# Development of a Secure E‑Voting System Using Blockchain Technology (BLOCKELECT)

Department of Computer Science, Olusegun Agagu University of Science and Technology (OAUSTECH), Ondo State

Author: Ughili Samuel Adiwu

Supervised by: [Insert Supervisor’s Name]

Date: [Insert Month, Year]

ABSTRACT

Electronic voting promises efficiency, accessibility, and rapid result tabulation, yet continues to face critical challenges—chiefly security, transparency, coercion-resistance, and trust. This thesis presents BLOCKELECT, a blockchain‑based secure e‑voting system that integrates five advanced technologies: (1) government‑grade identity verification via a mock National Identification Number (NIN) verification workflow; (2) a real‑time analytics dashboard for transparent monitoring; (3) multi‑signature security for administrative governance; (4) a Progressive Web App (PWA) for offline resilience and mobile‑first access; and (5) decentralized storage using IPFS. It additionally explores a privacy layer with zero‑knowledge proof concepts to preserve ballot secrecy while ensuring verifiability. The system is built on Ethereum smart contracts (Solidity), Node.js/Express backend, and a standards‑compliant web front end. Empirical evaluation covers functionality, performance on a local blockchain (Ganache), and security considerations (role‑separation via multi‑sig, integrity via blockchain, and privacy via ZK concepts). Results demonstrate that the architecture improves integrity, auditability, and fault tolerance compared to centralized alternatives, while remaining accessible and extensible for academic and real‑world trials. The thesis contributes a practical blueprint for secure digital elections within resource‑constrained settings.

Keywords: e‑voting, blockchain, Ethereum, smart contracts, zero‑knowledge proofs, IPFS, PWA, multi‑signature governance, identity verification, analytics.

ACKNOWLEDGEMENTS

I gratefully acknowledge the support of my supervisor, the Department of Computer Science at OAUSTECH, my colleagues, and my family. I also appreciate the open‑source communities behind Ethereum, Truffle, Web3, IPFS, and Chart.js. Their tools made this work possible.

TABLE OF CONTENTS

Abstract

Acknowledgements

List of Figures

List of Tables

Chapter One: Introduction

1.1 Background of the Study

1.2 Problem Statement

1.3 Aim and Objectives

1.4 Research Questions

1.5 Scope of the Study

1.6 Significance of the Study

1.7 Methodology Overview

1.8 Project Deliverables

1.9 Thesis Organization

Chapter Two: Literature Review

2.1 Electronic Voting: Concepts and Challenges

2.2 Threat Model for Digital Elections

2.3 Blockchain Technology: Foundations and Properties

2.4 Smart Contracts and Decentralized Applications

2.5 Consensus Protocols and Security Implications

2.6 Privacy‑Preserving Voting: Mix‑nets, Homomorphic Encryption, and ZK Proofs

2.7 Decentralized Storage: IPFS Principles and Use Cases

2.8 Progressive Web Apps for Civic Tech

2.9 Multi‑Signature Governance Models

2.10 Related Work and Comparative Analysis

2.11 Research Gap and Conceptual Framework

2.12 Summary

Chapter Three: Methodology and System Design

3.1 Research Methodology

3.2 Requirements Analysis

3.3 Use Case Modeling

3.4 System Architecture

3.5 Data Models and Smart Contract Design

3.6 Security Architecture and Threat Mitigation

3.7 Privacy Model and ZK‑Proof Conceptualization

3.8 NIN Verification Flow and API Design

3.9 IPFS Integration Design

3.10 PWA Architecture

3.11 Analytics Dashboard Architecture

3.12 Development Tools and Environments

3.13 Summary

Chapter Four: Implementation and Results

4.1 Implementation Environment

4.2 Smart Contract Implementation

4.3 Backend/API Implementation

4.4 Frontend Implementation

4.5 NIN Verification Module

4.6 IPFS Module

4.7 PWA Features

4.8 Analytics Dashboard

4.9 Privacy (ZK) Prototype Module

4.10 Testing Strategy and Test Cases

4.11 Performance Evaluation

4.12 Security Evaluation

4.13 Usability Evaluation

4.14 Deployment Considerations

4.15 Comparative Results and Discussion

4.16 Summary

Chapter Five: Conclusion and Future Work

5.1 Summary of Findings

5.2 Contributions

5.3 Limitations

5.4 Recommendations

5.5 Future Work

5.6 Conclusion

References

Appendices

LIST OF FIGURES

Figure 3.1 System High‑Level Architecture (Textual)

