Blockchain based eVoting System

*Note: Project: SC-5 — Group-10

Chandra Sekhar Kondeti
Department of CSE
Indian Institute of Technology Kanpur
Kanpur, India
22111076
chandrask22@iitk.ac.in

Hrishav Raj
Department of CSE
Indian Institute of Technology Kanpur
Kanpur, India
22111403
hrishavr22@iitk.ac.in

Suvam Basak
Department of CSE
Indian Institute of Technology Kanpur
Kanpur, India
22111277
suvambasak22@iitk.ac.in

Navneet Sharma
Department of CSE
Indian Institute of Technology Kanpur
Kanpur, India
22111087
navneets22@iitk.ac.in

Keshav Gupta
Department of CSE
Indian Institute of Technology Kanpur
Kanpur, India
Supervisor
keshavg@iitk.ac.in

Angshuman Karmakar
Department of CSE
Indian Institute of Technology Kanpur
Kanpur, India
Instructor
angshuman@cse.iitk.ac.in

***Abstract*—In an era of the Internet, every sector is pushing towards incorporating web-based services for ease of functionality, and voting is no exception. It has a great prospect of reducing administrative costs and increasing participation conveniently. There is no longer a need to print ballots or set up polling locations because voters can cast their ballots online from any location with a connection to the Internet. Despite these benefits, online voting solutions must be cautiously approached because they present new risks. A single vulnerability may make it easy to manipulate votes on a large scale. Electronic voting systems used in elections must be trustworthy, precise, secure, and efficient. Blockchain technology provides decentralized nodes for electronic voting and is an excellent alternative to traditional electronic voting systems. This project aims to create a reliable Blockchain-based electronic voting system.**

I. Introduction

This project report outlines a fundamental architecture that can be used to create Blockchain-based electronic voting systems (eVoting). The main goal of this project is to solve the issues like 'plague election systems', 'lack of transparency in the voting process', and 'centralized data storage issue.' Blockchain technology is a distributed network of nodes with consists of blocks connected with hash pointers, storing an in-depth record of every transaction that has ever been performed in the network. Even the slightest change to the data will affect the hash pointer. Therefore, if anyone tries to alter voting data on the Blockchain, it can be simply proven that the data has been tampered. However, Blockchain has the drawback of a significantly slower transaction processing rate. Whereas a traditional Database Management system (DBMS) handles a high degree of concurrency, low response time creates a responsive user experience. In this work, we used both DBMS (off-chain) and Blockchain (on-chain) to build an application of eVoting that can scale with the tamper resistance property.

In our project **Blockchain based E-voting system** [1], after conducting a detailed analysis of already available voting applications, we developed a web-based decentralized application (dApp) for eVoting based on Blockchain technology to rectify the issue of data manipulation and tampering. The concept involves one person appointed as *Senate Election Commissioner* who is responsible for the deployment of dApp (smart contract[2] and web services), registration of *Candidates*, setting time of *beginning* and *end* of voting and publication of *election results*. The *voters* can register by authenticating by Institute username. Voting data is stored in the Database and Blockchain, i.e. **Off-chain** and **On-chain**. To preserve the anonymity of the voters, we create a hash of their voting records and store it in the Blockchain. This hash is later compared to a previously computed hash and checked with On-chain data to ensure that the Database is consistent and unaltered. Storing vote cast hash in a public Blockchain and signing the transaction by the voter's private key makes the election **transparent**. Initially, the dApp was tested into a local Blockchain network and then deployed to an Ethereum testnet [3].

The system can be made more secure for commercial purposes by deploying this dApp on a Private Blockchain, providing privileged access to senate members. However, that reduces the transparency of the voting system. In this implementation, we adopted two approaches. (i) storing voting data as plain text, (ii) then we modified this approach to store data as a hash value to protect voters' privacy. The current design and implementation approach is yielding trustworthy

[1] https://git.cse.iitk.ac.in/suvambasak/sc5-evoting-g10.git
[2] https://ethereum.org/en/smart-contracts/
[3] https://sepolia.etherscan.io/

results, and we have achieved the objectives for this project in accordance with the project requirements.

