.
A React/Vite web application that talks to the voter's MetaMask wallet extension. The interface follows a wizard-style flow in which the voter enters their email, verifies the session, connects a wallet, registers, selects a ballot option, commits, reveals, and receives a receipt. The application also includes admin and public audit routes.

Audit API.
A Node.js/Express read-only service that backfills and tails `VoterRegistered`, `VoteCommitted`, `VoteRevealed`, and `ElectionFinalized` events into a SQLite cache. The public audit page reads from this API rather than directly from the chain RPC endpoint.

Monitoring.
Prometheus and Grafana collect service health, proof issuance, commit/reveal activity, gas-related metrics, and block-time panels. The monitoring stack is exercised during one-node laptop validation and supports later multi-node runs.

Implemented system. The capstone system is complete as a one-node laptop deployment and

supports configurable 1 to 4 node Clique/geth deployment. The implemented stack includes `ElectionManager`, the authentication service, the audit API, the React web application, the shared TypeScript package, deployment and seed scripts, cluster lifecycle scripts, backend service start/check/stop scripts, and the Prometheus/Grafana monitoring stack. The full product workflow has been manually validated: authentication, wallet registration, EIP-712 registration, commit, reveal, admin finalization, receipt display, and public audit. The public audit route reads from the audit API, which re-indexes from block 0 and tails new events. Grafana panels for block time, commit/reveal gas, commit/reveal event activity, and eligibility proofs have been populated during local testing.

Validation. Automated validation totals 69 passing tests: 35 contract and integration tests, 20 authentication-service tests, 7 audit-API tests, and 7 shared-package tests. The web application also builds successfully.

Table 2: Validation status.

Check	Result
Contract + integration tests	35 passing
Auth-service tests	20 passing
Audit API tests	7 passing
Shared package tests	7 passing
Web-app build	Success
Full product flow	Authentication, wallet, register, commit, reveal, admin finalize, audit
Monitoring	Grafana panels populated locally

5.4 Task Protocol

Each participant session lasts roughly 15 to 20 minutes and follows a fixed protocol:

1. **Briefing** (2 min). The facilitator explains the study purpose, obtains informed consent, and answers questions.
2. **Authentication task** (self-paced). The participant enters their campus email, receives the magic link, and completes wallet binding. Task-completion time is recorded from email submission to successful on-chain wallet registration.
3. **Voting task** (self-paced). The participant selects a candidate and submits a commit transaction. Time is recorded from ballot screen load to commit confirmation.
4. **Reveal task** (self-paced). After a short simulated waiting period the participant reveals their vote. The system verifies the commitment and displays the receipt.
5. **Post-task survey** (5 min). The participant completes the System Usability Scale (SUS) questionnaire (Brooke, 1996) and a seven-point Likert-scale perceived-trust instrument adapted from Mannonov & Myeong (2023).
6. **Semi-structured interview** (5 min, optional). Open-ended questions on pain points, confusing steps, and overall impressions.

5.5 Data Collection and Instruments

Data come from three sources.

Usability and trust survey. The SUS is a ten-item questionnaire yielding a composite score between 0 and 100; a score of 68 or above is conventionally regarded as above-average usability (Brooke, 1996). Perceived trust is measured on a seven-point Likert scale adapted from the Technology Acceptance Model (TAM) instrument used by Mannonov & Myeong (2023). Both instruments are administered through Google Forms immediately after the voting task.

On-chain performance logs. Prometheus and Grafana collect service health, proof issuance, commit/reveal activity, event indexing, gas-related metrics, and block-time panels. These signals make the system observable during the one-node deployment workflow and provide the instrumentation needed for later multi-node runs. Physical multi-host performance measurements belong to a later deployment study.
For the mock-election study, the useful performance measures are transaction confirmation time, commit and reveal latency, gas use per vote, event-indexing delay, and service health during the session. Those measures can be exported from the monitoring stack and audit API alongside usability and trust data.

5.6 Software Stack

Table 3 summarizes the software components used across development, deployment, and evaluation.