Figure 3.2 Use Case Diagram (Textual)

Figure 3.3 Smart Contract Components

Figure 3.4 NIN Verification Sequence

Figure 3.5 PWA Caching Layers

Figure 4.1 Analytics Dashboard Overview

Figure 4.2 Multi‑Signature Proposal Workflow

LIST OF TABLES

Table 2.1 Comparative Analysis of E‑Voting Approaches

Table 3.1 Functional Requirements

Table 3.2 Non‑Functional Requirements

Table 4.1 Test Matrix

Table 4.2 Performance Benchmarks

Table 4.3 Security Controls Mapping

CHAPTER ONE: INTRODUCTION

1.1 Background of the Study

Elections are foundational to representative governance, yet the processes that enable them are often strained by logistical complexity, scalability challenges, security incidents, and disputes regarding legitimacy. Electronic voting (e‑voting) has emerged to streamline registration, balloting, counting, and auditing. However, centralized e‑voting systems are susceptible to single points of failure, insider attacks, and opaque tallying. Blockchain technology—an append‑only, tamper‑evident ledger maintained by a network—offers new primitives for integrity, transparency, and auditability. Smart contracts enforce rules deterministically, while decentralized storage (IPFS) and modern web technologies (PWAs) increase resilience and accessibility. This thesis advances BLOCKELECT, an enterprise‑grade e‑voting prototype designed to demonstrate how blockchain and adjacent technologies can improve election security and transparency under practical constraints.

1.2 Problem Statement

Traditional and centralized e‑voting systems exhibit weaknesses that undermine trust: tampering risks, opaque tallying, insider threat, coercion risks, and outages. In many contexts, identity verification and voter eligibility checks are inconsistent, and auditability is weak. The research problem addressed here is how to design and implement a practical e‑voting system that improves integrity, transparency, and resilience by leveraging blockchain, while enforcing robust identity verification and preserving voter privacy.

1.3 Aim and Objectives

Aim: To design and implement a secure, blockchain‑based e‑voting system (BLOCKELECT) that enhances integrity, transparency, and resilience, while integrating verifiable identity checks and privacy‑aware mechanisms.

Objectives:

• Implement Ethereum‑based smart contracts to record votes immutably and enforce rules.

• Integrate a mock NIN verification workflow to model government‑grade identity checks.

• Introduce multi‑signature governance to reduce single‑administrator risk.

• Provide a PWA client with offline capabilities and mobile‑first UX.

• Store ancillary election artifacts on IPFS for censorship resistance.

• Prototype a zero‑knowledge proof concept to preserve ballot secrecy while preventing double voting.

• Build a real‑time analytics dashboard to monitor election metrics.

• Evaluate the system’s functionality, performance, and security under realistic assumptions.

1.4 Research Questions

• How can blockchain primitives be applied to increase integrity and auditability of e‑voting?

• What governance mechanisms reduce administrative single‑point failures in election management?

• How can identity verification be modeled without centralizing sensitive personal data?

• What privacy techniques are feasible for ballot secrecy in a practical prototype?

• What are the performance and usability trade‑offs of a decentralized approach?

1.5 Scope of the Study

The system targets small to medium‑scale elections (e.g., departmental or institutional) conducted on a local blockchain test network for evaluation. It focuses on vote casting, tallying, identity verification flow design, transparency via analytics, and resilience via PWA and IPFS. It does not attempt to replace national electoral systems at production scale; rather, it provides a credible blueprint and an academically rigorous prototype.

1.6 Significance of the Study

This work contributes a well‑documented reference implementation that demonstrates end‑to‑end election workflows on blockchain with integrated identity verification, governance, and privacy concepts. It aids researchers, educators, and policymakers by clarifying trade‑offs and presenting actionable design patterns, and it provides students with a high‑quality academic and engineering artifact.