The subsequent content is organized like this: Section II discusses some related work already been done. Our system design is illustrated in Section III. The methodology of registering voters, generating voting hash, and evaluating tampered resistance before publishing results shown in Section IV. The section V add a discussion on some issue of eVoting systems. In the last Section VI add the conclusion and future direction.

II. Related work

Initially we carried out an online survey about the existing online Voting platforms to gain information about the way we need to develop our dApp. There have been several attempts to develop voting system apps, some of which have faced challenges and drawbacks, including:

Voatz: Voatz is a mobile voting app that has been used in several US states. However, the app has faced criticism for security and privacy concerns, with some experts suggesting that the app may be vulnerable to hacking or other forms of tampering [1].

Democracy Live : Democracy Live is an online voting system that has been used in several US states. However, the system has faced criticism for accessibility issues, as it may not be accessible to voters with certain disabilities [2].

Follow My Vote : Follow My Vote is a Blockchain-based voting system that aims to provide a highly secure and transparent voting process. However, the system has faced challenges in gaining widespread adoption, as some experts have raised concerns about the practicality of using Blockchain technology for voting as Blockchain was less secure in past days [3].

SecureVote: SecureVote is a mobile voting app that has been used in several countries. However, the app has faced criticism for its complexity, with some users finding it difficult to navigate and use effectively [4].

We also carried out literature survey for the project details of which are cited in Biblography [5] [6] [7] [8].

III. System design

Based on the proposed project requirements, we worked on system design for a decentralized eVoting application based on Blockchain technology. We incorporated the following steps to implement eVoting system:

Setting up the environment for Blockchain technology: We carried out research on available frameworks for Blockchain technology like Hyperledger[4], Ethereum[5], Multichain[6], etc. As Ethereum provides the complete implementation of Blockchain technology, so that the complex work of creating the complete Blockchain is not required and more focus is on building the application. We set up our environment for eVoting on Solidity[7] as it is designed to write Smart Contracts that can be executed on the Ethereum Virtual Machine[8] (EVM).

Creating a Smart Contract for eVoting system: Blockchain stores data as a distributed ledger. Each block of data contains a set of transactions. These transactions are verified by the Smart Contract before storing into a block of the Blockchain. If not verified, the transactions cannot be added into the block.

- The Smart Contract is written in solidity language.
- Remix[9] provides a web-based Smart contract development and deployment platform.
- The initial testing and deployment of the smart contract were done on Remix IDE.

Front end: To access front end of the eVoting dApp, we have used HTML[10] with the Bootstrap[11] library. Bootstrap is widely used to design responsive and mobile-first websites. It provides a set of pre-built user interface (UI) components, such as buttons, forms, modals, and navigation menus, that can be easily customized and integrated into our website. The webpages are dynamically generated in the web server by the backend framework based on the page request.

Back end: Back-end Services are written in (i) Flask and (ii) Web3.py.

- **Flask**[12] is a web framework written in Python. That is used to build web applications. Flask provides the tools and functions to manage Databases and user login sessions easily. In this project, Flask interacts with the Database to store and retrieve Candidates, Voters, and Voting details and serve dynamic webpages on request.

Web3.py, a Python library used to interact with the Ethereum network, provides a high-level interface for working with the Ethereum Blockchain. All the voting transactions with the Smart Contract were carried out using Web3.py. Infura[13] web services were incorporated to provide an API endpoint to the Ethereum Blockchain network.

Database: SQLite[14] stores Off-chain data on the Database tables, while on-chain data is stored on the Ethereum Blockchain.

A. Application architecture

The high-level design of the eVoting system is shown in Figure 1. All the Voters and the admin interact with the web server from a web browser. The web server is running backend services to serve the request. Based on the request, the server communicates with Database, authenticates the Voter and the Admin, and serves the dynamic web pages. Handle the vote cast, generate the voting hash add that to the Smart Contract. Verify the consistency with the Blockchain before publishing the results.

