Simayi - Blockchain Voting System

Youngjae Moon, Alex Richardson, and Azhar Hasan

***Abstract*—In this work, we present Simayi, which is a decentralized Blockchain voting system. We discuss how it can be used by end-users, the overall system design, benefits, and limitations. We examine the architecture, design, and implementation of this system, as well as the potential benefits and challenges in regards to adopting a blockchain-based voting platform.**

I. Introduction

A. Background

THE traditional electoral process has been problematic in various ways, such as voter fraud, lack of transparency, and logistical challenges. Blockchain technology has the potential to address these issues by providing a secure, transparent, and tamper-proof system for recording and validating votes. Ever since Bitcoin and other related currencies were released into the world [1] [2], the idea of using blockchains for validating other sorts of records in a cryptographically secure fashion has proliferated [3]. While blockchain voting seems attractive on the surface, it generally currently suffers from low transaction speeds, and overall a false sense of security with current cryptographic techniques - a vote that is ”secured” with technology which is actually compromised can be in many ways more damaging than an illegally counted hand-casted vote [3]. Paradoxically, these digital record systems could make voting and other such record-keeping systems less secure [4].
Nonetheless, blockchain is attractive for automated, trust-less based communication and maintaining of records, and thus, for our project, we wanted to experiment implementing our own application and blockchain framework from scratch. We wanted to create a Minimum Viable Product consisting of a theoretically cryptographically secure framework for transmitting and recording votes, elections, and users.

B. Contributions

Our contributions in this research project are as follows:

1) A proof of concept for a scalable and decentralized blockchain voting system that we have called Simayi. It uses standard off the shelf web and cryptographic technology that can be easily maintained and deployed, such as RSA signing [5] and SHA256 hashing, and is full stack, in that we have implemented everything from a persistent proof-of-concept backend to a front-end interface.
2) Implemented our own blockchain code completely from scratch. Ours is heavily based on the Bitcoin paper’s description of a blockchain for simplicity and ease of conveyability [1].
3) Discussed the technical benefits and limitations of our framework, and show how it can be used by end users.
4) Speed of transactions on this system depending on the difficulty of our proof of work.
5) Proposed potential refinements to this framework are acknowledged and discussed.

Youngjea Moon is currently a M.Sc. in computer science and Engineering Graduate Fellowship recipient at Vanderbilt University. e-mail: youngjae.moon@Vanderbilt.Edu

Alex Richardson is currently a M.Sc./Ph.D. in computer science and Engineering Graduate Fellowship recipient at Vanderbilt University. e-mail:william.a.richardson@vanderbilt.edu

Azhar Hasan is currently a M.Sc./Ph.D. in computer science and Engineering Graduate Fellowship recipient at Vanderbilt University. e-mail: azhar.hasan@vanderbilt.edu

Manuscript received May 3, 2023

II. System Design

A. Code base

Code available at this link: https://github.com/Pingumaniac/Blockchain_Voting_System

B. Overview

As can be seen in Figure 1, the front-end contacts a backend node of its choice. In our current extant code, the node forwarding is done by a simple server dispatch service consisting of PHP endpoints, but this could be replaced in future work with dispatching at the front-end level. Upon validation of the request as valid, the backend node will incorporate it into its blockchain and forward the accepted block to the other backend nodes for them to do likewise.

C. Backend Node Design

Each backend is implemented as a Flask node running on Ubuntu and logging data to SQLite. Currently in our design, all other backend nodes are known to one another via a hardcoded list, but a discovery service can be added, albeit centralized. In each backend node, it stores elections, votes, and users within a blockchain, fairly similar to the Bitcoin original scheme [1]. The entire current state of users, votes, and elections is stored at each blockchain block, along with a link (hash) to the previous block, and the transition information that indicates the change from the previous state. This is illustrated in Figure 3.