1.7 Methodology Overview

The research followed a design‑science methodology: (1) problem identification and objective definition; (2) design and development of artifacts (smart contracts, APIs, PWA client, analytics, IPFS integration); (3) demonstration via controlled experiments on Ganache; (4) evaluation through testing, performance measurements, and security analysis; and (5) communication via this thesis and repository documentation.

1.8 Project Deliverables

• Smart contract(s) for election management and voting.

• Node.js/Express backend with NIN verification mock API and health endpoints.

• Web client with PWA features, analytics dashboard, and admin interface.

• IPFS utilities for decentralized storage.

• ZK proof concept code for privacy exploration.

• Documentation: README, feature guides, and this thesis.

1.9 Thesis Organization

Chapter One introduces context and objectives. Chapter Two reviews relevant literature and technologies. Chapter Three describes methodology and system design. Chapter Four details implementation and results. Chapter Five concludes, discusses limitations, and outlines future work.

CHAPTER TWO: LITERATURE REVIEW

2.1 Electronic Voting: Concepts and Challenges

Electronic voting includes any computer‑assisted process in voter registration, ballot casting, transmission, counting, and auditing. Core desiderata include eligibility, uniqueness (no double voting), accuracy, verifiability, privacy, coercion resistance, availability, and transparency. Conventional e‑voting systems centralize data, thereby exposing systemic risks: database manipulation, denial‑of‑service against critical servers, and insider threat. Transparent audit trails and verifiability remain difficult when data is mutable and logs are not publicly reproducible.

2.2 Threat Model for Digital Elections

Adversaries include: external attackers (network intrusions, DDoS), insiders (privileged admin abuse), dishonest candidates or observers (attempted tampering), and coercers. Attacks range from ballot stuffing and deletion to voter impersonation, replay attacks, and result manipulation. Effective designs employ layered defenses: cryptographic integrity, redundancy, strict roles and authorizations, and post‑election audits.

2.3 Blockchain Technology: Foundations and Properties

Blockchains provide a distributed ledger with cryptographic linking of blocks, consensus‑based ordering, and append‑only semantics. Security properties include immutability against bounded adversaries, transparency via publicly replayable state transitions, and censorship resistance in decentralized networks. Public chains (e.g., Ethereum) enable smart contracts—programs executed by all validators—enforcing shared rules. For research and prototyping, local networks (Ganache) facilitate rapid iteration.

2.4 Smart Contracts and Decentralized Applications

Smart contracts capture election rules (candidate registration, time‑bounded voting, tallying) and enforce them deterministically. Their state (votes, candidates) is globally consistent across nodes. DApps integrate contracts with web UIs using wallets (e.g., MetaMask) for transaction signing. Formal verification and audits are ideal but out of scope; however, careful checks (e.g., bounds validation, modifiers for access control) mitigate many risks.

2.5 Consensus Protocols and Security Implications

Consensus determines block ordering and finality. Proof‑of‑Work (PoW) and Proof‑of‑Stake (PoS) offer distinct security models and performance profiles. For academic prototypes, we rely on local deterministic chains like Ganache. Production e‑voting would require robust networks with adequate decentralization and economic security.

2.6 Privacy‑Preserving Voting: Mix‑nets, Homomorphic Encryption, and ZK Proofs

Privacy is fundamental to free elections. Mix‑nets permute encrypted ballots to break linkability. Homomorphic encryption aggregates encrypted votes without revealing individual choices. Zero‑knowledge proofs (ZKPs) let a voter prove eligibility and uniqueness (via nullifiers) without revealing the vote. Practical, succinct ZK systems (e.g., Groth16) enable efficient verification but require careful circuit design and trusted setup considerations. BLOCKELECT includes a ZK concept module demonstrating how a proof interface could integrate with ballot flows.

2.7 Decentralized Storage: IPFS Principles and Use Cases

IPFS provides content‑addressed storage where files are identified by hashes (CIDs). This improves integrity and resilience for candidate assets and artifacts that need not live on‑chain. Gateways and local nodes provide access; pinning strategies ensure availability. Sensitive data should be encrypted before IPFS publication.

