National University of Computer and Emerging Sciences
Islamabad Campus

Project Final Report

CY4001

Secure Software Design

Submitted by: Javeriah Faheem, Ayesha Areej, Sabreena Azhar

Roll number: 22I-7421, 22I-1711, 22I-1751

Date: 29th November 2025

Table of Contents

1. Project Context & Overview

1.1 Problem Statement:

Modern voting systems face serious security challenges. Paper-based systems are slow, prone to human error, and susceptible to manipulation. Existing online voting systems often fail to provide confidentiality, integrity, and auditability simultaneously. From a secure software design perspective, the challenges include:

- Preserving the confidentiality of voter choices.
- Ensuring integrity so that votes cannot be altered or duplicated.
- Maintaining availability throughout the election period.
- Providing audit trails for transparency and accountability.

1.2 Objectives & Goals

- Apply secure software design principles to build a privacy-preserving voting system.
- Implement end-to-end encryption for voting secrecy.
- Enforce voter authentication to prevent unauthorized or duplicate votes.
- Maintain integrity using digital signatures and secure logging.
- Introduce audit trails to ensure election results can be independently verified.
- Deliver a working prototype, documentation, and security analysis report.

1.3 Proposed Solution

System overview:

The Secure E-Voting System is a lightweight GUI-based voting application designed to preserve confidentiality, ensure vote integrity, prevent double-voting, and maintain system availability throughout an election. The system integrates secure software design principles such as encryption, authentication, auditability, tamper resistance, and access control. The system allows voters to register using their CNIC, receive a unique token, authenticate, cast an encrypted vote, and receive confirmation. Administrators can add candidates and view aggregated results but cannot access individual votes. The system generates audit logs to support transparency and post-election verification.

Secure Software Principles Integrated:

- Confidentiality: End-to-end encryption of votes.
- Integrity: Digital signatures and tamper detection.
- Availability: Lightweight architecture with redundancy and error handling.
- Accountability: Secure audit trails and logging for verification.

Threats & Countermeasures:

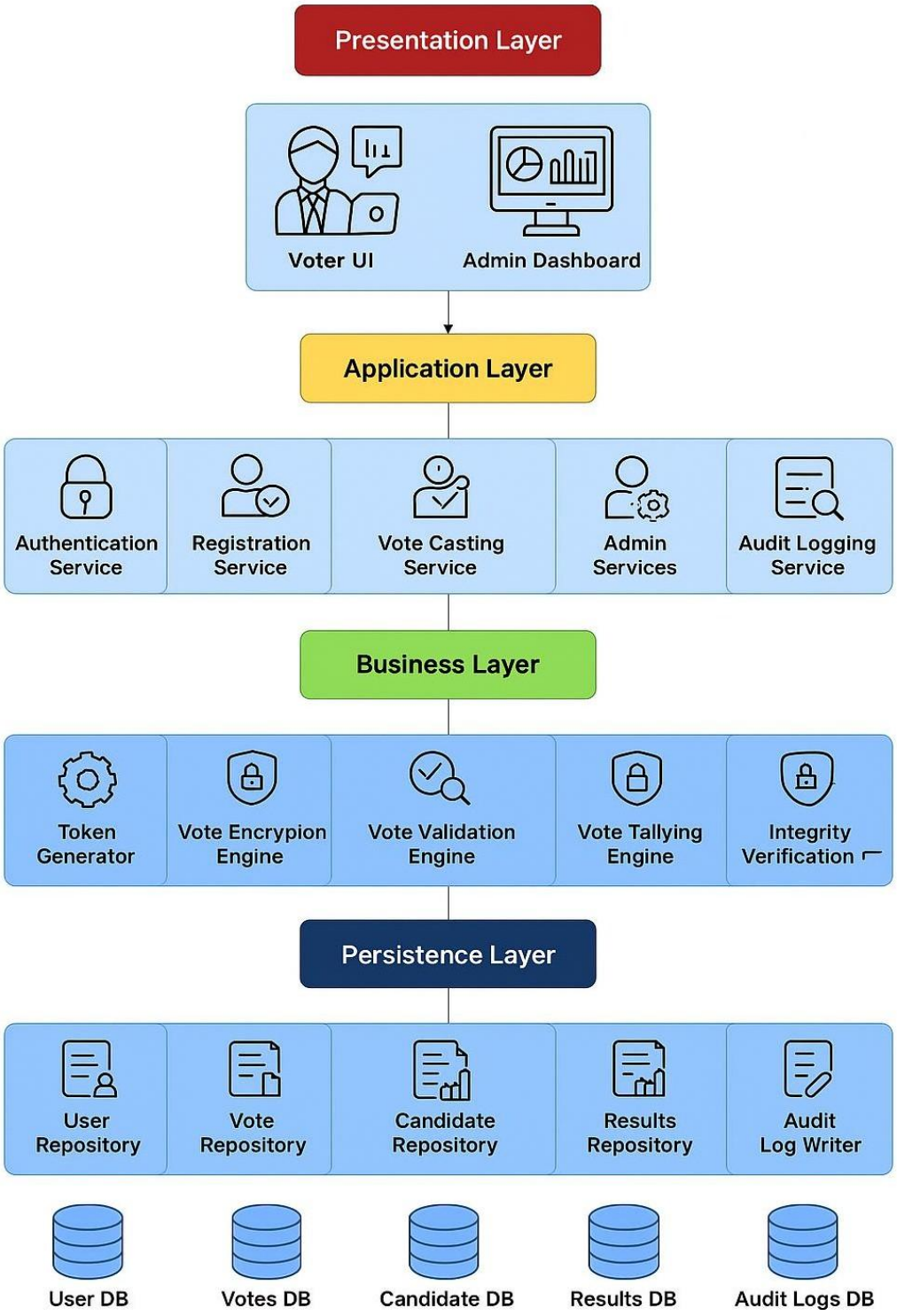
Threat	Risk Level	Mitigation
Double voting / replay attack	High	Unique voter tokens, one-time authentication
Vote tampering	High	Digital signatures, hashing, audit logs
Unauthorized access	Medium	Strong voter authentication, input validation
Denial of Service (DoS)	Medium	Lightweight design, redundancy
Privacy leaks	High	Separation of identity from vote, blind-signing mechanisms

1.4 Methodology

The project follows the Secure Software Development Lifecycle (S-SDLC):

- Requirements Analysis: Identify functional needs (voter registration, casting, tallying, audit). Define security goals using CIA triad and threat modeling (STRIDE).
- Design: Use security frameworks (NIST SSDF, principle of least privilege). Design modules for registration, encryption, authentication, and auditing.
- Implementation: Apply secure coding practices: input validation, no hard-coded secrets, exception handling, cryptographic best practices.
- Testing: Conduct unit tests, integration tests, and security testing (penetration tests, boundary testing, replay attack simulations).

1.5 Architecture



2. Threat Modeling & Risk Analysis

Using the following frameworks for threat identification and risk analysis:

2.1 PASTA

Stage 1: Business Objectives & Security Goals

The primary objectives and security goals of the system are:

→ Preserve privacy by ensuring confidentiality of voter identity and vote content.
→ Prevent double voting through robust authentication and token mechanisms.
→ Maintain integrity by ensuring votes cannot be altered, forged, or deleted.
→ Maintain system availability during election hours.
→ Prevent administrators from accessing individual encrypted votes.
→ Provide complete audit trails for post-election verification.
→ Prevent unauthorized access to sensitive components.
→ Ensure scalability and reliability during peak usage.

Stage 2: Technical Scope

This stage defines the system's technical boundaries, components, and flows.

System Components:

- Registration Interface
- Login System
- Token Generator
- Voting Interface
- Voting Backend
- Vote Counting Engine
- Admin Panel
- User DB
- Votes DB
- Audit/Logs DB

External Entities:

- Voters
- Administrators
- Potential adversaries (internal and external)

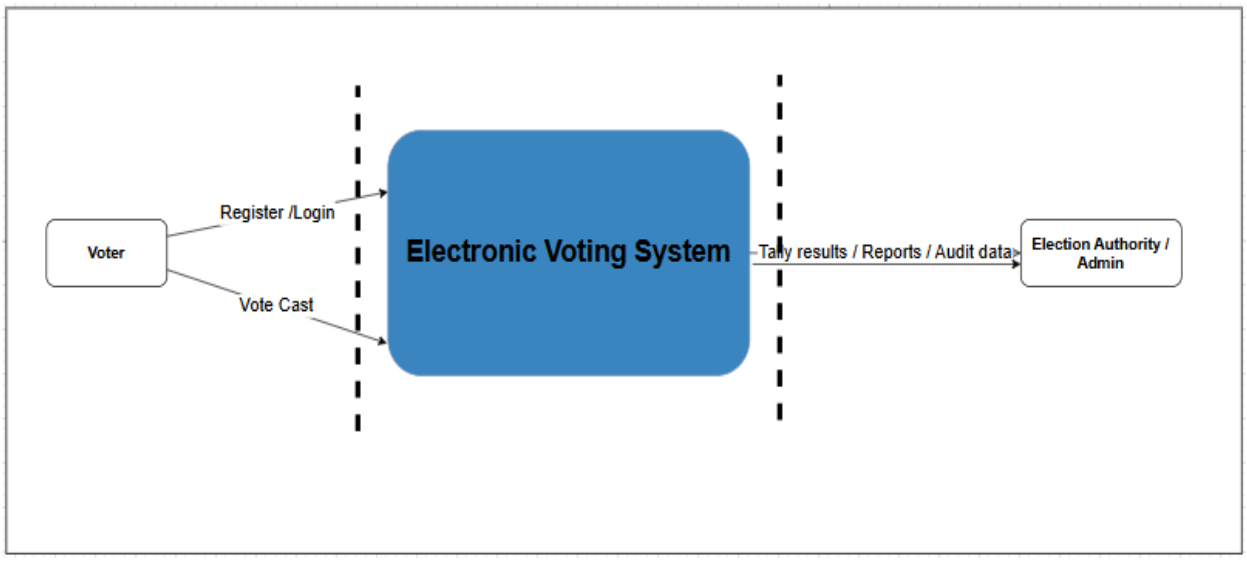
Data Flows:

- Registration → User DB
- Authentication → Token Verification
- Vote submission → Encryption → Ballot DB
- Audit logs → Audit DB
- Tallying → Results DB → Admin Panel

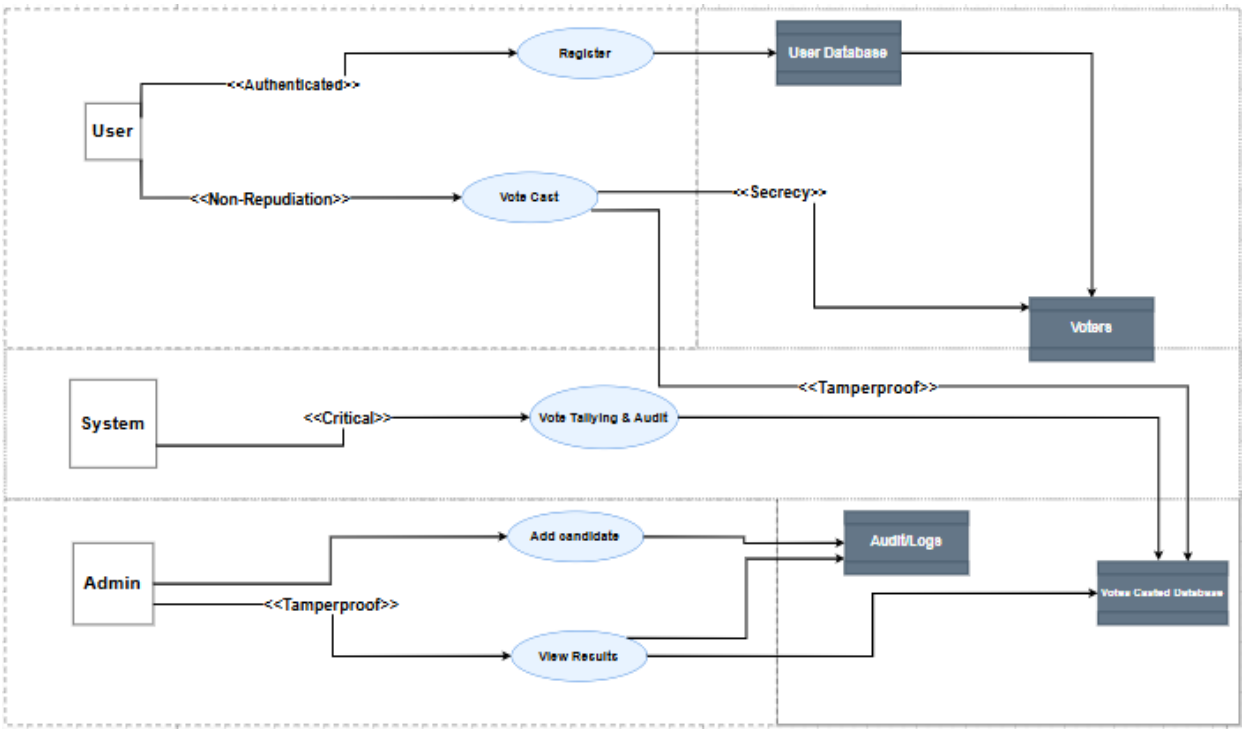
Sensitive Assets:

- CNICs
- Credentials
- Voter tokens
- Encrypted ballots
- Audit logs
- Candidate list
- Election results

DFD LEVEL 0:



DFD LEVEL 1:



Stage 3: Application Decomposition

Subsystems:

- Registration subsystem
- Token issuance & casting subsystem
- Vote tallying & audit subsystem
- Admin management subsystem

Entry Points:

- Registration form
- Login form
- Vote submission endpoint
- Admin login
- Admin candidate management

Trust Boundaries:

- User-voter interface → Backend
- Admin panel → Backend
- Backend → Databases
- Audit logs → Read-only access