1) Proof of work: In order for the backend node to show that it has ”skin in the game,” it is forced to spend CPU time in order to create blocks that can be accepted into the new eleciton/voter history. To this end we follow [1] and require that a randomly generated nonce is adjusted until a certain number of zeros at the end of the hash are achieved. This effectively means that a given backend node will always take a fixed amount of time to reply to any creation or update request that involves creating a new block - since in our

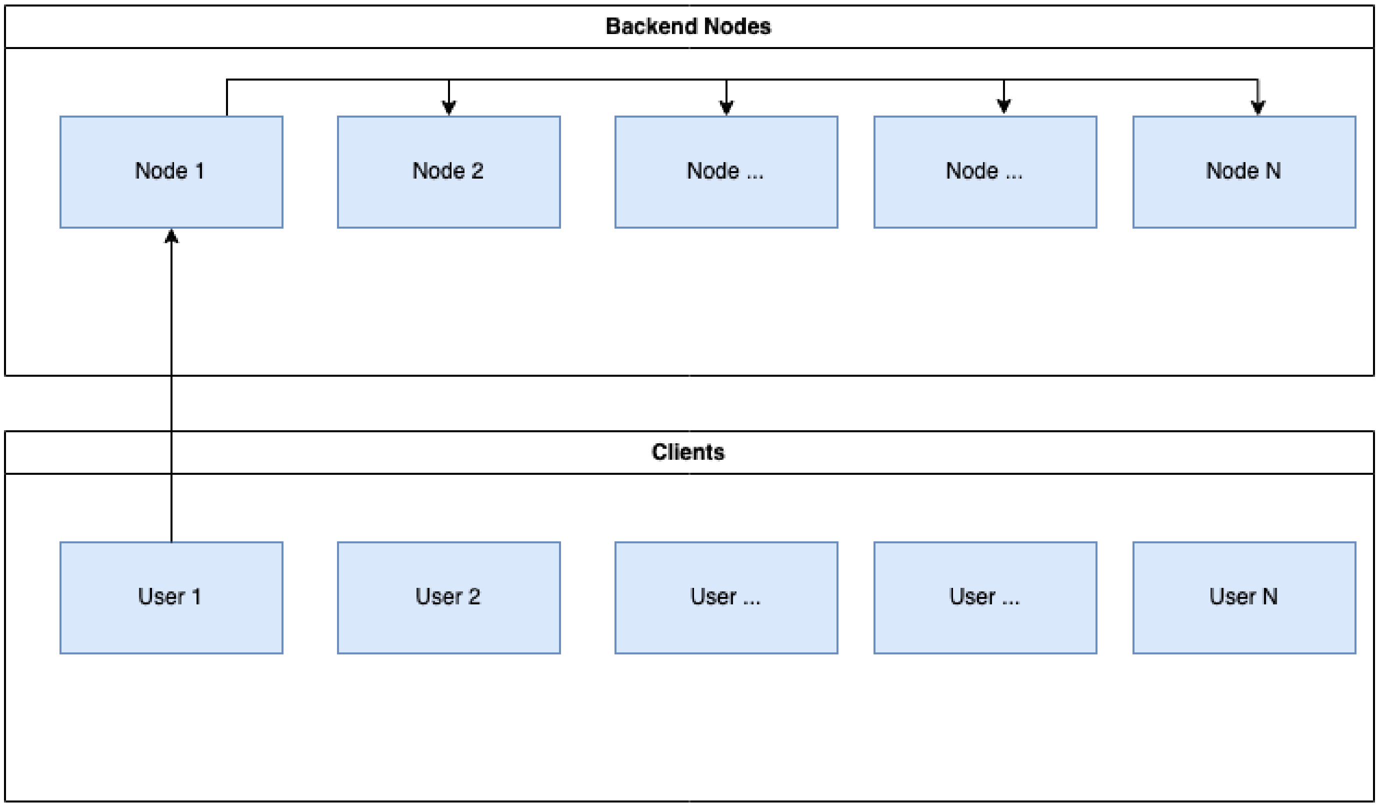


Fig. 1: Overall system design interface. The user nodes each can communicate with an arbitrary backend node to log changes and make requests.

Number of Zeros (BASE64)	Proof of work Time (s)
1	0.0006897449493408203
2	0.0055869221687316895
3	0.07369097471237182
4	1.5643455147743226
5	17.140231049060823

Fig. 2: Time for proof of work to be fulfilled on our test server given a certain number of required zeros at the end of the hash, in BASE64. More zeros means more security, but longer compute times. Average of 30 trials for each setting.

scheme, every election creation, user creation, and vote casting involves creating a new blockchain that contains the updated election/voter state. In Table 2 we show on our extant testing web server the length of time needed to fulfill a proof of work given the number of zeros mandated. As per the obvious, the more zeros that are mandated at the end of the hash, the longer it will take to find a nonce that satisfies this requirements - especially since we naively increment it sequentially in order to find the appropriate hash. We use a mandated 4 unicode zero characters in BASE64 encoding in our current code, but this can be adjusted based on speed and security requirements.