2.8 Progressive Web Apps for Civic Tech

PWAs provide installability, offline caching, background sync, and push notifications—useful during connectivity issues or peak loads on election day. Service workers cache critical assets, and background sync can queue ballot submissions until connectivity returns. This improves availability and user experience on mobile devices.

2.9 Multi‑Signature Governance Models

Multi‑signature (multi‑sig) wallets or governance modules require M‑of‑N approvals for critical operations, reducing unilateral admin power. In elections, candidate registration, date setting, and resets should require multiple officials’ consent, creating an auditable trail.

2.10 Related Work and Comparative Analysis

• Helios: A web‑based open‑audit voting system emphasizing verifiability.

• Estonia’s i‑Voting: Pioneering national‑scale remote voting with cryptographic audits and periodic public scrutiny.

• Blockchain Voting Pilots: Various pilots demonstrate feasibility but surface challenges in privacy, usability, and scalability.

BLOCKELECT’s contribution is a cohesive integration of blockchain, NIN‑style identity checks, multi‑sig governance, IPFS, analytics, and PWA capabilities in an academic prototype focused on end‑to‑end workflows and documentation.

2.11 Research Gap and Conceptual Framework

Gaps include: (a) insufficient governance controls in many prototypes; (b) weak integration between identity verification and on‑chain eligibility; (c) lack of offline‑first UX in civic contexts; and (d) limited practical privacy integrations. The conceptual framework links identity, governance, privacy, and transparency as mutually reinforcing pillars around a blockchain core.

2.12 Summary

The literature points to blockchain’s promise for integrity and auditability, and to ZK and related cryptography for privacy. Practical systems must integrate governance and identity while maintaining usability. These insights shape BLOCKELECT’s methodology and design.

CHAPTER THREE: METHODOLOGY AND SYSTEM DESIGN

3.1 Research Methodology

We adopt a design‑science research (DSR) approach: build an artifact to address a relevant problem and evaluate it rigorously. The SDLC used agile iterations: requirement gathering, prototyping, feedback, and refinement.

3.2 Requirements Analysis

Functional Requirements (selected):

• Register officials and candidates.

• Set election start/end dates; enforce time‑bounded voting.

• Verify voter identity via NIN‑style workflow and mark eligibility on‑chain.

• Cast exactly one vote per eligible voter; prevent duplicates.

• Provide admin multi‑signature approvals for critical operations.

• Expose analytics and health endpoints; export data.

• Store non‑sensitive artifacts (e.g., candidate photos) on IPFS.

Non‑Functional Requirements:

• Security: Access control, integrity, and least privilege.

• Privacy: Hide voter choices; conceptually enable ZK integration.

• Availability: PWA offline support, background sync.

• Performance: Acceptable latency under local network testing.

• Usability: Mobile‑first responsive UI.

• Auditability: On‑chain logs and proposal histories.

3.3 Use Case Modeling (Textual)

Actors: Voter, Official, Owner (deployer), System (smart contract), IPFS, MetaMask Wallet.

Main Use Cases:

• Voter identity verification (NIN) and registration.

• Vote casting during the active window.

• Official proposes candidate additions, date changes, or reset.

• Officials confirm proposals until reaching required threshold; system executes operation.

• Admin and observers view analytics; export results.

3.4 System Architecture (Textual)

• Client (PWA): HTML/CSS/JS with service worker, MetaMask integration, analytics views.

• Backend (Express): Serves static assets, health checks, mock NIN API, IPFS helpers, ZK interface.

• Blockchain (Ganache/Ethereum): VotingSys.sol smart contract for core logic.

• Storage (IPFS): Candidate photos/documents, optionally encrypted.

• Observability: Analytics JS module, event listeners for on‑chain events.

3.5 Data Models and Smart Contract Design

Core Structures:

• Candidate { name, party, votes }

• Officials: mapping(address => bool), officialsList[]

• Voter State: hasVoted[address], ninVerified[address], voterNIN[address]

• Election Window: startDate, endDate

• Multi‑Sig Proposals: id, type, data, proposer, confirmations, executed, confirmed[address]

Key Functions:

• registerCandidate (official‑only)

• setElectionDates (official‑only)

• verifyVoterNIN (official‑only)

• vote (requires ninVerified and active window)

• proposeAddCandidate / proposeSetElectionDates / proposeResetElection

• confirmProposal; internal execution upon threshold

• getters for candidates, proposals, officials, totals

Design Rationale: Separation of concerns, explicit access control via modifiers, and data minimization for privacy (store NIN hash or token, not raw PII).