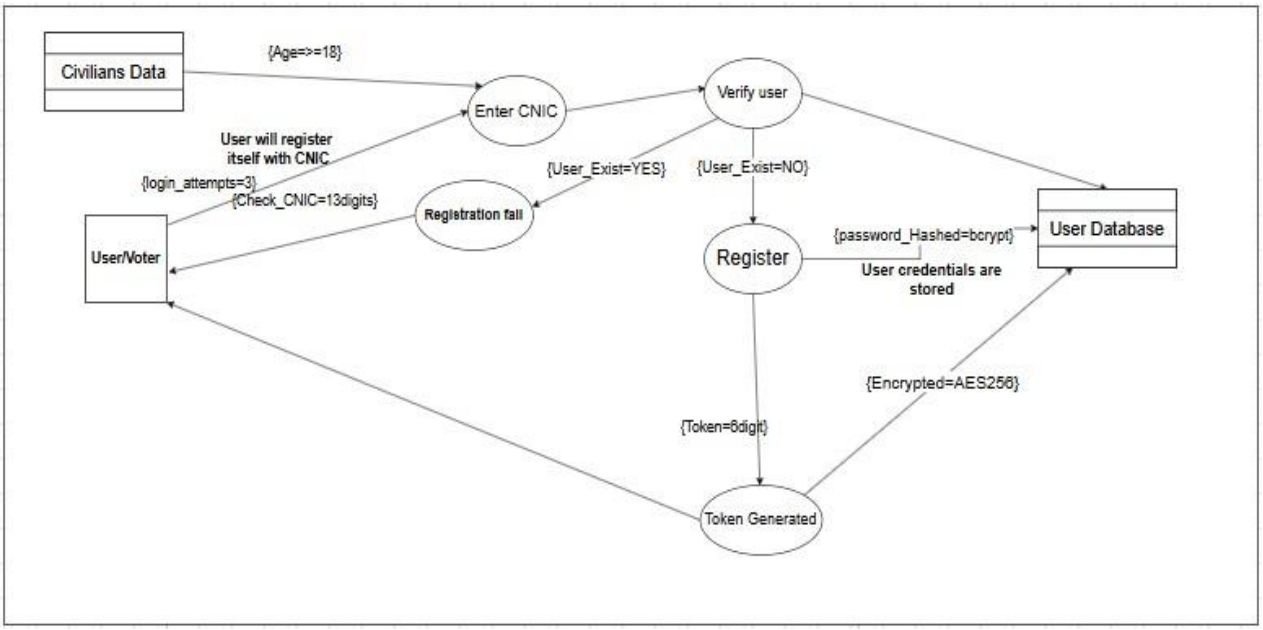
Assets & Controls:

Our assets require controls as follows:

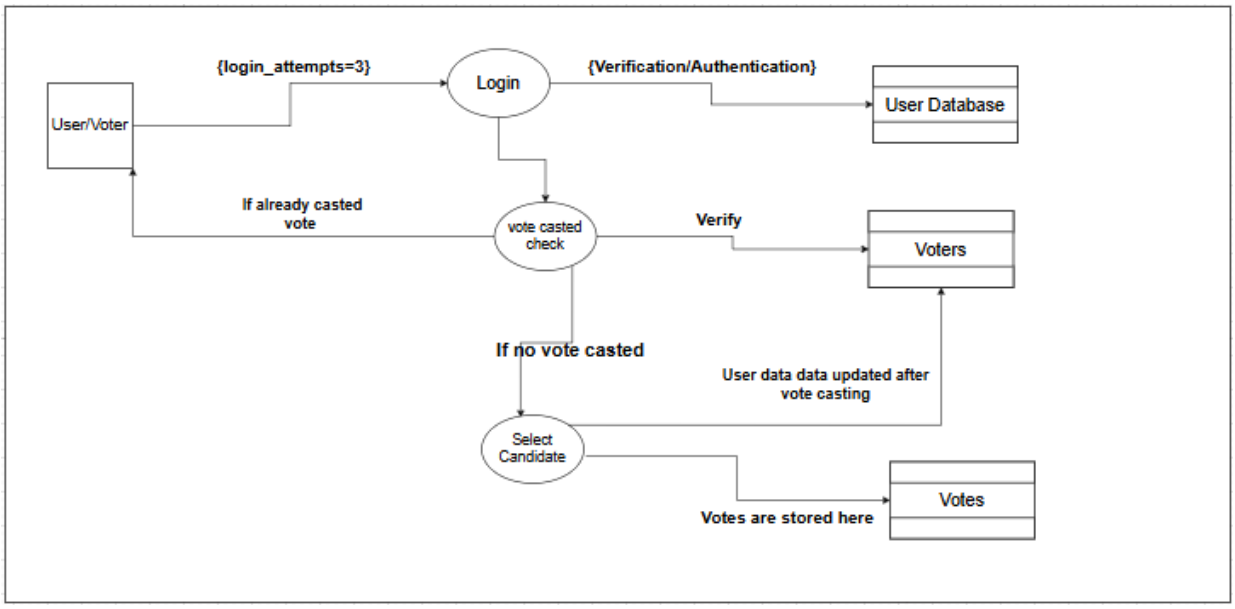
- Tokens → Anti-replay, one-time use
- Votes → Encryption, tamper resistance
- Logs → Integrity protection
- Databases → Access control
- Credentials → Encryption + hashing

DFD LEVEL 2

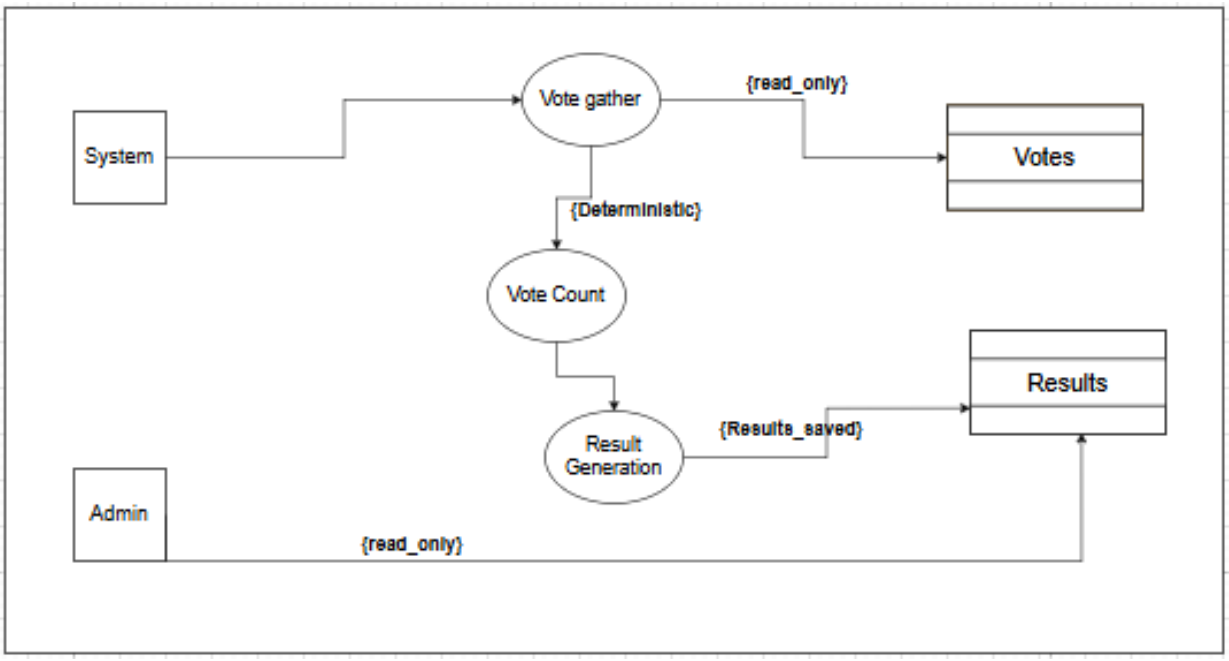
Subsystem 1:



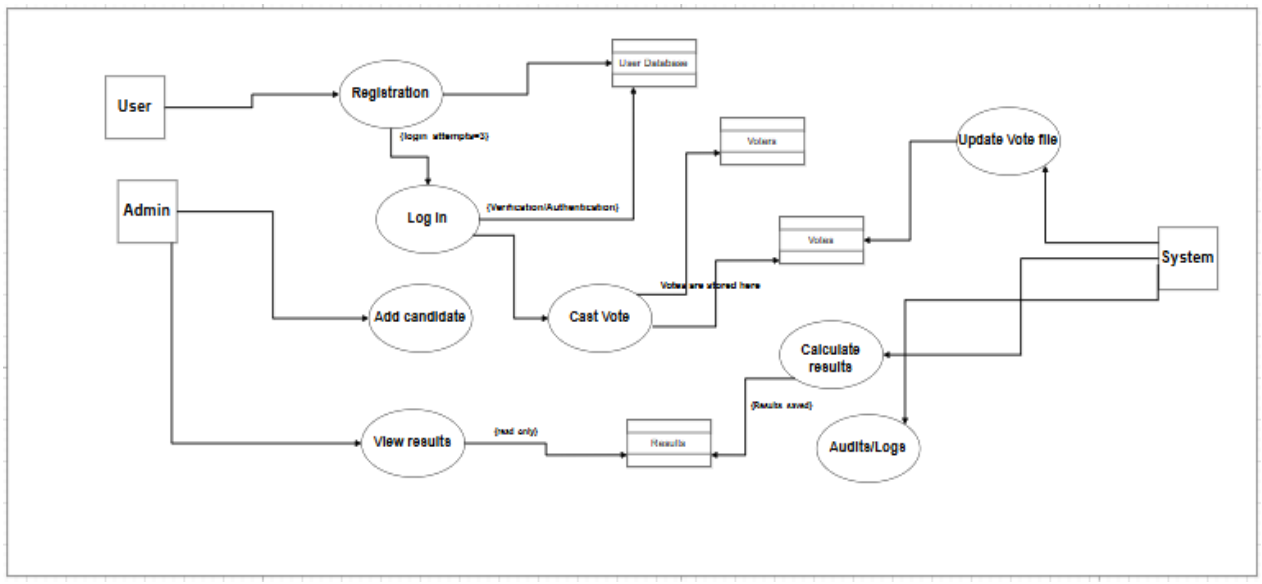
Subsystem 2:



Subsystem 3:



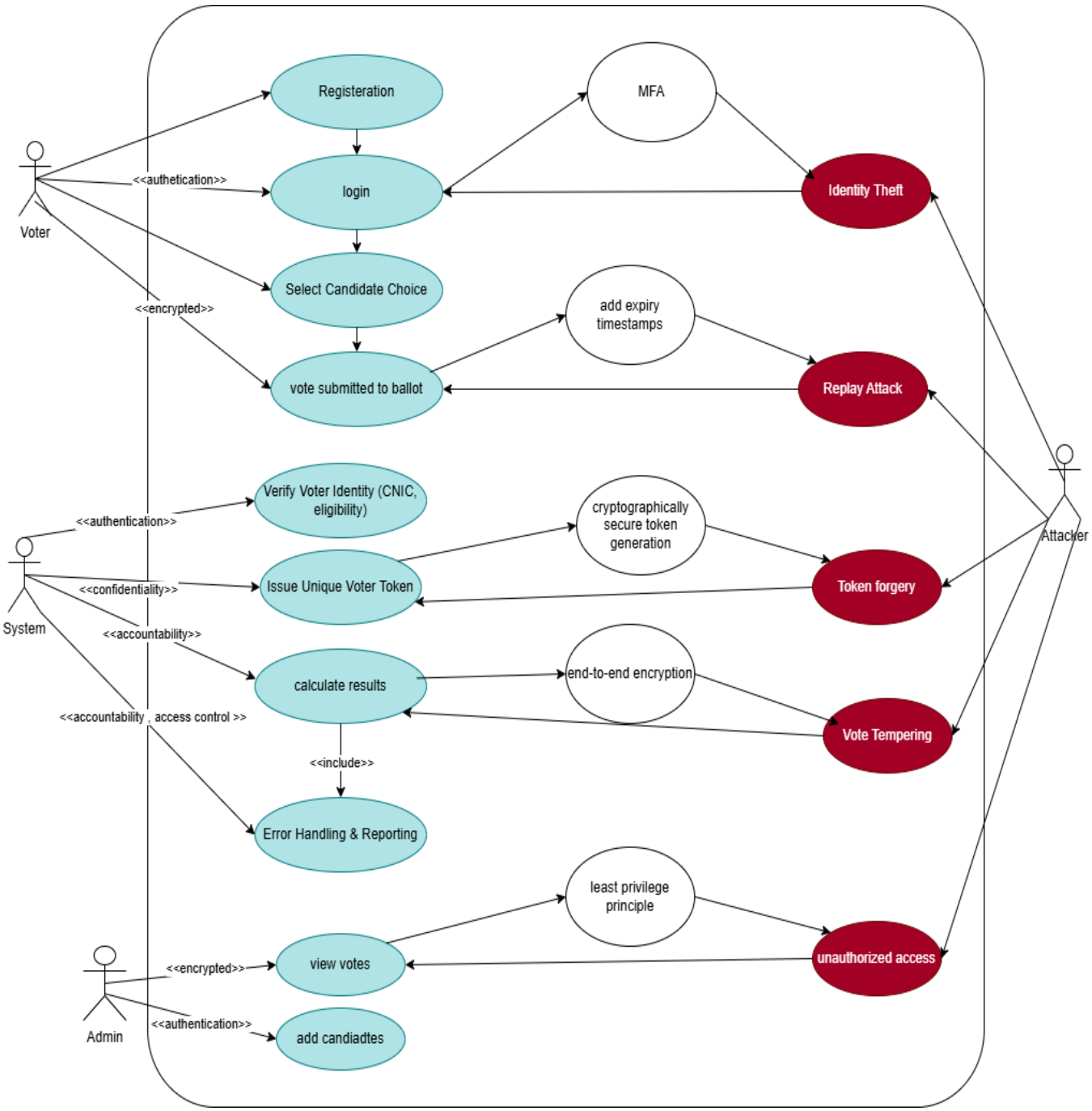
Subsystem 4:



Stage 4: Threat Analysis (Attackers & Attack Surfaces)

Attacker Profile	**Description**	**Relevant Attack Surfaces**
Malicious Voter	A legitimate voter attempting to cheat or manipulate the system (double voting, impersonation).	Registration input fields, Login form, Voting submission, Token generation/validation, GUI event handlers
External Attacker	Someone outside the system attempting to break in, steal data, or manipulate votes.	Registration input fields, Login form, Voting submission, Communication channels, Databases, Token validation
Compromised Admin	Insider attacker with elevated privileges abusing access.	Admin panel, Databases, Logs and system files, Vote counting process
Network Attacker	Attacker on the same network attempting interception or manipulation.	Communication channels, Token generation/validation, Login form, Voting submission

Misuse case Diagram



Stage 5: Vulnerability & Weakness Analysis

Key weaknesses the system may face:

Module / Component	**Identified Vulnerabilities & Weaknesses**
Registration Module	• Weak authentication mechanisms • Lack of input validation (SQLi, malformed data)• Token theft/reuse risks
Login Module	• Weak authentication • Brute force susceptibility • Replay attacks on login requests
Voter Client (UI)	• Unencrypted traffic exposing credentials • Weak client-side validation • Susceptible to phishing or spoofed UI
Vote Casting Module	• Replay attacks (duplicate vote submissions) • Token reuse or predictable token generation • Missing integrity verification of vote payload
Communication Channels	• Unencrypted traffic enabling MITM • Sniffing & packet replay• Session hijacking risks
Databases (User DB / Votes DB)	• Database manipulation (tampering user/vote data) • Lack of strict access controls • Injection vulnerabilities
Admin Panel	• Privilege escalation risks • Insider threat - unauthorized vote viewing/modification • Weak audit logging
Vote Counting / Tallying Engine	• Tampering with vote totals • Unauthorized access to tally logic

Stage 6: Attack Modeling & Simulation

Below are some of the modeled attack scenarios for the e-Voting System. Each scenario describes the attacker type, method of attack, targeted component, the sequence of steps, and the resulting impact.