[4] https://www.hyperledger.org/
[5] https://ethereum.org/en/
[6] https://www.multichain.com/
[7] https://soliditylang.org/
[8] https://ethereum.org/en/developers/docs/evm/
[9] https://remix.ethereum.org/
[10] https://html.com/
[11] https://getbootstrap.com/
[12] https://flask.palletsprojects.com/en/2.2.x/
[13] https://www.infura.io/
[14] https://sqlite.org/index.html

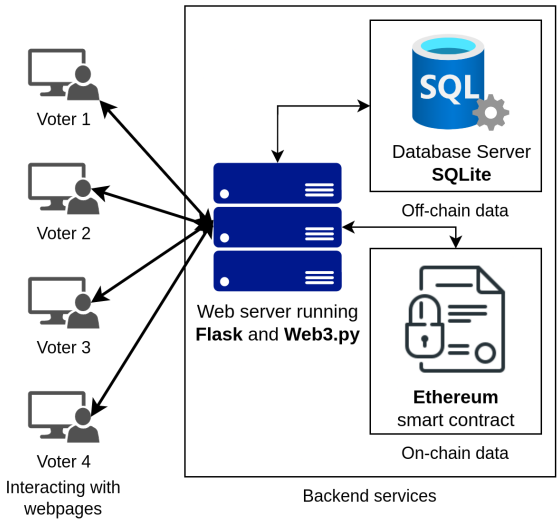

Fig. 1. Overall architecture of the application

The data stored on On-chain and Off-chain is shown following subsection:

1) Off-chain data: A relational Database table consists of three tables shown in Figure 2 (i) Candidate: list of the candidates, (ii) Voter: list of the registered voters, and (iii) OTP: stores unverified OTPs.

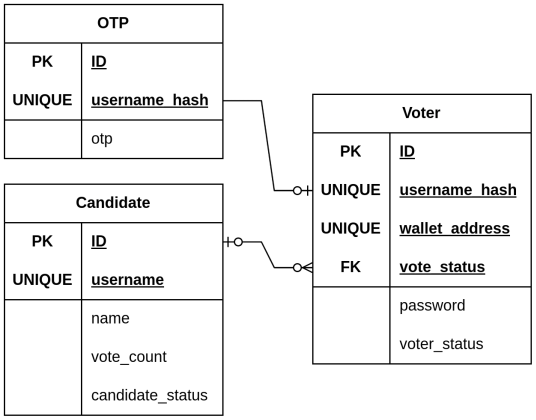

Fig. 2. Database design

2) On-chain data: The time (UNIX timestamp) of election *beginning* and *end* is stored in the Blockchain. Only the owner (who deployed, Admin) of the Smart Contract can update this data. It also includes the total vote count and a pair of key-value maps i.e. (i) Candidate votes: Map(Hash(candidate username), Vote hash), (ii) Voter timestamps: Map(Hash(Voter), UNIX timestamp) shown in Figure 3.

IV. Methodology

In this section, we illustrate the methodology of E-voting process. The eVoting system involves involves four core modules which are illustrated in subsequent paragraphs.

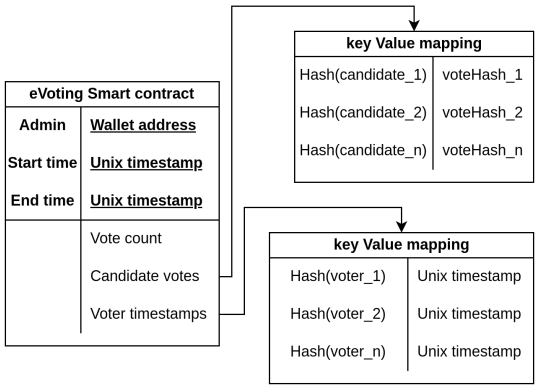

Fig. 3. Data in Blockchain

A. Application deployment

In the project file structure, we have (i) directory `CSV` containing the CSV file[15] with the details of the candidates, (ii) directory `admin` containing a JSON file[16] with details of admin, and (iii) directory `contract` with a Smart Contract source code file[17]. The Admin deploys the Smart Contract with a given Python script named `deploy.py`[18]. On the successful deployment of the Smart Contract, the contract address gets automatic updates into the `admin.json` file and a new `ABI.json` file gets created in `contract` directory. In the next step, we initiate the Flask application. With a fresh start, it automatically creates Database tables read (i) `admin.json` and (ii) `candidates.csv` files initialize the tables. The `README.md` file[19] includes all the steps to create an environment and execute these scripts and start the application.