3.6 Security Architecture and Threat Mitigation

• Integrity: On‑chain votes and events; immutable audit trail.

• Authorization: onlyOfficial modifier and owner‑gated operations; M‑of‑N multi‑sig for critical actions.

• Input Validation: bounds checks (candidate indices, election dates).

• Privacy: NIN hashes or pseudonymous tokens; ZK concept for ballot secrecy.

• Availability: PWA caching; background sync queuing for intermittent connectivity.

• Logging/Monitoring: Health endpoints; client analytics; event subscriptions.

3.7 Privacy Model and ZK‑Proof Conceptualization

We conceptualize a ZK pipeline where a voter obtains a proof attesting to eligibility and non‑reuse (nullifier) without revealing identity or vote choice. The smart contract verifies the proof and checks nullifier uniqueness before tallying an encrypted commitment or a blinded record. This thesis implements a ZK interface module and verification endpoint mock to validate feasibility and integration points.

3.8 NIN Verification Flow and API Design

• Input: NIN, date of birth, surname.

• Backend validates structure, rate‑limits requests, and compares against test dataset.

• Upon success, official accounts can mark ninVerified[voter]=true with a NIN hash stored on‑chain.

• Errors are descriptive and safe (no sensitive leakage). Health check advertises feature availability.

3.9 IPFS Integration Design

• Validate file size/type; upload via ipfs‑http‑client; persist CID.

• Retrieve via gateway or local node; implement fallback logic.

• Recommend optional encryption for sensitive content prior to upload.

3.10 PWA Architecture

• Service worker caches shell and critical routes.

• Offline page informs users and queues actions.

• Background sync retries queued vote submissions when connectivity resumes.

• Web App Manifest enables install prompts and app shortcuts.

3.11 Analytics Dashboard Architecture

• Charts: vote distribution, turnout over time, activity feed from blockchain events.

• Exports: CSV for results and logs.

• Responsive layout for mobile and desktop.

3.12 Development Tools and Environments

• Solidity ^0.8; Truffle; Ganache for local test chains.

• Node.js/Express backend; Web3.js for chain interactions.

• Chart.js for visualizations; ipfs‑http‑client for storage; MetaMask for signing.

• Version control via Git/GitHub.

3.13 Summary

The design balances integrity, governance, privacy awareness, usability, and deployability in an academic setting. Next, we cover implementation details and evaluation.

CHAPTER FOUR: IMPLEMENTATION AND RESULTS

4.1 Implementation Environment

• OS: Windows (development), Node.js v22+

• Blockchain: Ganache (deterministic accounts)

• Tooling: Truffle for compilation/migration; Browserify/Babel for frontend bundling.

• Wallet: MetaMask configured for Ganache network.

4.2 Smart Contract Implementation

VotingSys.sol implements candidates, voters, election window, NIN verification state, and a multi‑sig proposal mechanism. Access control uses modifiers (onlyOfficial, electionActive, onlyNINVerified). Resets clear candidates and voter state. Getter methods return arrays for compatibility with Web3 decoding.

4.3 Backend/API Implementation

Express server exposes:

• /api/health – status and feature flags

• /api/contract-data – ABI/address for client

• /api/verify-nin – mock verification endpoint (rate‑limited, input‑validated)

• /api/zk/verify – proof verification mock for concept demonstration

Security headers, CORS safeguards, request size limits, and logging are applied.

4.4 Frontend Implementation

The UI is composed of:

• index.html – voter interface (MetaMask connect, candidate list, vote casting)

• official.html – admin interface (proposal creation, candidate mgmt, date setting)

• analytics.html – real‑time dashboard with charts and activity feed

• manifest.json, service worker, and offline page for PWA behavior

JavaScript modules handle wallet detection, contract calls, UI feedback, and analytics rendering.