Scenario S1: Fraudulent Registration using Stolen CNIC

Field	Details
Attack Path (Steps)	1. Attacker obtains victim's CNIC (photo/WhatsApp scan/social engineering). 2. Opens registration form and enters stolen CNIC. 3. System issues token to attacker, believing they are legitimate. 4. Attacker logs in using fake identity and prepares to vote.
Tools Used	Social engineering, leaked CNIC databases, fake scanner apps
Targeted Asset	Voter identity, registration module
Impact	Unauthorized voting, victim loses right to vote, election legitimacy weakened
Controls Required	MFA during registration, CNIC verification with OTP, rate limiting, registration logs, anomaly detection

Scenario S2: Replay Attack on Vote Submission

Field	Details
Attack Path (Steps)	1. Attacker sniffs network traffic between voter and system. 2. Copies the encrypted vote submission request packet. 3. Replays the same packet multiple times. 4. System receives multiple identical requests and may store multiple votes if no replay protection.
Tools Used	Wireshark, TCP replay tools, packet sniffers
Targeted Asset	Vote submission endpoint, token freshness check

Impact	Duplicate votes, inflated numbers, compromise of election integrity
Controls Required	One-time-use tokens, timestamping, nonce verification, HTTPS/TLS, replay-detection middleware

Scenario S3: SQL Injection on Registration Form

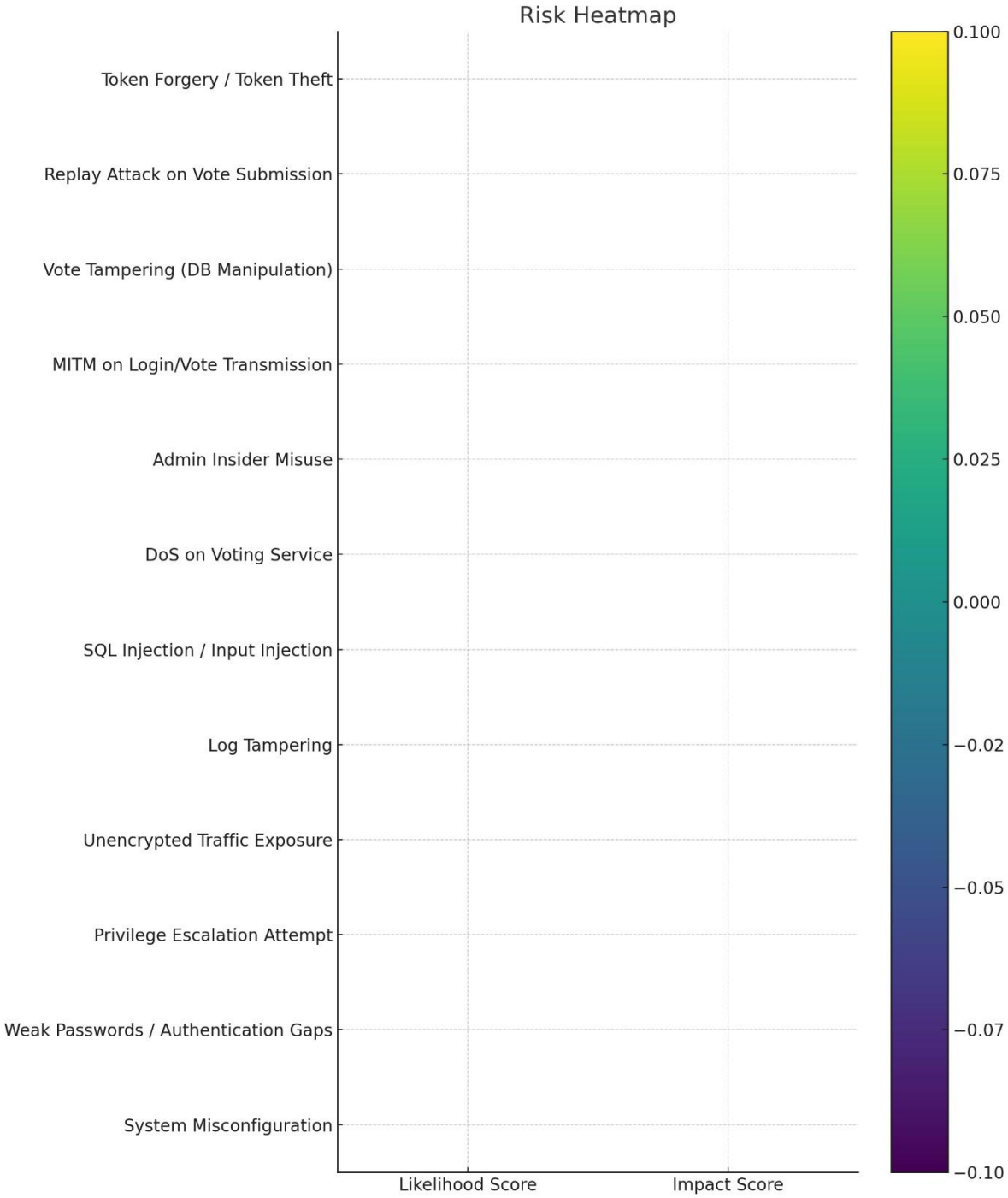
Field	Details
Attack Path (Steps)	1. Attacker types SQL payload into CNIC or name field. 2. Backend executes query without sanitization. 3. Attacker modifies user records, deletes data, or bypasses checks. 4. Gains elevated access or corrupts registration DB.
Tools Used	SQLMap, Burp Suite, manual payload crafting
Targeted Asset	User DB, registration subsystem
Impact	Data corruption, fake users, bypassing identity checks, system takeover
Controls Required	Input validation, prepared statements, WAF, proper error handling

Scenario S4: DoS Attack Before Polls Open

Field	Details
Attack Path (Steps)	1. Attacker finds public IP/port of voting system. 2. Uses botnets to send thousands of requests per second. 3. System gets overloaded and becomes unavailable. 4. Legitimate voters cannot access system.
Tools Used	LOIC/HOIC, botnets, automated HTTP request generators
Targeted Asset	System availability, front-end server
Impact	Voters unable to cast vote, election delays, denial of democratic process
Controls Required	Rate limiting, firewalls, load balancers, cloud scaling, IP throttling

Stage 7: Risk Analysis & Management

Threat	Likelihood	Impact	Targeted Asset	Existing Controls	Required / Recommended Controls	Overall Risk
Token Forgery / Token Theft	High	High	Voter identity, voting rights	Unique UUID token	One-time-use tokens, token expiration, server-side tracking, MFA	Critical
Vote Tampering (DB Manipulation)	Medium	Very High	Encrypted ballots, tally results	Basic DB access control	Append-only storage, digital signatures, integrity hashes, RBAC	Critical
MITM on Login/Vote Transmission	Medium	High	Credentials, vote content	None (in prototype)	Enforce HTTPS/TLS, certificate pinning	High
Admin Insider Misuse	Medium	High	Candidate list, results, logs	Admin login only	RBAC separation, immutable logs, admin activity auditing	High
DoS on Voting Service	High	Medium	Frontend service availability	None	Rate limiting, request throttling, CAPTCHA, load balancing	High
SQL Injection / Input Injection	Medium	High	User DB, Votes DB	Basic validation	Strict input validation,	High


Risk Heatmap
Token Forgery / Token Theft
Replay Attack on Vote Submission
Vote Tampering (DB Manipulation)
MITM on Login/Vote Transmission
Admin Insider Misuse
DoS on Voting Service
SQL Injection / Input Injection
Log Tampering
Unencrypted Traffic Exposure
Privilege Escalation Attempt
Weak Passwords / Authentication Gaps
System Misconfiguration
Likelihood Score
Impact Score
0.100
0.075
0.050
0.025
0.000
−0.02
−0.05
−0.07
−0.10

2.2 STRIDE

Below are the STRIDE tables for different modules:

1. Registration Module

STRIDE Category	Threat Identified	Possible Impact
Spoofing	Fake CNIC / impersonation during registration	Unauthorized voter accounts, fraudulent voting
Tampering	Modification of registration request or stored voter data	Corrupted records, registration denial
Repudiation	Voter denies ever registering	Weak accountability, audit inconsistencies
Information Disclosure	Leakage of CNIC or personal data	Privacy breach, identity exposure
Denial of Service	Flooding registration endpoint	Legitimate users unable to register
Elevation of Privilege	Attacker registers with admin-like privileges	System compromise

2. Login / Authentication Module

STRIDE Category	Threat Identified	Possible Impact
Spoofing	Stolen credentials / forged authentication token	Unauthorized access
Tampering	Manipulating authentication requests	Incorrect authentication outcomes
Repudiation	User denies login activity	Loss of traceability
Information Disclosure	Password/session leakage	Account takeover
Denial of Service	Brute-force login attempts	Lockout of legitimate voters

3. Vote Casting Module

STRIDE Category	Threat Identified	Possible Impact
Spoofing	Fake or reused voter token	Illegitimate votes
Tampering	Alteration of encrypted vote in transit	Manipulated election results
Repudiation	Voter denies submitting a vote	Disputed outcomes
Information Disclosure	Revealing vote choice	Loss of ballot secrecy
Denial of Service	Blocking the vote submission endpoint	Voters unable to cast votes
Elevation of Privilege	Voter attempts modifying system logic	Election integrity compromised

4. User Database

STRIDE Category	Threat Identified	Possible Impact
Spoofing	Attacker impersonates admin to modify DB	Unauthorized edits
Tampering	Changing voter status (e.g., “already voted”)	Double voting or voter suppression
Repudiation	No logged actions for DB updates	No traceability
Information Disclosure	Leak of voter details	Privacy violations
Denial of Service	DB locked/deleted	Voters unable to authenticate
Elevation of Privilege	Unauthorized access to DB roles	Full system compromise

5. Votes Database

STRIDE Category	Threat Identified	Possible Impact
Spoofing	Insertion of fake votes	Inflated or false results
Tampering	Modifying stored encrypted ballots	Rigged election
Repudiation	No record of who inserted ballots	Unverifiable vote history
Information Disclosure	Exposure of encrypted ballots	Potential privacy attack vectors
Denial of Service	Write-block on votes DB	Votes not stored
Elevation of Privilege	Unauthorized write/delete access	Election integrity collapse

2.3 DREAD

Applied DREAD to 5 major threads identified via STRIDE:

Threat	D	R	E	A	D	Avg	Priority
Token Forgery	9	8	7	9	8	8.2	High
Vote Tampering (DB)	10	7	6	9	7	7.8	High
MITM on Vote Submission	8	9	8	8	9	8.4	High
DoS on Voting Service	7	6	8	10	7	7.6	Medium
Admin Privilege Misuse	6	4	4	7	6	5.4	Medium

2.4 Mitigation Recommendations

Reflection

Working through PASTA forced a full end-to-end breakdown of the system. It became obvious where vulnerabilities could realistically occur rather than relying on guesswork. STRIDE helped classify those threats systematically, ensuring no category was ignored. DREAD then forced prioritization because in the real world not every threat matters equally, and security resources are always limited.

The combination of these three frameworks produced a threat model that is structured and complete. It highlighted weaknesses we had not initially noticed, especially around token reuse, insider threats from admins, and the potential for MITM attacks if transport security is not implemented.

Mitigation Strategies for Top Threats

1. MITM Attacks on Login / Blind-Signing / Vote Submission

Threat:
Attackers on the same network could intercept CNICs, credentials, OTPs, or even the token/signature pair.

Mitigation Strategy:
The system needs to force HTTPS end-to-end. TLS stops passive sniffing and active tampering. In production, HSTS + strict CSP headers are also required so browsers refuse downgrades or mixed content.

Feasibility:
High.
HTTPS with certificates is standard and trivial to enable. Talisman middleware can enforce HSTS automatically. The main challenge is deployment-side configuration, not code.

2. Brute Force & Credential Stuffing on Admin/User Login

Threat:
Attackers repeatedly guess passwords or use leaked email/password combos from other breaches.

Mitigation Strategy:
Rate limiting, IP-based throttling, and lockout after several failures. MFA for users ensures

stolen passwords won't be enough. Admin accounts get the strongest throttling + monitored logs.

Feasibility:
Very High.
All mitigations are easy to implement: limiter, an OTP module, and DB-backed login attempts. CPU cost is negligible.

3. SQL Injection on Registration, Login, Blind Sign, and Vote Submission

Threat:
Attackers inject SQL through form fields to dump DB content or modify tables.

Mitigation Strategy:
Use parameterized queries everywhere (sqlite3 placeholders), validate CNIC/token formats with regex, and reject malformed or unexpected inputs early. Never concatenate user input into SQL.

Feasibility:
High.
The framework already supports safe parameter binding and input validators are cheap to extend

4. CSRF on /submit_vote & /blind_sign

Threat:
If a logged-in voter visits a malicious site, that site could auto-submit a forged vote or trigger blind-signing.