B. Voter registration

The Voter registration by verification of the Institute username is shown in Figure 4. It involves the following steps:

1) A new Voter needs to enter his/her username, wallet address, and password, and confirm password to be registered.
2) Server interacts with the Database to verify whether the username and wallet address is unique.
3) If the details are not unique, voter is not registered.
4) If found unique, interact with Ethereum to check valid wallet address.
5) If it is invalid address, Registration is declared failed.
6) If it is a valid address, a 6-digit OTP is generated by server and a email is sent.

[15] https://git.cse.iitk.ac.in/suvambasak/sc5-evoting-g10/-/blob/main/CSV/candidates.csv
[16] https://git.cse.iitk.ac.in/suvambasak/sc5-evoting-g10/-/blob/main/admin/admin.json
[17] https://git.cse.iitk.ac.in/suvambasak/sc5-evoting-g10/-/blob/main/contract/eVote.sol
[18] https://git.cse.iitk.ac.in/suvambasak/sc5-evoting-g10/-/blob/main/deploy.py
[19] https://git.cse.iitk.ac.in/suvambasak/sc5-evoting-g10/-/blob/main/README.md

7) OTP generated is stored in Database and validated against the OTP entered by the Voter.
8) If there is a mismatch, Registration fails else Voter is therafter declared registered and this OTP value is deleted by the Server.

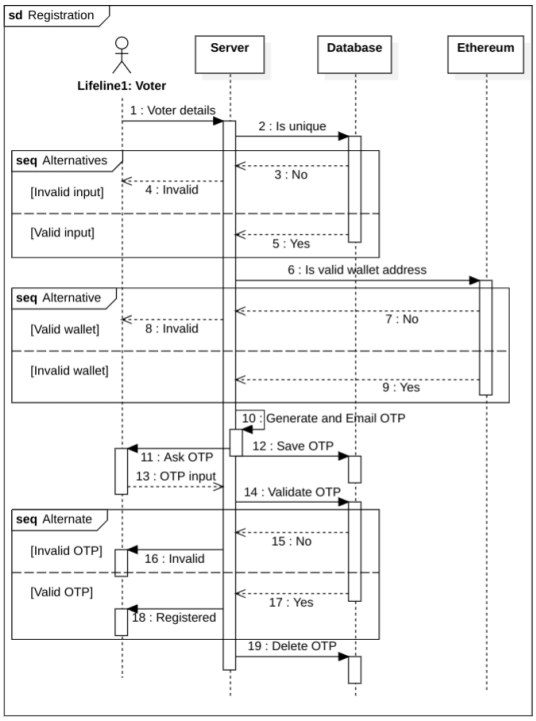

Fig. 4. Registration sequence diagram

C. Casting vote

The process of vote-casting is shown in Figure 5

1) Voters enter the login credentials
2) Server interacts with the Database to validate credentials.
3) Invalid credentials
4) Login failed
5) Valid credentials
6) Query all active (not blocked) candidates and voter details
7) Gets the Voter details and candidate list
8) Share the page with voter details and the candidate list
9) Voter selects a Candidate from the list
10) Ask for the private key
11) Voter enters the private key
12) Queries all the voters who have voted for the selected candidate
13) Gets the list of voters
14) Compute vote hash: The computation vote hash concatenates all the username hash from the voter list and calculates the sum of their voter ID and creates a hash of concatenated username hash with the sum of voter IDs. For example: If the Voter V_3 has selected candidate C_3 and V_1, V_5 are the Voters already voted for the candidate C_3, then the vote hash (V_H) will be given a hash function $H()$:

$$V_H = H(H(V_1)|H(V_3)|H(V_5)|(ID_{V_1} + ID_{V_3} + ID_{V_5}))$$

15) Create and sign a transaction with the Voter's private key and wallet address with the data CandidateVotes($H(candidate)$, V_H)
16) Sends the transaction to the Ethereum network
17) The Smart Contract validates the time
18) The transaction fails if an invalid time
19) The error message is given to the Voter
20) Store/Update the key-value pair CandidateVotes($H(candidate)$, V_H)
21) Sends the transaction receipt
22) Update the vote cast into the Database
23) Show the transaction hash