4.5 NIN Verification Module

The module provides forms for NIN, DOB, and surname; interacts with the backend; and, upon success, enables registration and marks on‑chain ninVerified via an official’s confirmation flow. Error messages are explicit (e.g., INVALID\_NIN\_FORMAT, NIN\_NOT\_FOUND) to assist users while avoiding data leakage.

4.6 IPFS Module

Users can upload candidate photos or documents. The module validates file constraints and stores results by CID. Gateway fallbacks maintain accessibility when local nodes are unavailable. CIDs can be displayed alongside candidate profiles.

4.7 PWA Features

Service worker caches the application shell and critical routes; background sync queues votes when offline and retries later. The manifest exposes name, icons, theme color, and shortcuts, making the app installable on mobile and desktop.

4.8 Analytics Dashboard

Using Chart.js, the dashboard displays vote distribution, turnout over time, and a live activity feed derived from blockchain events. CSV export allows post‑election analysis and archival.

4.9 Privacy (ZK) Prototype Module

The zk‑voting module demonstrates generation/verification interface calls (e.g., snarkjs patterns) and shows how a nullifier could prevent double voting without revealing voter identity. In production, fully designed circuits and audited verification contracts would be mandatory.

4.10 Testing Strategy and Test Cases

Testing included:

• Unit checks of contract functions (candidate registration, voting bounds, window checks).

• Integration tests across UI → backend → blockchain.

• NIN API cases: valid/invalid formats, suspended IDs, and non‑existent records.

• PWA offline tests: disable network, cast vote, re‑enable, confirm queue flush.

• IPFS: upload/retrieve, CID integrity.

A representative test matrix is provided in Table 4.1.

4.11 Performance Evaluation

Local tests on Ganache show sub‑second UI responses and transaction submission latencies dependent on block mining intervals. Chart rendering remains smooth under simulated loads (hundreds of events). Storage overhead for candidate assets is off‑loaded to IPFS. The system is suitable for departmental‑scale elections; large‑scale deployments would require further optimization.

4.12 Security Evaluation

• Integrity: Transactions and events provide a non‑repudiable log; tallies are reproducible by re‑executing chain state.

• Authorization: Multi‑sig proposals for critical actions; onlyOfficial checks enforced in contract.

• Input validation: Strict checks on indices and dates; backend rate limits and sanitization.

• Privacy: Avoids storing raw PII; explores ZK concepts for ballot secrecy.

• Availability: PWA cache, background sync; multiple IPFS gateways.

Threats like collusion among officials are mitigated—not eliminated—by raising signature thresholds (e.g., 3‑of‑5), strengthening governance.

4.13 Usability Evaluation

The UI adopts a mobile‑first design with clear flows: verify identity, register, cast vote, and view analytics. Error messages and health endpoints assist troubleshooting. Installable PWA reduces friction for repeat users.

4.14 Deployment Considerations

For pilots beyond local chains, deploy to testnets (e.g., Sepolia), integrate persistent IPFS pinning, and configure environment variables for endpoints. Security hardening (audits, formal verification) is necessary before high‑stakes use.

4.15 Comparative Results and Discussion

Compared to centralized e‑voting, BLOCKELECT improves transparency and auditability via on‑chain state and public event logs. Compared to prior blockchain prototypes, it emphasizes governance (multi‑sig), identity verification flow, PWA resilience, and analytics, offering a more comprehensive stack for academic evaluation.

4.16 Summary

Implementation validated feasibility of a secure, usable, and observable e‑voting stack using commodity tools. Next, we synthesize findings and outline future improvements.

CHAPTER FIVE: CONCLUSION AND FUTURE WORK

5.1 Summary of Findings

BLOCKELECT demonstrates that blockchain, coupled with identity verification flows, multi‑signature governance, decentralized storage, and privacy concepts, can form a practical foundation for trustworthy e‑voting in academic and pilot contexts. The system maintained integrity and auditability while improving availability and usability through PWA design.

5.2 Contributions

• An integrated, end‑to‑end blockchain e‑voting prototype with identity, governance, analytics, IPFS, and PWA features.

• A documented methodology and architecture suitable for replication and coursework.