5.7 Analysis Plan

Usability. SUS scores will be computed per the standard protocol and reported as mean, standard deviation, and distribution histogram. The a priori benchmark is a mean SUS score ≥ 70, indicating good usability. Trust ratings will be reported as median and

Table 3: Software stack.

Component	Tool / Version
One-node deployment	Laptop deployment workflow
Multi-node support	Configurable geth/Clique nodes, `NODE_COUNT=1..4`
Contract language	Solidity 0.8.24
Dev framework	Hardhat
Contract libraries	OpenZeppelin 5.x
Type-safe bindings	TypeChain (ethers-v6)
Front end	React
Wallet	MetaMask extension
Auth backend	Node.js + Express + JWT + SQLite
Audit backend	Node.js + Express + SQLite index
Monitoring	Prometheus + Grafana
Gas profiling	hardhat-gas-reporter
Containerization	Docker
Survey platform	Google Forms

interquartile range. Task-completion times (authentication, voting, reveal) will be reported as median and 95th-percentile values.

Performance. Transaction confirmation time, commit and reveal latency, gas use, event-indexing delay, and service-health metrics will be pulled from Prometheus exports, audit API records, and application logs, then presented as summary statistics and time-series plots.

Security (preliminary). The automated validation suite exercises the contract, authentication service, audit API, and shared TypeScript package. Additional static analysis, fuzzing, STRIDE threat modeling (Shostack, 2014), and penetration testing are scoped as future hardening work.

Expected outputs. The analysis will produce the following artifacts:

- SUS score histogram and descriptive statistics table.
- Trust-rating bar chart (pre- vs. post-task, if both are collected).
- Task-completion-time box plots for each phase (authentication, commit, reveal).
- Performance time-series dashboards (block time, throughput).
- Gas-usage summary table.
- Security-hardening notes for findings that emerge during validation.

5.8 Ethical Considerations

Because the evaluation involves human participants, the mock-election study was submitted for Institutional Review Board review. An IRB exemption was approved for the usability study. The task is non-political and minimal-risk, such as voting for a favorite fruit. Campus identity stays off-chain, and the study reports only aggregated, anonymized results. Participation is voluntary. All participants provide informed consent before the session begins. No vote content or personally identifiable information is stored on-chain.

5.9 Limitations and Scope

Four design boundaries shape how the results should be read.
The mock election removes the social pressures of a contested race, so the results speak to system usability and technical performance, not political behavior. A sample of 15 to 20 participants is standard for formative usability testing (Faulkner, 2003) and sufficient to surface the majority of interface problems, though not for broad statistical generalization. The commit/reveal protocol provides moderate privacy: votes become public after the reveal phase. Homomorphic tallying and zk-SNARKs would strengthen ballot privacy further; both are scoped as extensions in Section 6, not prerequisites. The completed validation uses a one-node laptop deployment workflow. Multi-host consensus behavior, physical-node resource constraints, and wide-area network effects belong to a later deployment study.

6 Future Work

Several extensions are noted throughout the report. This section collects them in one place so their scope and priority are explicit.

Physical multi-node deployment. The codebase supports configurable 1 to 4 node Clique/geth deployment. A natural operational extension is to run that configuration across physical Raspberry Pi validators and collect block-production, availability, recovery, and resource-use measurements under multi-host conditions.

Production email and key management. The current local deployment uses a development email-verification mode. A broader pilot should connect production SMTP delivery and harden issuer-key handling beyond a local environment variable, for example through a managed secret store or hardware-backed signing process.

Stronger ballot privacy. The current commit/reveal protocol reveals individual votes after the reveal window. Homomorphic tallying, as in Helios, or zk-SNARKs would allow the system to publish only aggregate results. Both carry significant implementation and performance costs; a feasibility study is the logical next step.

Alternative authentication for device-limited populations. The current registration flow assumes every voter has access to a personal device with a browser and MetaMask. Populations without such access would require an alternative path, such as a QR-code ticket issued at a physical kiosk. This path introduces new trust assumptions, including ticket interception and kiosk security, that would need their own threat analysis.