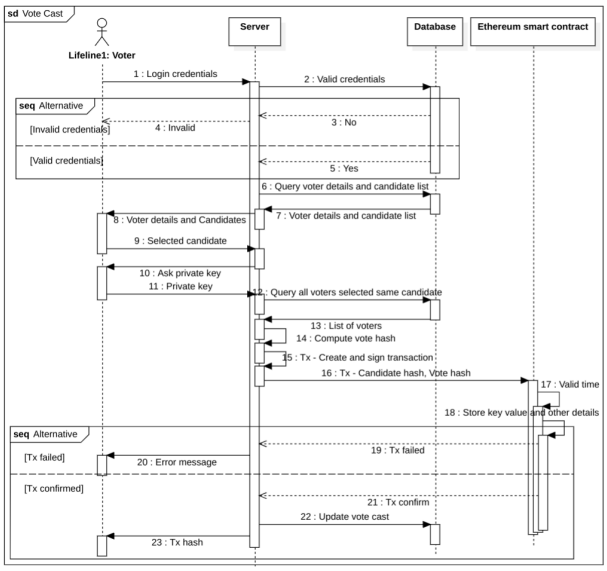

Fig. 5. Vote cast sequence diagram

D. Publish results and Tamper resistance

In our implementation, Admin can see the state of results during election time whenever they log in. Once the voting time ends, for each candidate, we have a hash value stored in Blockchain, which is a continuous process i.e., whenever a vote is cast, a new hash value is generated, concatenating the existing hash and new voter hash. Similarly, the hash values for all candidates can be generated locally from the vote-casting data available in the Database. If the locally computed hash value matches with the hash value stored in the Blockchain, it implies the results are not tampered with. This process of verification is executed when Admin publishes the results shown in Figure 6. Essentially the sequence of the step is as follows:

1) Admin enters login credentials
2) Validates the credentials from the Database
3) If the credentials are not valid

4) Login denied
5) If the credentials are valid
6) Query all the Voters and Candidates list
7) Gets the Voters and Candidates list
8) build and show the admin panel page
9) Admin publish the result
10) Query the Admins wallet address from the Database
11) Gets the Admins wallet address
12) Fetch Candidate votes hash maps from Blockchain
13) Start Contract checks the wallet address (if voters try to fetch the same data before the election ends, Smart Contract rejects)
14) Gets the key value hash map
15) Query the election results from Database
16) Gets the election result (How has voted whom)
17) Computes the vote hash again as shown in the previous sub-section IV-C
18) Match the computed hash with the hash retrieved from the Blockchain
19) If all the hash are matched, make the result status public in the Database
20) Databased updated
21) Publish the result
22) If any of the hash are mismatched, failed to publish the results