• A privacy prototype showcasing ZK integration paths for ballot secrecy.

• A detailed evaluation highlighting strengths and trade‑offs.

5.3 Limitations

• Prototype‑scale evaluation (local chain) rather than production networks.

• Mock NIN backend—no live integration with government systems.

• ZK circuits conceptual rather than audited, production‑ready designs.

• Usability tests limited to small participant groups.

5.4 Recommendations

• Conduct larger user studies focusing on accessibility and coercion‑resistance.

• Integrate stronger privacy (audited ZK circuits; homomorphic tallying where applicable).

• Pursue formal verification and third‑party smart contract audits.

• Establish governance policies and incident response runbooks in collaboration with stakeholders.

5.5 Future Work

• Full ZK pipeline with audited verifier contracts, multi‑issuer credentials, and revocation lists.

• On‑chain identity frameworks (e.g., verifiable credentials) to replace mock NIN while preserving privacy.

• Advanced analytics (anomaly detection, turnout forecasts) and public transparency portals.

• Multi‑chain deployments (e.g., L2 rollups) to reduce fees and increase throughput.

• Robust IPFS pinning/cluster strategies and end‑to‑end encryption of sensitive artifacts.

5.6 Conclusion

This thesis presented BLOCKELECT, a secure e‑voting prototype integrating blockchain, governance, identity verification, decentralized storage, PWA resilience, and privacy concepts. The approach advances integrity, transparency, and availability while remaining practical for academic evaluation and pilot deployments. With targeted enhancements—especially in privacy and formal assurance—BLOCKELECT can evolve toward real‑world civic use cases.

REFERENCES (Selected)

[1] S. Nakamoto, “Bitcoin: A Peer‑to‑Peer Electronic Cash System,” 2008.

[2] G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” Yellow Paper, 2014.

[3] V. Buterin, “A Next‑Generation Smart Contract and Decentralized Application Platform,” Ethereum Whitepaper, 2014.

[4] J. Benet, “IPFS ‑ Content Addressed, Versioned, P2P File System,” 2014.

[5] E. Ben‑Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, “Zerocash: Decentralized Anonymous Payments from Bitcoin,” IEEE S&P, 2014.

[6] J. Groth, “On the Size of Pairing‑based Non‑interactive Zero‑Knowledge Arguments,” EUROCRYPT, 2016.

[7] B. Adida, “Helios: Web‑based Open‑Audit Voting,” USENIX Security, 2008.

[8] A. Springall et al., “Security Analysis of the Estonian Internet Voting System,” ACM CCS, 2014.

[9] OWASP, “Top 10 Web Application Security Risks,” 2021.

[10] Truffle Suite Documentation, https://trufflesuite.com/docs

[11] MetaMask Docs, https://docs.metamask.io

[12] Chart.js Docs, https://www.chartjs.org/docs/latest/

[13] ipfs‑http‑client Docs, https://www.npmjs.com/package/ipfs-http-client

[14] snarkjs Docs, https://github.com/iden3/snarkjs

Note: Please format the references according to your department’s style (APA/IEEE/Chicago) and add any institution‑specific policy documents (e.g., national election laws or NIMC/NIN specifications) as needed.

APPENDICES (Summaries)

Appendix A: System Setup Guide

• Prerequisites: Node.js, MetaMask, Ganache.

• Commands: npm install; npm run ganache; npm run migrate; npm start.

• Access: http://localhost:3000 (main), /analytics.html (analytics), /official.html (admin).

Appendix B: Test Data and Cases

• Valid NINs: as provided in README.

• Invalid/Suspended cases to test error handling.

• Offline test steps for PWA queueing.

Appendix C: Screenshots (Reference)

• Voting interface, analytics dashboard, MetaMask prompts, admin proposal workflow.

Appendix D: Risk Register (Excerpt)

• Insider tampering → multi‑sig approvals, immutable logs.

• Connectivity outage → PWA caching and background sync.

• Data censorship → IPFS storage and gateway fallbacks.

Appendix E: Ethical and Legal Considerations

• Protect voter privacy; minimize PII on chain.

• Obtain consent for pilots; adhere to institutional review requirements.

— End of Thesis Report —