Additional security hardening. A systematic STRIDE analysis (Shostack, 2014), Slither static analysis, Echidna fuzz testing, and penetration testing are appropriate next hardening steps for the deployed system.

References

Adida, B. (2008). Helios: Web-based open-audit voting. In *Proceedings of the 17th USENIX security symposium* (pp. 335–348).

Brooke, J. (1996). SUS: A "quick and dirty" usability scale. *Usability Evaluation in Industry*, 189–194.

Cortier, V., Galindo, D., Glondu, S., & Izabachène, M. (2014). Election verifiability for Helios under weaker trust assumptions. In *Proceedings of the european symposium on research in computer security (ESORICS)* (pp. 327–344).

Dagher, G., & et al. (2018). BroncoVote: Secure voting system using Ethereum's blockchain. In *Proceedings of the international conference on information systems security and privacy (ICISSP)* (pp. 221–230).

Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, *35*(3), 379–383. doi: 10.3758/BF03195514

Jafar, U., Haider, A., & Ali, R. (2021). Blockchain for electronic voting system—review and open research challenges. *Sensors*, *21*(17), 5874.

Kaspersky Lab. (2019). *Polys: Blockchain-based voting platform for online and offline elections.* `https://polys.me/assets/Polys_Whitepaper.pdf`.

Koch, C., Wachsmann, S., & Boneh, D. (2018). ZoKrates: A toolbox for zkSNARKs on Ethereum. In *Proceedings of the 2018 IEEE european symposium on security and privacy workshops* (pp. 47–54).

Kosba, A., Miller, A., Shi, E., Wen, Z., & Papamanthou, C. (2016). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of the IEEE symposium on security and privacy* (pp. 839–858).

Kshetri, N., & Voas, J. (2018). Blockchain-enabled e-voting. *IEEE Software*, *35*(4), 95–99.

Lewis, J. R. (1994). Sample sizes for usability studies: Additional considerations. *Human Factors*, *36*(2), 368–378. doi: 10.1177/001872089403600215

Mannonov, K. M., & Myeong, S. (2023). Citizens' perception of blockchain-based e-voting systems: A TAM approach. *Sustainability*, *16*(11), 4387.

McCorry, P., Shahandashti, S. F., & Hao, F. (2017). A smart contract for boardroom voting with maximum voter privacy. In *Proceedings of the 2017 ACM workshop on privacy in the electronic society* (pp. 65–75).

Moreno-Sánchez, J., Atlas, K., & Kiayias, A. (2018). Slate: Statically analyzable smart contracts. In *Proceedings of the 2018 IEEE symposium on security and privacy* (pp. 395–414).

National Research Council. (2006). *Asking the right questions about electronic voting.* Washington, DC: The National Academies Press. doi: 10.17226/11449

Schiarelli, V., & Dupuis, M. (2023). Evaluating the public perception of a blockchain-based election. In *Proceedings of the ACM SIGITE conference on information technology education* (pp. 121–130).

Sharma, T., Gupta, R., & Patel, S. (2022). Blockchain-based e-voting systems: A technology review. *Electronics*, *13*(1), 17.

Shostack, A. (2014). *Threat modeling: Designing for security.* Wiley.

Specter, M. A., Koppel, J., & Weitzner, D. (2020). The ballot is busted before the blockchain: A security analysis of Voatz, the first internet voting application used in U.S. federal elections. In *Proceedings of the 29th USENIX security symposium* (pp. 335–348).

Tsang, P., Quinlivan, S., & Li, X.-Y. (2016). OzKoin: A zero knowledge Ethereum voting system. In *Proceedings of the 2016 ACM CCS workshop on blockchain security and privacy* (pp. 19–26).

Tsukuba City, & AWS. (2018). *Blockchain and personal ID-based online voting in Tsukuba's society 5.0 trials.* Case Study.

West Virginia Secretary of State. (2018). *West Virginia mobile voting pilot.* Press Release and Reports.