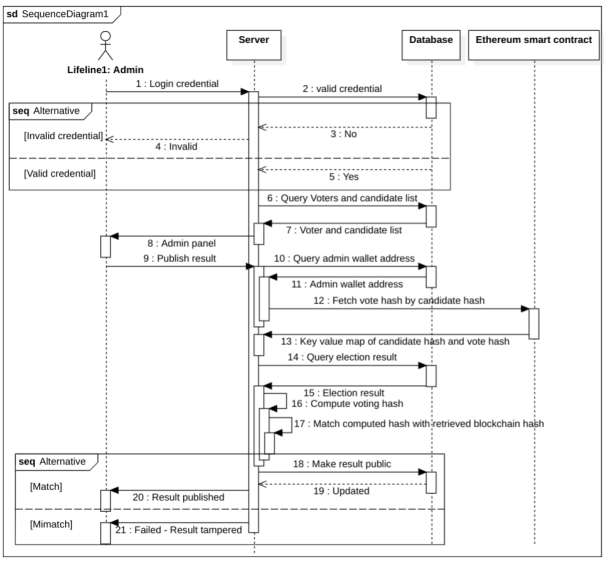

Fig. 6. Publish result sequence diagram

V. Discussion

Our first approach involved storing the voting data in plaintext format on the Blockchain, which means that the information about which voter voted for which candidate was publicly available on the Blockchain. This approach compromised the privacy of voters and their voting choices.

Our second approach addressed this issue by hashing the voter and candidate usernames using a SHA256 hash function before storing the data on the Blockchain. This approach provided a certain level of privacy because it was not possible to know which voter voted for which candidate by looking at the Blockchain data directly. However, it still had a flaw because by counting the number of voting transactions related to each candidate's hash, one could predict which candidate had received how many votes.

In our current approach, we came up with a new technique of hashing the vote cast data discussed in the previous subsection IV-C. This approach made it difficult to know which voter had voted for which candidate, as the voter hash was not directly stored on the Blockchain. Additionally, the data for each candidate was hashed multiple times with other values before being stored on the Blockchain, which made it difficult to predict the voting results by counting the number of transactions related to each candidate's hash because it stores only one hash value corresponding to one candidate hash irrespective of the number of Voters voted. This approach provides a higher level of privacy but at the cost of transparency.

The content highlights the trade-off between privacy and transparency in storing voting data on a Blockchain. Storing concatenated hashed data on the Blockchain makes the process more secure but less transparent, which can be challenging in some situations.

VI. Conclusion and Future Work

This project justifies all the essential requirements to set an eVoting system based on Blockchain technology. To summarize, all the requirements mentioned in the Project document has been addressed. **Additional features** like

- Incorporation of OTP feature
- Setting election start, end time and extension of time from the admin panel
- Blocking/Unblocking of Voter and Candidates

has been implemented. For future implementation, we recommend it to be deployed on a Private Blockchain as it will be more secure.

VII. Contribution

The Contribution of members to the Project work is listed below.

- **Suvam Basak**
  - Frontend: Design webpages
  - Backend: Flask web application, Database design
  - Testing: Integration testing
- **Navneet Sharma**
  - Backend: Setting up the environment and Writing Smart Contract for eVoting
  - Testing: Integration testing
- **Hrishav Raj**
  - Backend: Invoking Smart Contracts from Python with Web3.py, Deploying the Smart Contract to Ethereum test network with python script
  - Testing: Integration testing
- **Chander Shekhar**
  - Backend: Python script for sending the email.

REFERENCES

[1] Specter, Michael A., James Koppel, and Daniel Weitzner. "The ballot is busted before the Blockchain: A security analysis of voatz, the first internet voting application used in us federal elections." Proceedings of the 29th USENIX Conference on Security Symposium. 2020..

[2] Specter, Michael A., and J. Alex Halderman. "Security Analysis of the Democracy Live Online Voting System." USENIX Security Symposium. 2021

[3] https://followmyvote.com/

[4] Abuidris, Yousif, et al. "Secure large-scale E-voting system based on Blockchain contract using a hybrid consensus model combined with sharding." Etri Journal 43.2 (2021): 357-370.

[5] F. . Hjálmarsson, G. K. Hreiarsson, M. Hamdaqa and G. Hjálmtýsson, "Blockchain-Based E-Voting System," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2018, pp. 983-986, doi: 10.1109/CLOUD.2018.00151.

[6] Agora. "Bringing our voting systems into the 21st century." (2017)..

[7] McCorry, Patrick, Siamak F. Shahandashti, and Feng Hao. "A smart contract for boardroom voting with maximum voter privacy." Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21. Springer International Publishing, 2017.

[8] Dagher, Gaby G., et al. "Broncovote: Secure voting system using ethereum's Blockchain." (2018).