Mitigation Strategy:
Origin checking, SameSite cookies = Lax/Strict, and explicit CSRF token adoption if needed. Vote submission is particularly sensitive — origin validation alone blocks most opportunistic CSRF.

Feasibility:
Medium-High.
Origin/referrer checks are extremely cheap. Full CSRF-token integration would require modifying templates, but doable.

5. Token Theft & Double Voting

Threat:
Attacker steals the blinded token/signature pair from storage, browser, or network and reuses it for multiple votes.

Mitigation Strategy:
Tokens must be used only recorded as SHA-256 hashes in a used_tokens table. Even if stolen, the attacker cannot reuse them because the system rejects duplicates.

Feasibility:
Very High.
Hashing tokens + checking a table is trivial and already part of the design.

6. DoS / Bot Flooding Login or Vote Endpoints

Threat:
Bots spam endpoints to prevent legitimate voters from accessing the system or to drain OTP resources.

Mitigation Strategy:
Rate limiting on all sensitive routes, server-side throttling, and pagination/log capping for admin log viewer. Optionally Cloudflare/NGINX rate limits in deployment.

Feasibility:
Medium-High.
App-level rate limits are easy, but infrastructure-level DoS mitigation requires hosting support.

7. Weak Password Storage / Predictable OTPs

Threat:
Attackers crack user passwords or guess OTPs.

Mitigation Strategy:
Use bcrypt hashing with salt, strict password policy, short OTP validity windows, and hashing OTPs in DB rather than storing plaintext.

Feasibility:
High.
Password hashing libraries already exist and OTP logic is simple.

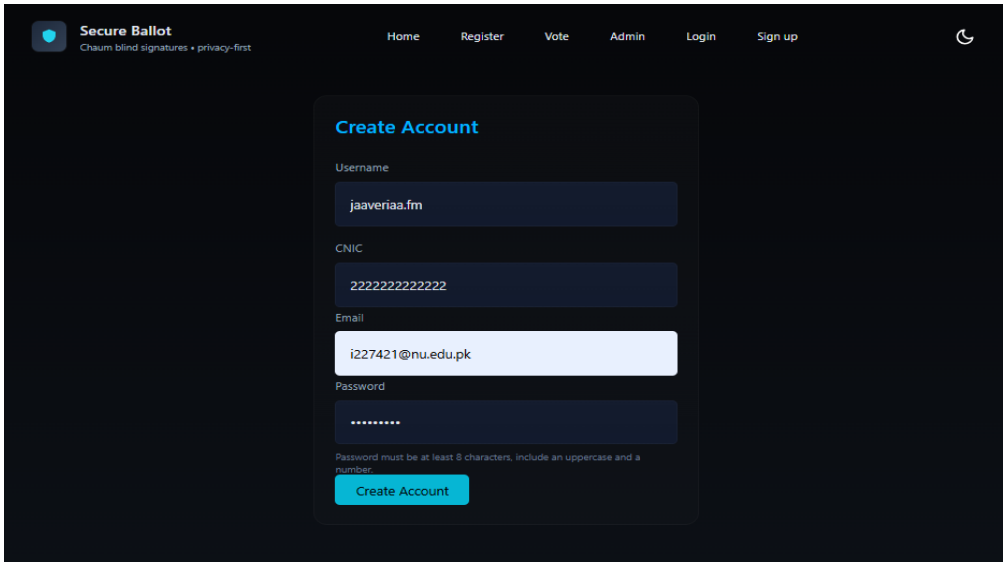
3. System UI & Testing

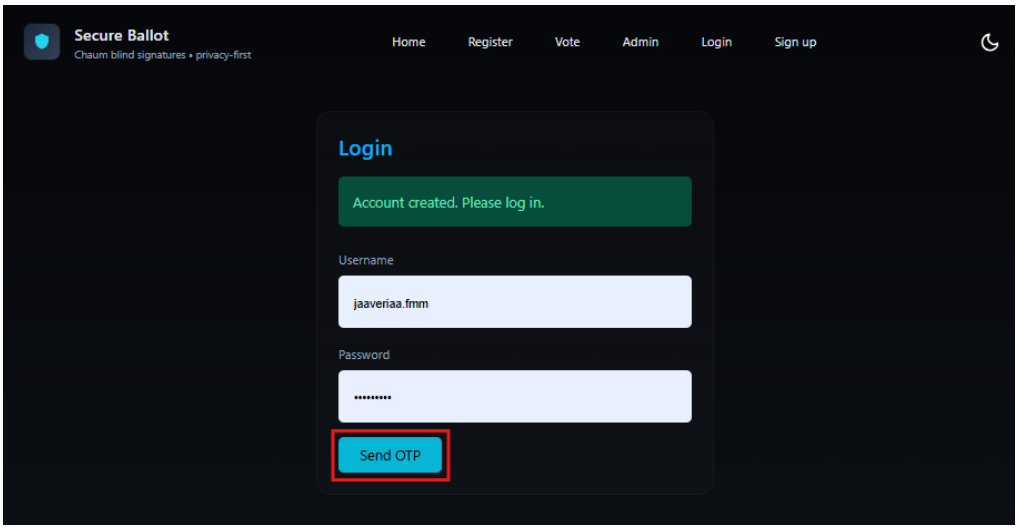
Following is the overall system testing that we conducting after the implementation of secure coding practices.

3.1 Functional Testing:

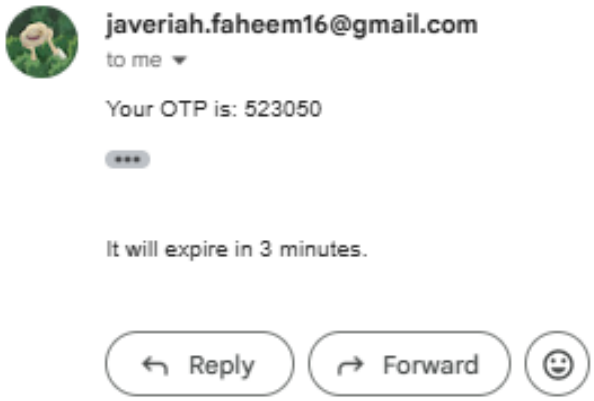
1. User Registration & Login

Normal case ✅

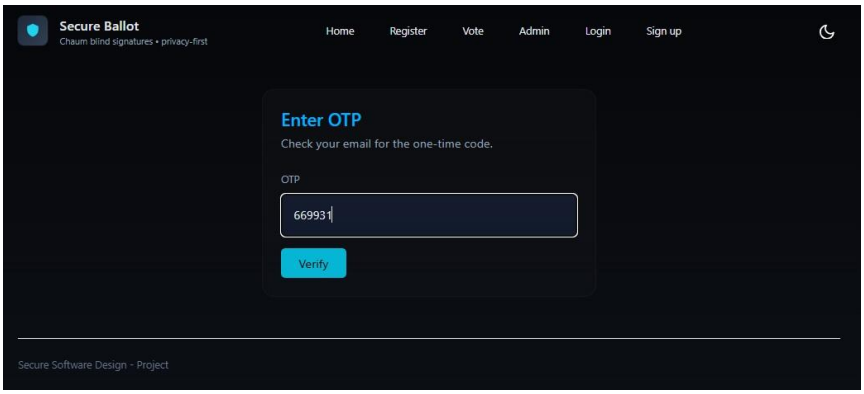




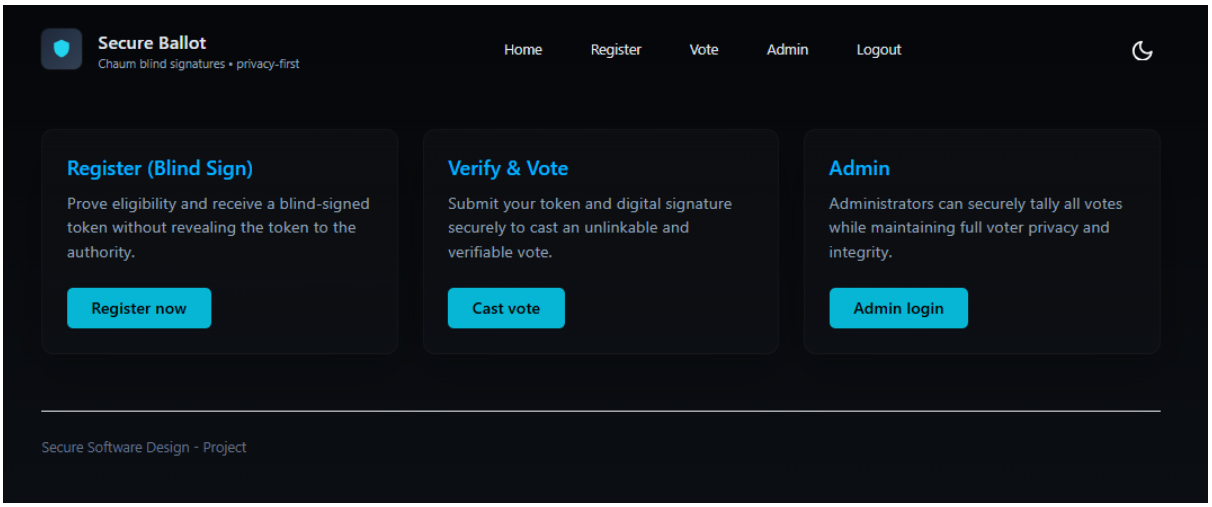
- User receives the OTP on the email that they registered with:



- User enters OTP

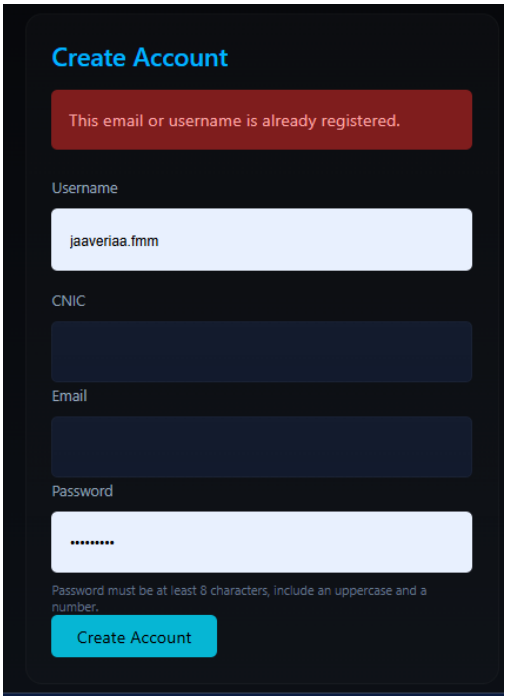


- Logged in successfully 😊

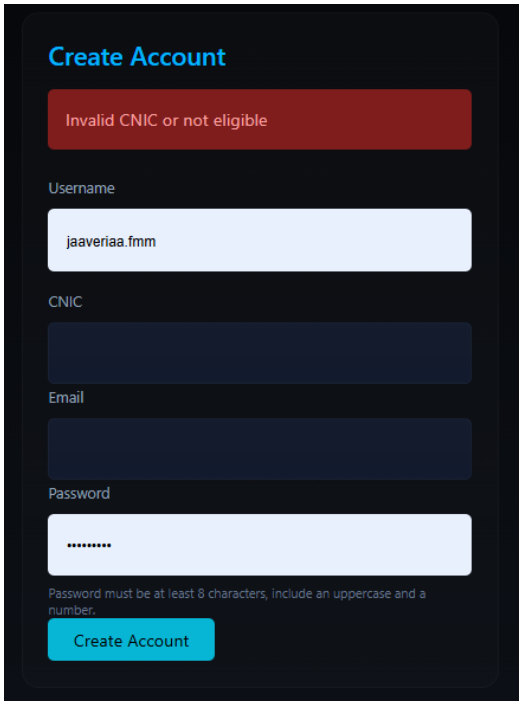


Edge cases

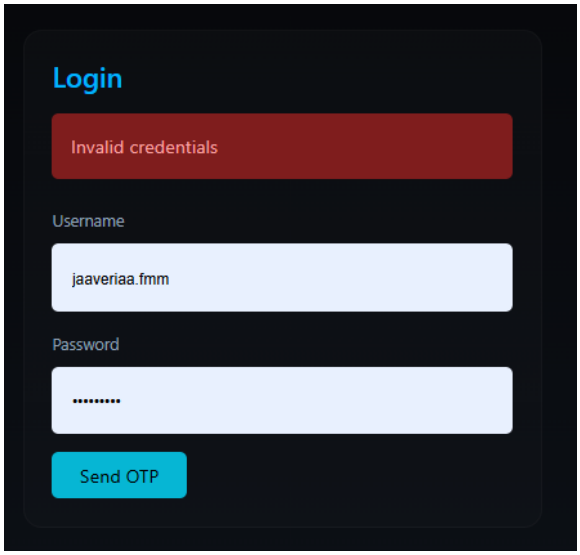
- Using pre-registered email/username:



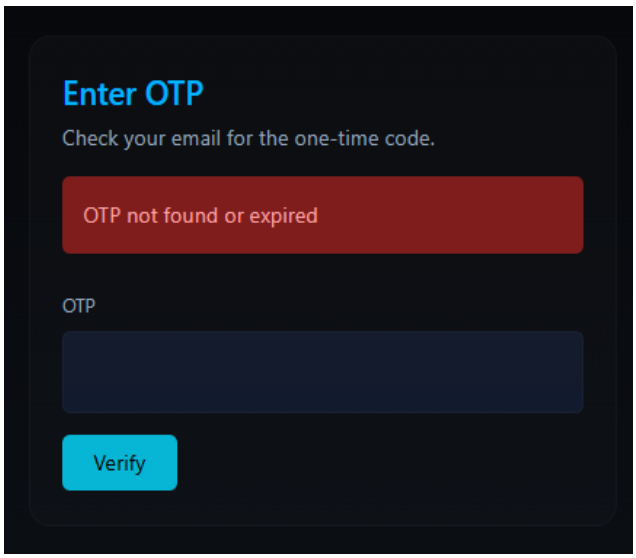
- Using invalid/non-eligible CNIC:



- Using invalid username/password

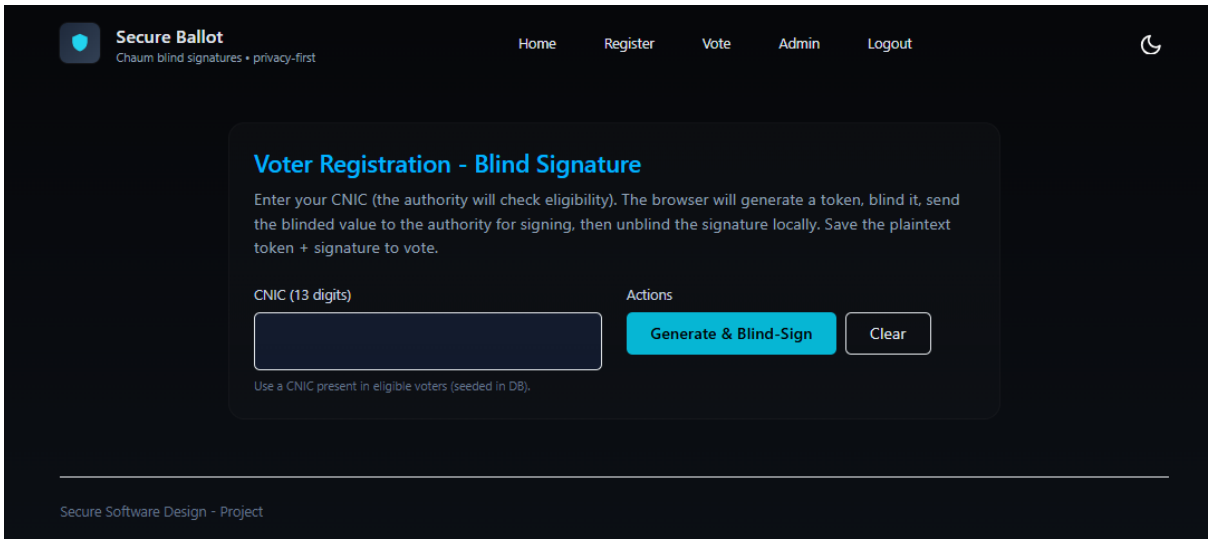


– Invalid or expired OTP

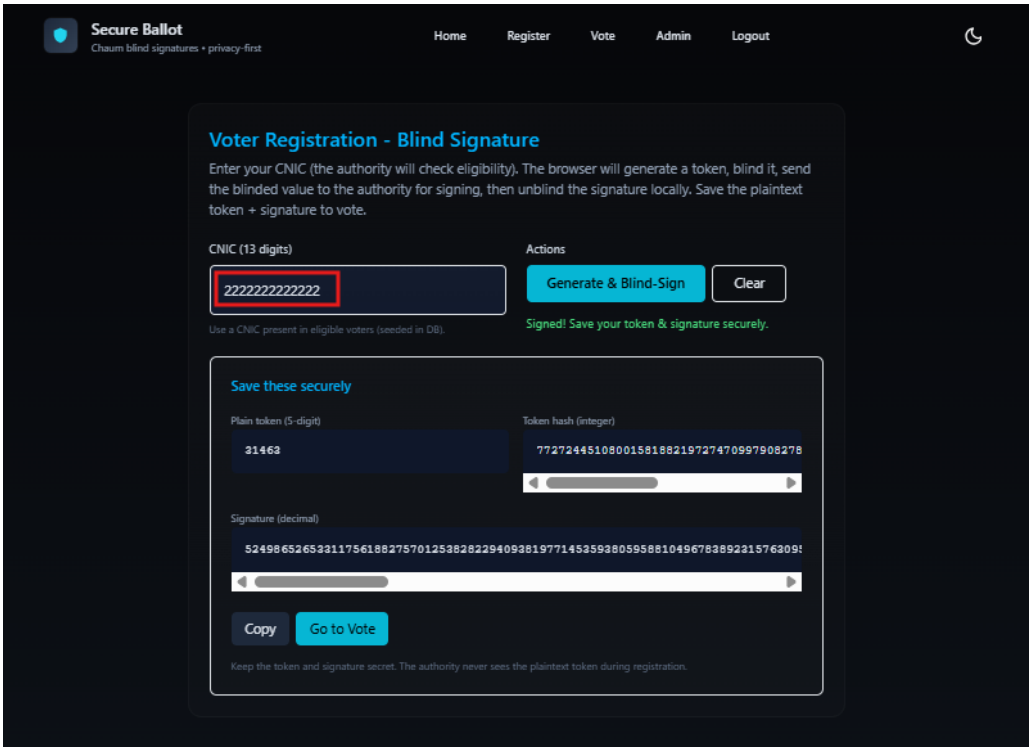


2. Vote Registration & token generation

Normal case ✅



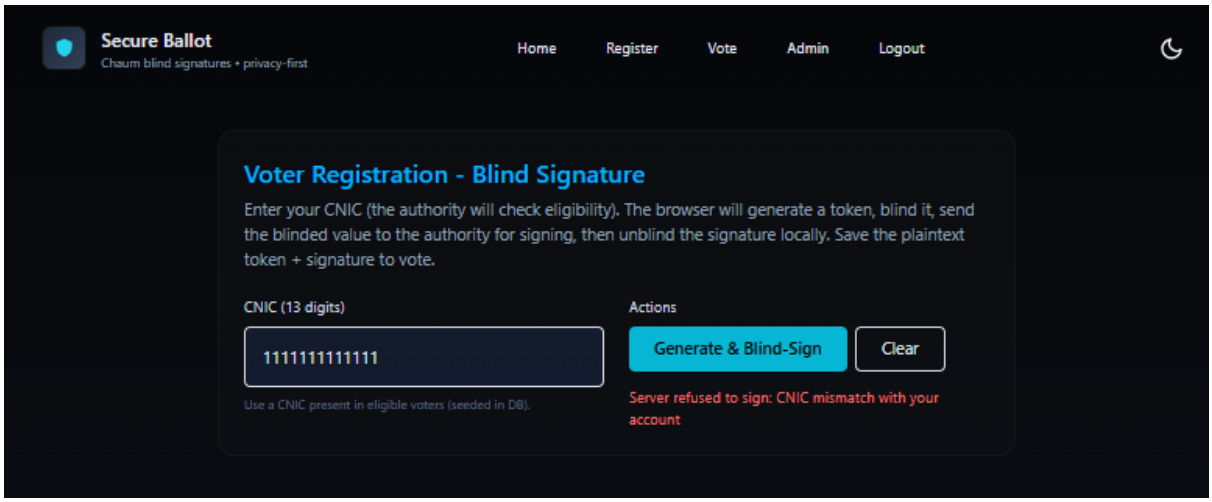
- User is required to enter the CNIC with which the user made their account, any other CNIC not linked to this account would be considered invalid. So for this Normal case:



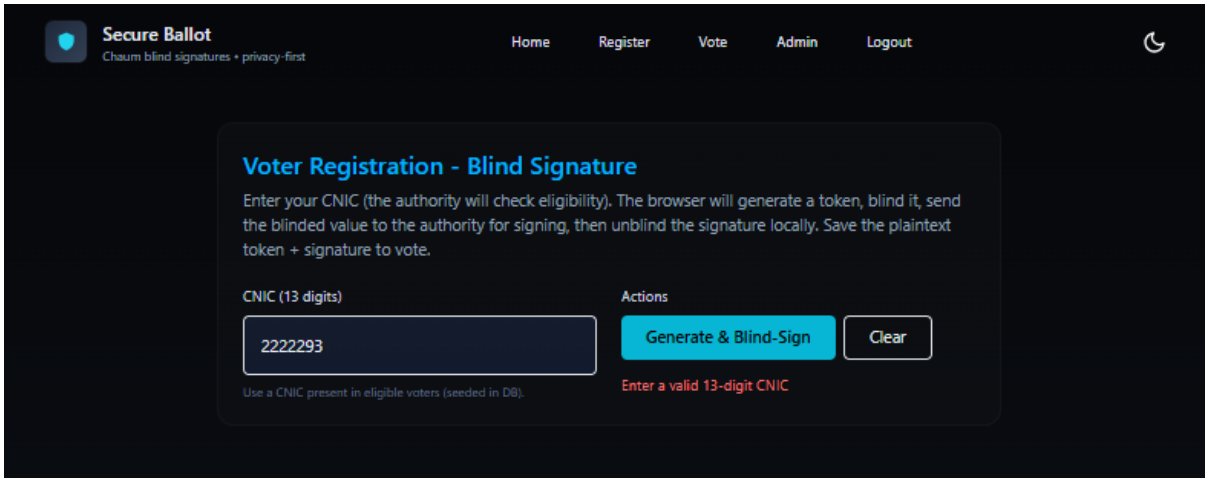
- Token generated successfully

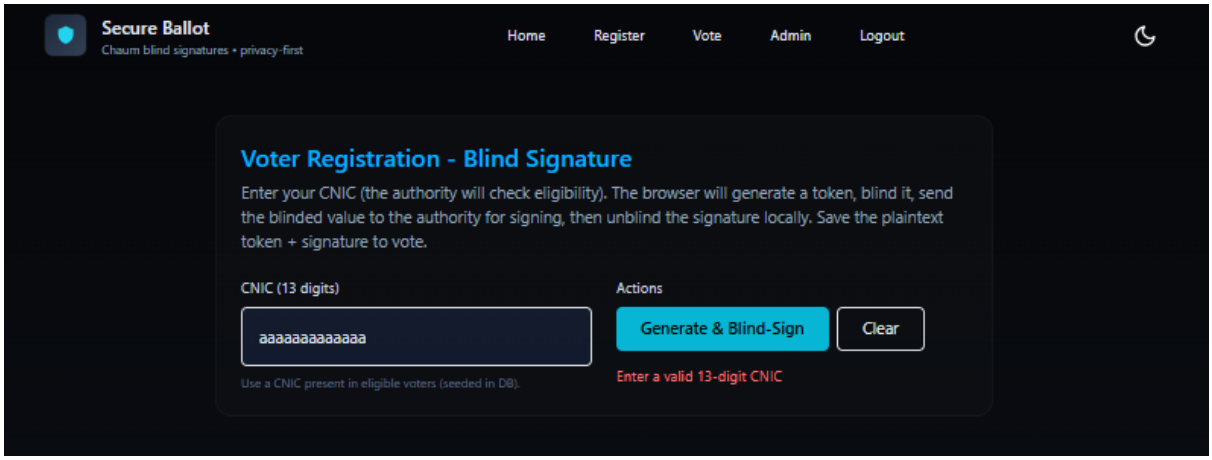
Edge cases

- CNIC not linked to the user account ☹

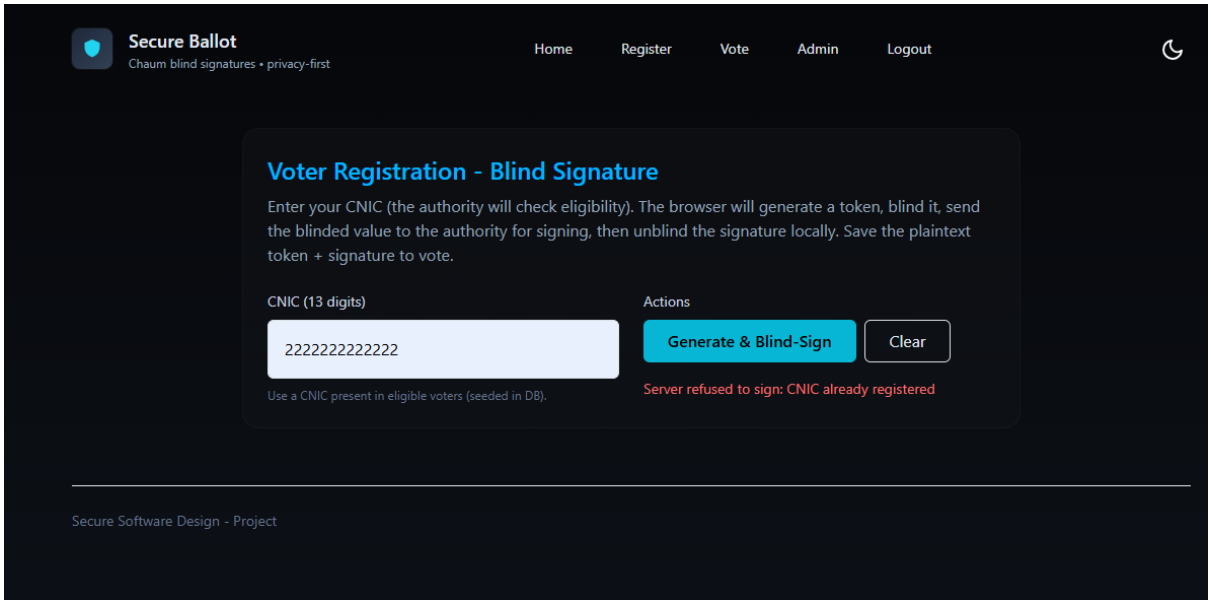


- Invalid format/number of digits ☹





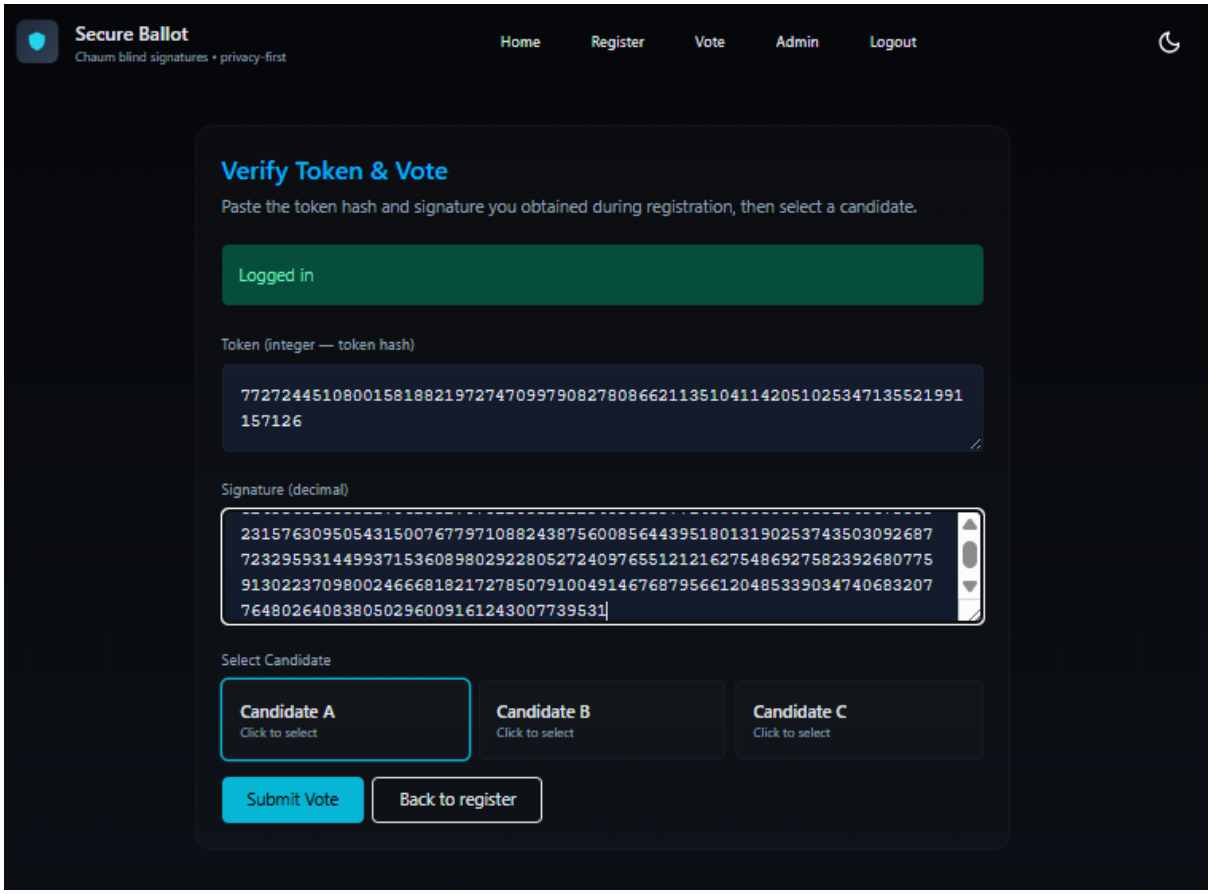
- Using the eligible CNIC to generate the token again:

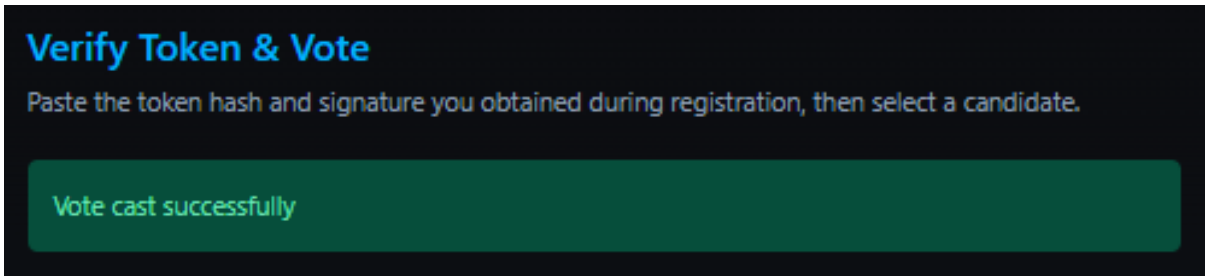


3. Vote Casting

Normal Case ✅

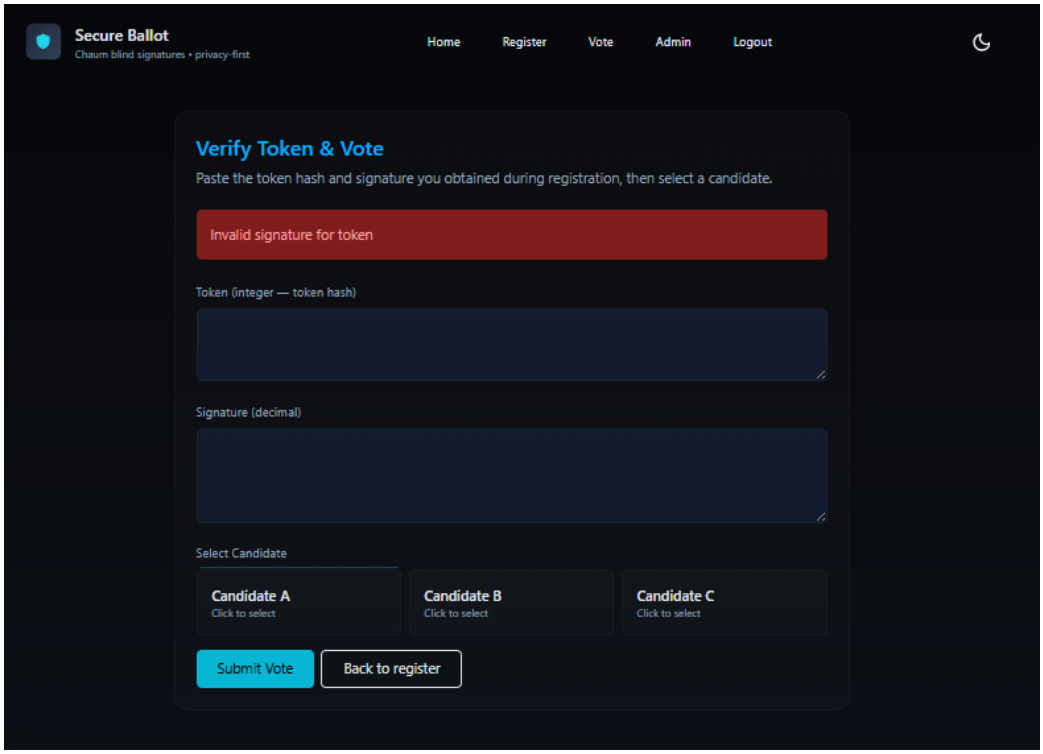
- User enters the token generated using blind signature: (one time token 😉)



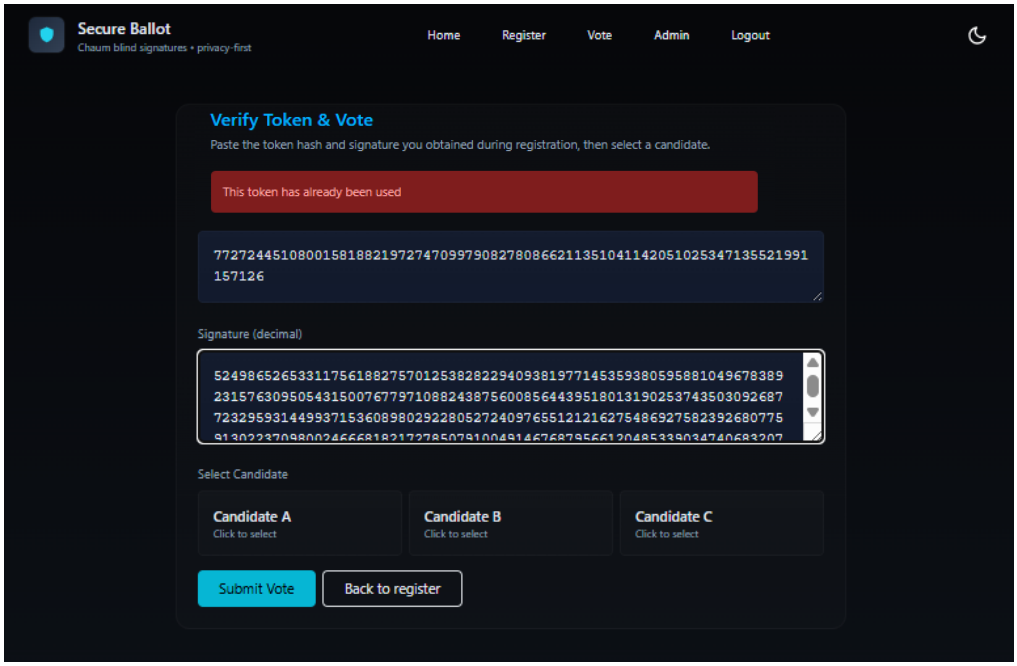


Edge Cases

- User enters invalid token 😵

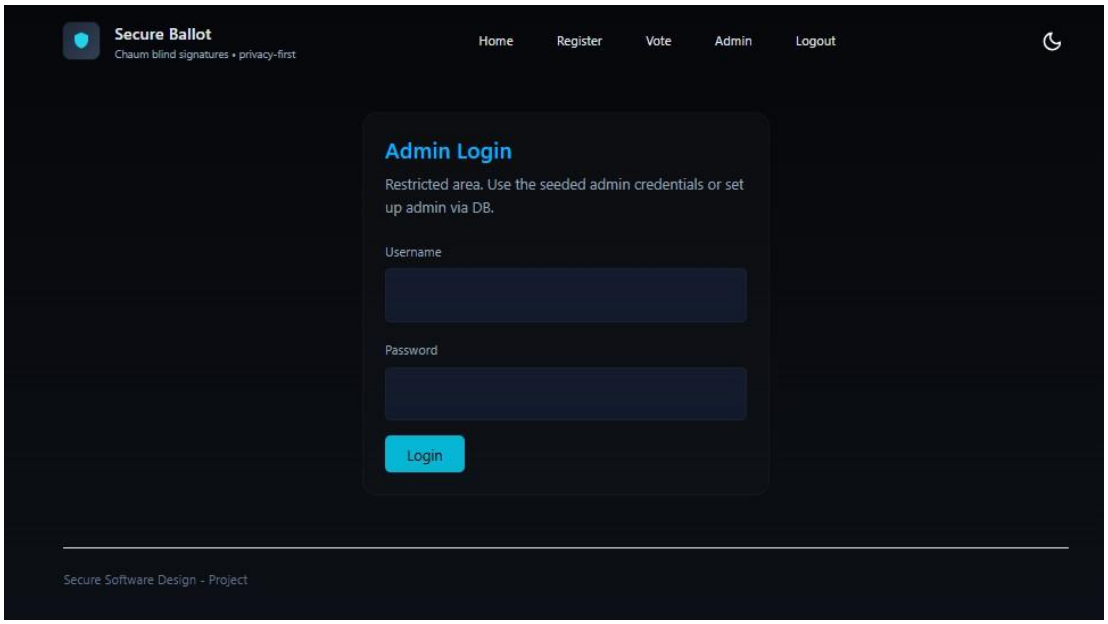


- User enters the same token again through which a vote has already been casted once 😵)

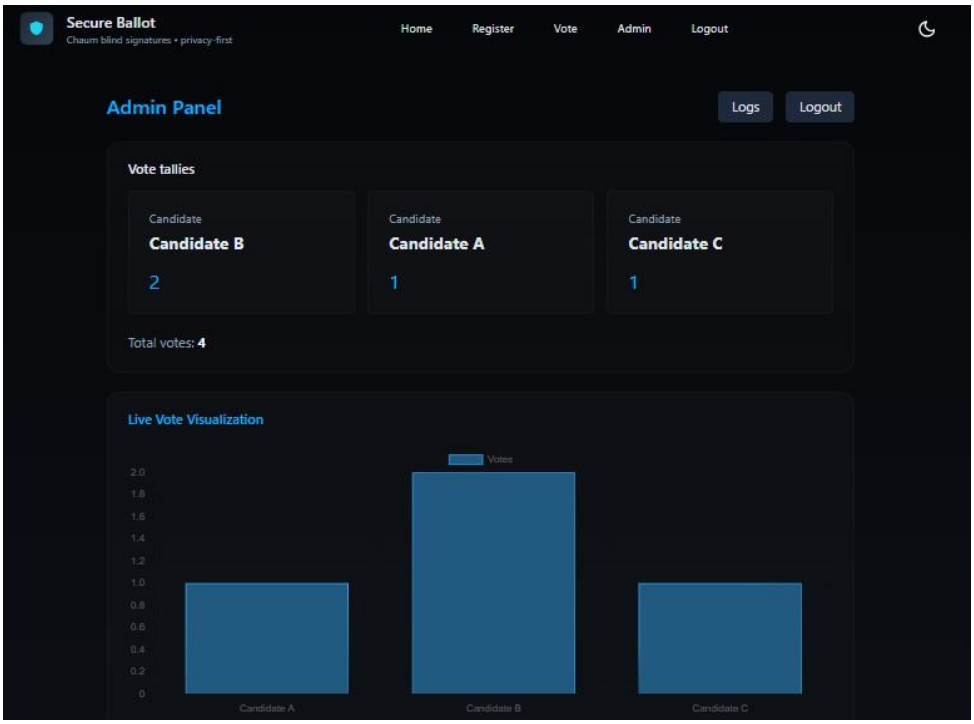


4. Admin Panel

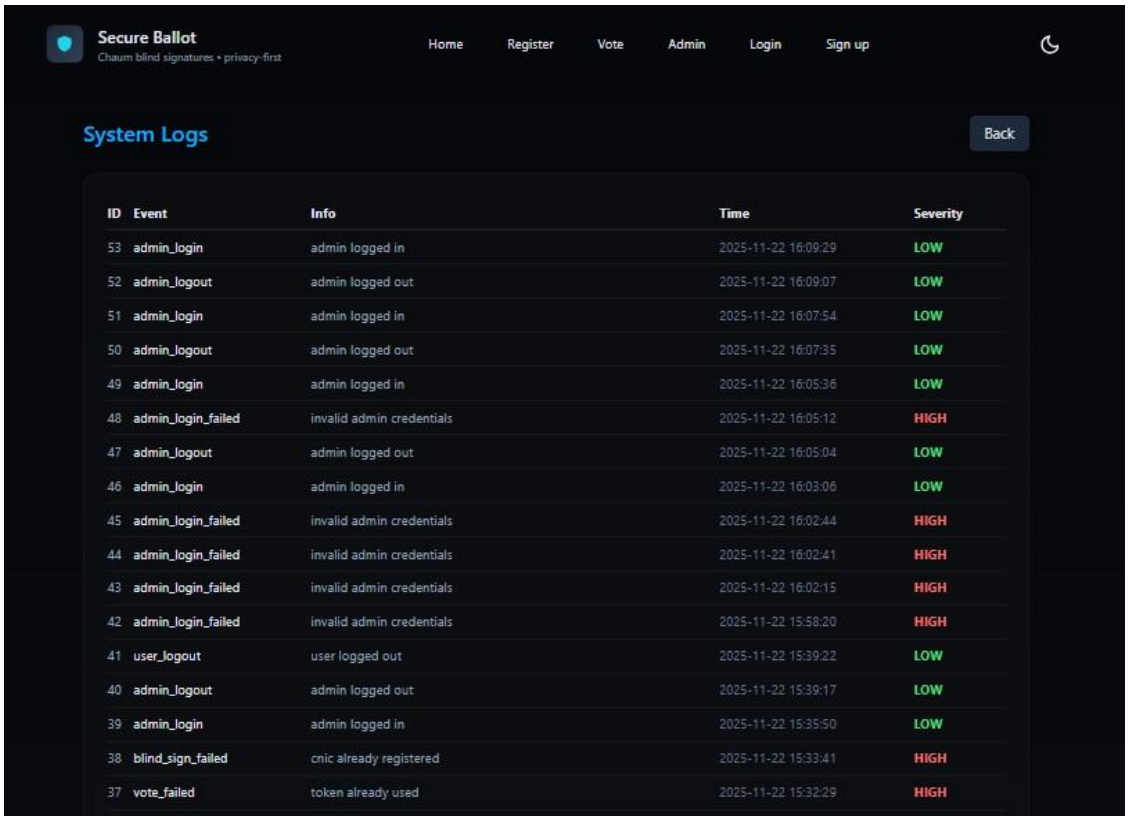
-Admin login:



- Admin can only see amount of votes ; no linkability to user

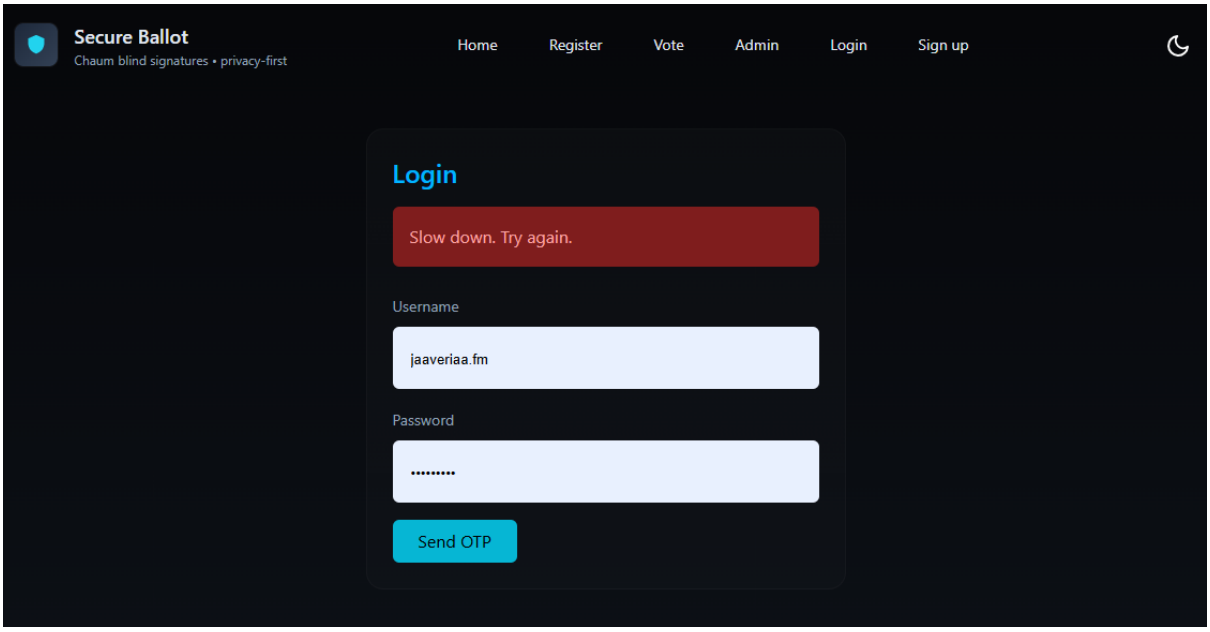


- Admin can also see logs:

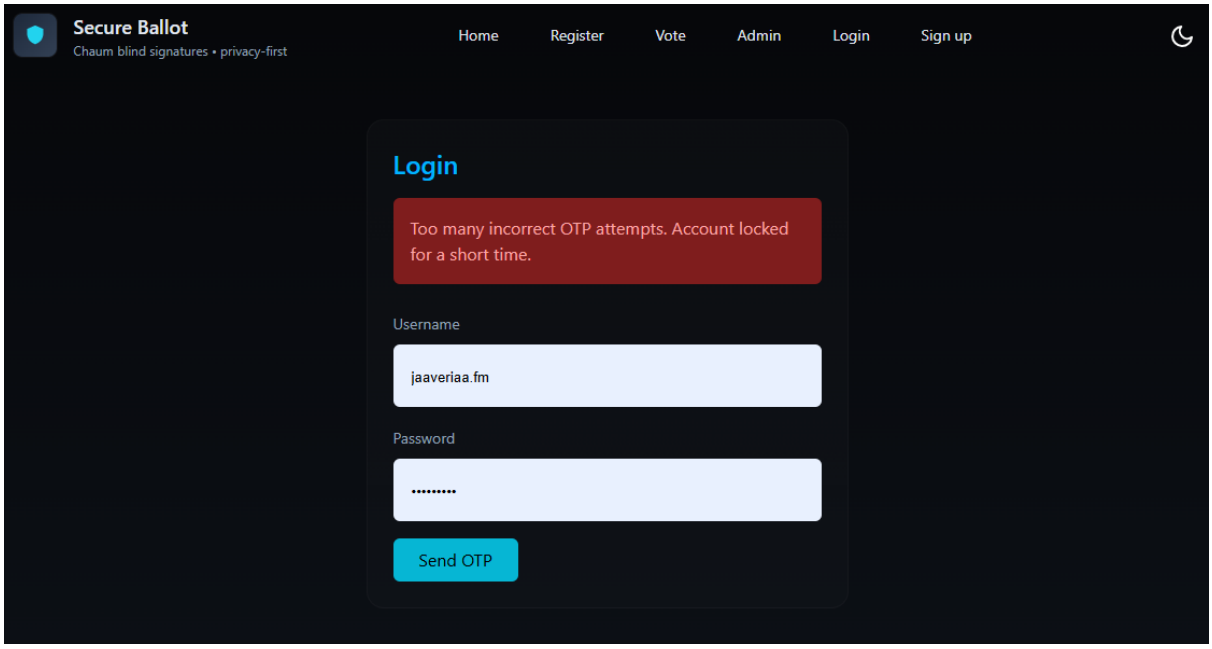


3.2 Nonfunctional (Security) Testing:

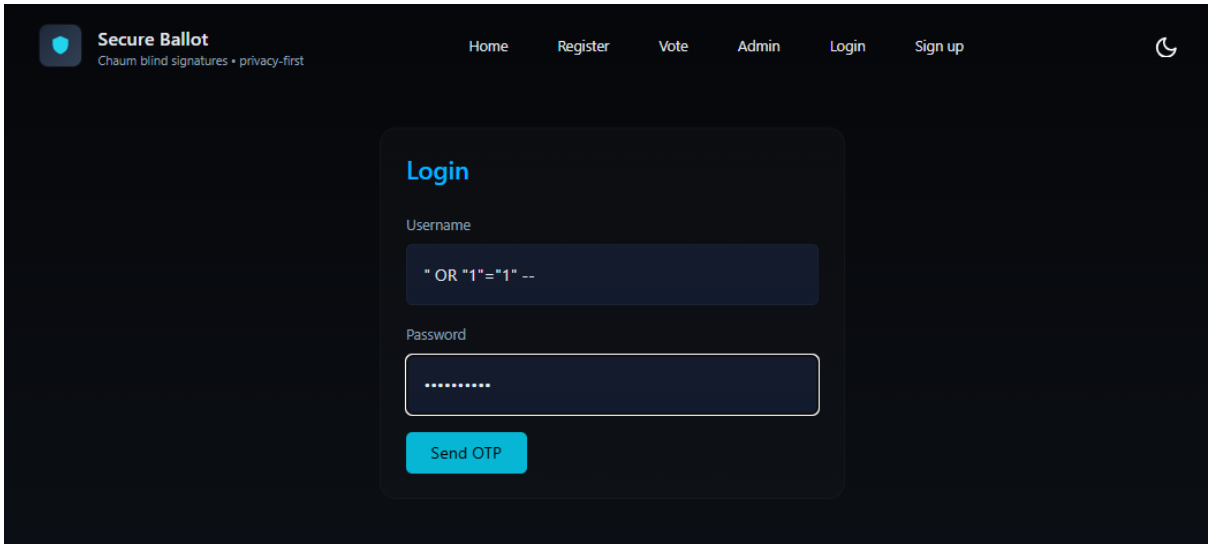
1- Rate limiting applied on crucial points like login to prevent brute force attacks/DOS.

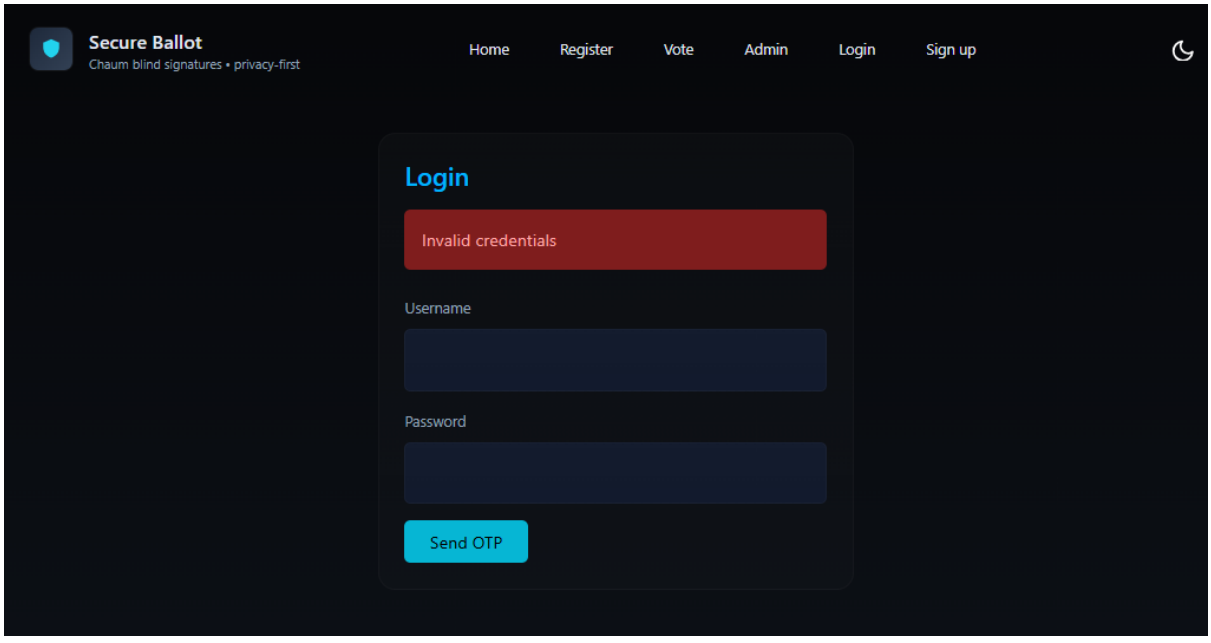


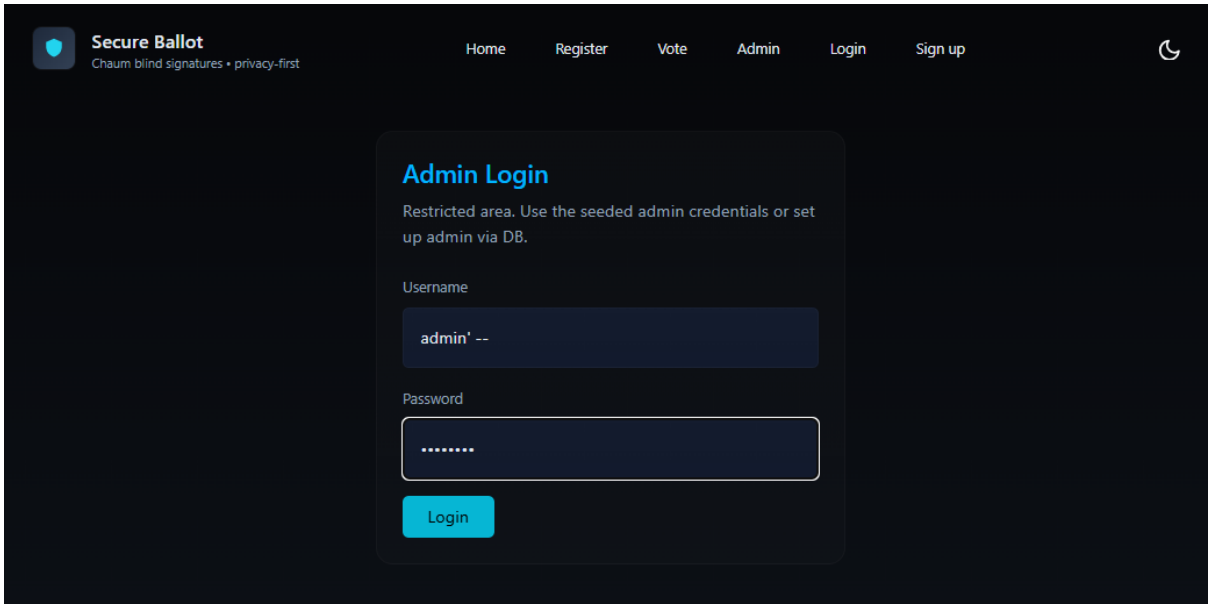
2- Account lockout when too many wrong attempts are made in a minute on MFA:

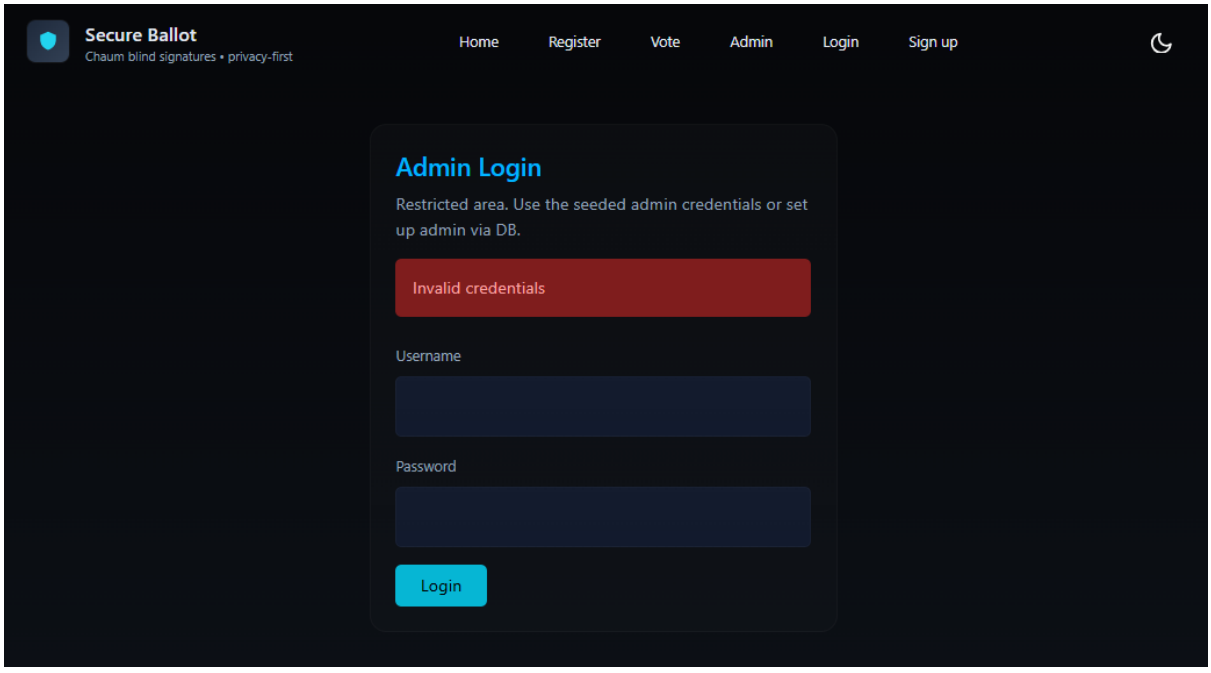


3- SQLi tested against every input point:

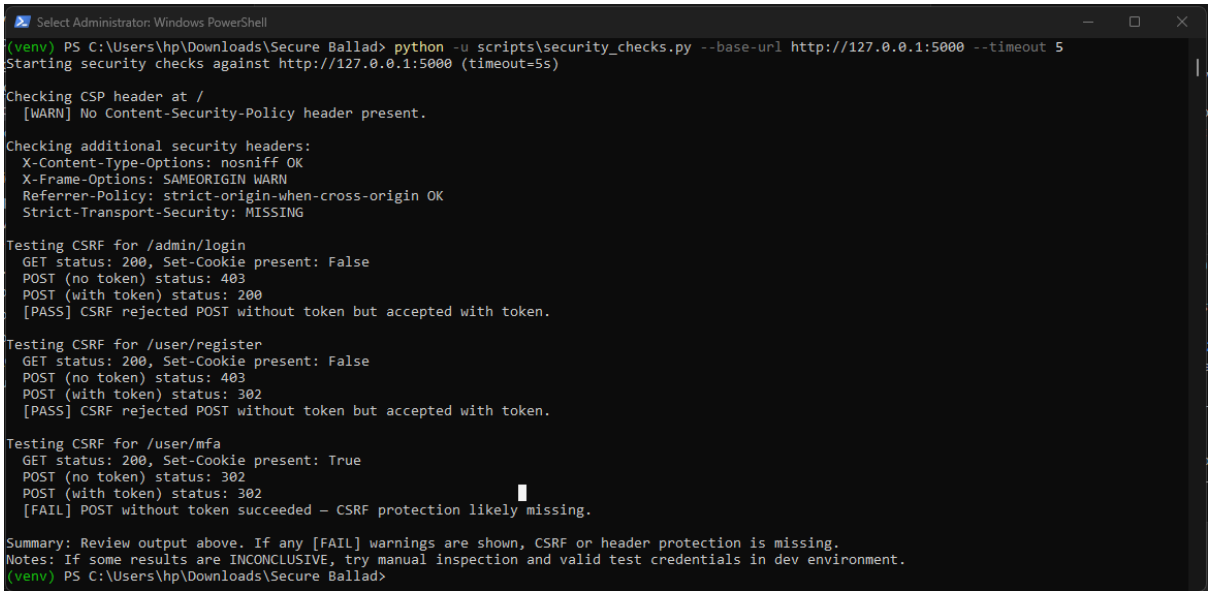




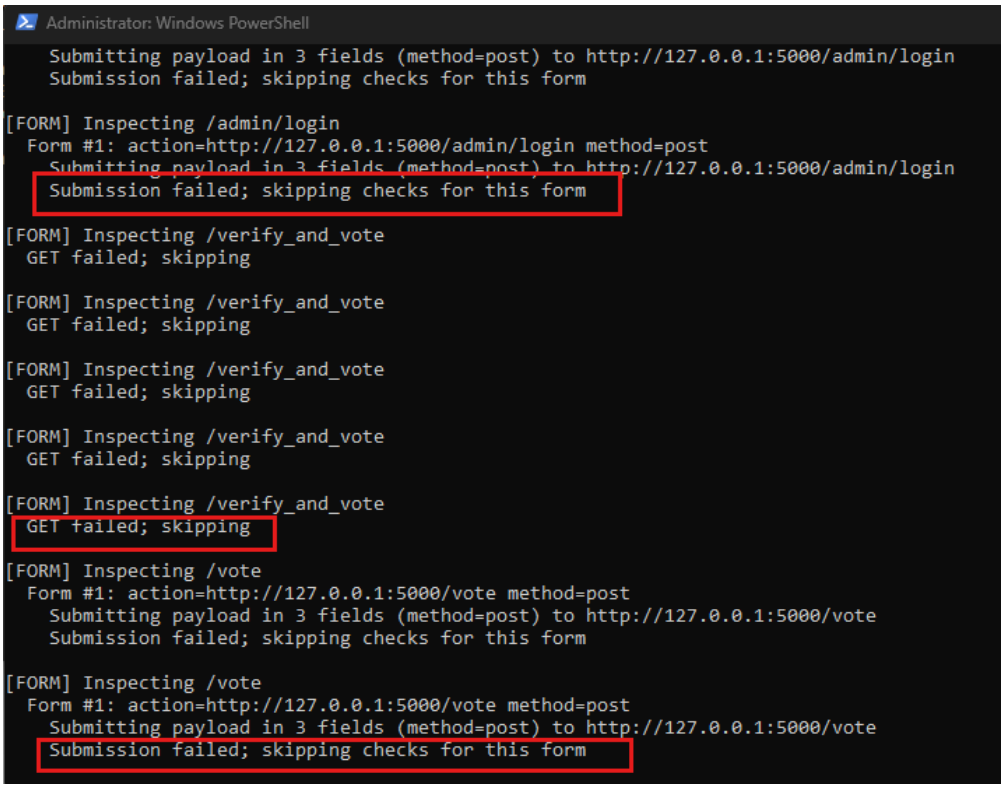

Secure Ballot
Chaum blind signatures • privacy-first
Home
Register
Vote
Admin
Login
Sign up
Admin Login
Restricted area. Use the seeded admin credentials or set up admin via DB.
Username
admin' --
Password
Login


Secure Ballot
Chaum blind signatures • privacy-first
Home
Register
Vote
Admin
Login
Sign up
Admin Login
Restricted area. Use the seeded admin credentials or set up admin via DB.
Invalid credentials
Username
Password
Login

4- Checking for CSRF: ran a script to check and got following results indicating that CSRF tokens are successfully applied which defend against cross-site request forgery reliably



5- Checking for XSS: ran a script to check for any XSS vulnerabilities and got following results:



The highlighted fields indicate that:

→ *"Submission failed; skipping checks for this form"*

It means:

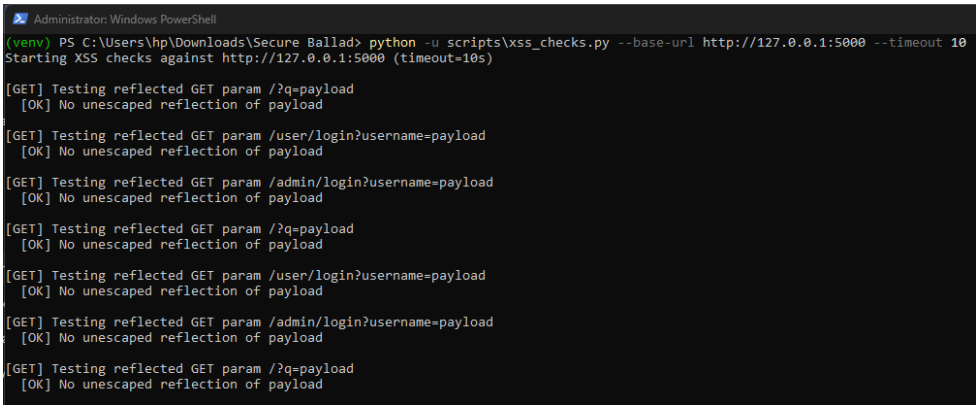
- The scanner attempted to POST malicious payloads to certain forms.
- But our forms require login or our validation rejected the injection attempt.
- Because the submission didn't succeed, the scanner couldn't test reflection.
- Hence: check passed, no vulnerability here.

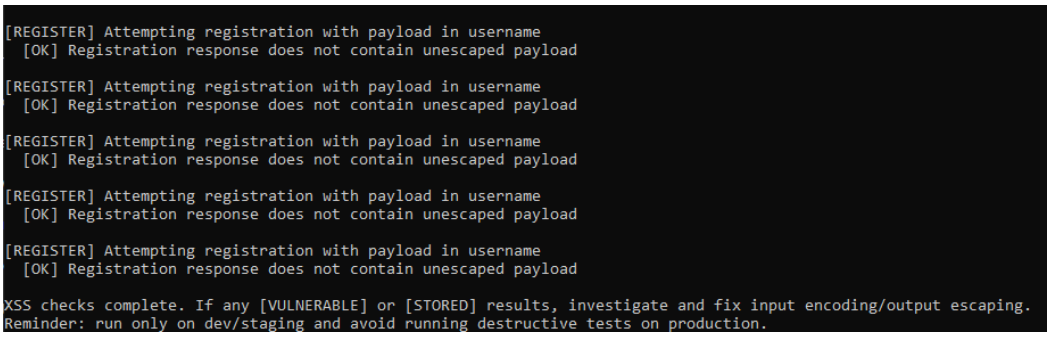
→ *"GET failed; skipping"*

This means the scanner tried to open pages like /verify_and_vote but:

- It requires login
- The endpoint blocked unauthenticated access
- So the scanner couldn't test the form
- Hence: check passed, no vulnerabilities here either.

More checks:





The checks indicate that:

→ *"[OK] Registration response does not contain unescaped payload"* means the scanner tried to register with a username like <script>alert(1)</script> Then checked if we are printing the username somewhere without escaping. Our application did not reflect the payload unsafely which simply means that Stored XSS was not found, so no vulnerabilities here either.

→ Lastly the "[OK] No unescaped reflection of payload" means the tester sent malicious strings like "><script>alert(1)</script> in GET params. Our page did not reflect the payload back in an unsafe way. Our application is not vulnerable to reflected XSS in those tested fields. It means that Our pages are not spitting user input back raw

NO VULNERABILITES FOUND THROUGH STATIC MEANS

3.3 SAST Implementation

SAST Findings & Fixes (Using Bandit)

Static Application Security Testing (SAST) was performed on the project using Bandit, a Python security scanner. Bandit analyzed all Python files and highlighted several potential weaknesses. Below are the critical findings relevant to our system, along with explanations and fixes that were applied.

1. Hard-Coded Credentials

Findings:

Item	Details
ID	B105
Severity	High
Description	Hard-coded default admin username/password in code.
Risk	Attackers who access the codebase or logs can gain admin privileges.

Fix applied:

Item	Details
Strategy	Move credentials to environment variables. Never store passwords in code.
Example Fix	ADMIN_USER = os.getenv("ADMIN_USER")
Impact	Prevents credential exposure and reduces attack surface.

2. Use of assert For Security Logic

Findings:

Item	Details
ID	B101
Severity	Medium
Description	assert used in validation logic. Assertions are removed when Python runs in optimized mode.
Risk	Security checks silently disappear in production.

Fix applied:

Item	Details
Strategy	Replace assert with explicit condition checks + error handling.
Example Fix	if not condition: raise ValueError("Invalid input")
Impact	Ensures security checks always run.

3. Broad Exception Handling (except:)

Findings:

Item	Details
ID	B110
Severity	Medium
Description	Code catches all exceptions without specifying type.
Risk	Real errors get swallowed → debugging harder, failures hidden, logic may break silently.

Fix applied:

Item	Details
Strategy	Catch specific exceptions (sqlite3.IntegrityError, etc.)
Example Fix	except sqlite3.IntegrityError:
Impact	Predictable error handling and better visibility.

4. Weak Cryptographic Practices

Findings:

Item	Details
ID	B303 / B304
Severity	High
Description	Weak or non-approved hash functions used in non-critical places.
Risk	Attackers could brute-force or forge values.

Fix applied:

Item	Details
Strategy	Use SHA-256 / PBKDF2 / bcrypt for all hashing.
Impact	Stronger cryptographic guarantees.

Conclusion:

The Secure Ballad project delivered a working, security-focused e-voting prototype with MFA (time-limited OTPs stored hashed and checked in constant time), secure password hashing (bcrypt), parameterized DB queries and schema constraints, per-user rate limiting and lockouts for MFA and admin logins, CSRF protection and CSP via Talisman, encrypted vote-at-rest option, a safe migration helper, redacted logging, and dev scripts for SAST, XSS and CSRF testing; critical SAST findings were triaged and fixed, with supporting tests and CI integration added. For future enhancements production hardening, adding TLS termination, centralized secret management and rotation, stronger authority key storage, stricter CSP and cookie flags, and continuous monitoring and automated scans are to be considered.