2) Blockchain Vulnerability: In our current scheme, since we only submit a new block for each CRUD request sent from an end user, if there are a sufficient number of transactions occurring using our framework, and the number of legitimate CPUs versus malicious CPUs is high enough, new block mining will quickly bury old transactions under enough blocks such that changing votes or otherwise fabricating history for them will quickly become infeasible. However, if either of these conditions does not hold, history can be fabricated. One potential way to address this would be for legitimate backend nodes to regularly mine new blocks, regardless of the number of transactions occurring.

D. Request Structure

The requests that are sent to the backend nodes, while varying in contents, follow this structure as shown in Figure 4. The user makes a digital signature onto the request using their private key, which is not sent to the server, which can be verified upon arrival. On the front-end side, we use the native Javascript API functions to produce these cryptographic signatures, while on the backend node side, we use pycryptodome to verify these signatures that are sent. These signatures are signed on SHA256 hashes generated from concatenating together the rest of the message content.

E. Verification

All requests to the backend nodes are verified of their user provenance via the RSA digital signature that is associated with every request. For elections and votes, it is verified that respectively the election and the vote is not a double-count (has not already been created).

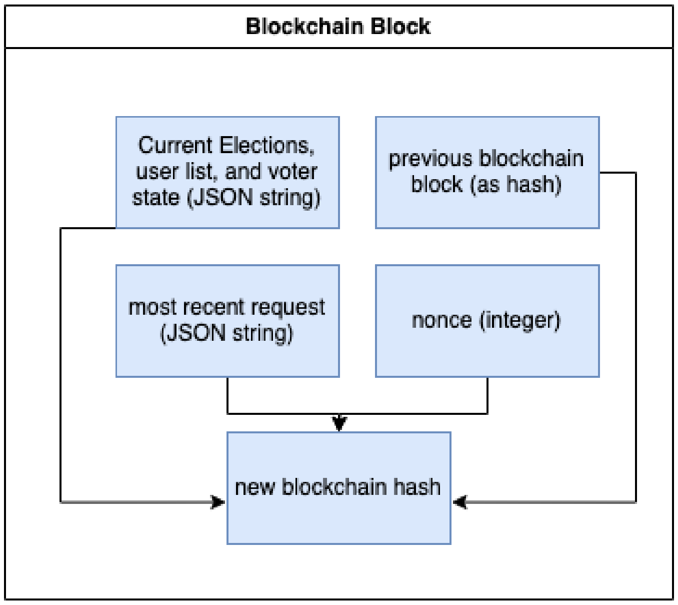

Blockchain Block
Current Elections, user list, and voter state (JSON string)
previous blockchain block (as hash)
most recent request (JSON string)
nonce (integer)
new blockchain hash

Fig. 3: General blockchain block structure. Current global election state is stored in each block along with a link to the past block and the most recent request that triggered this new block.

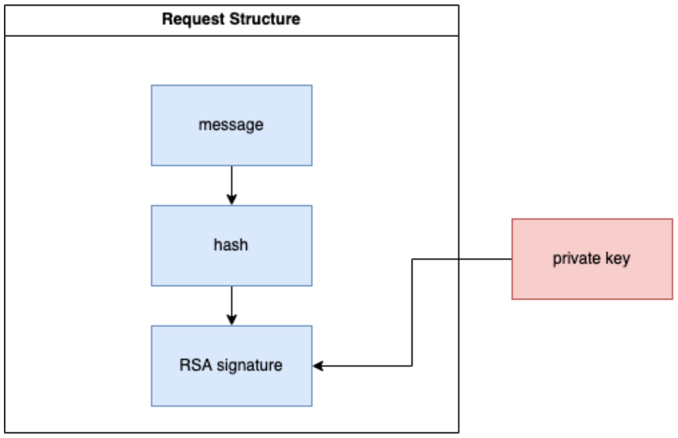


Fig. 4: General Request structure to the backend nodes. Message attributes are concatenated together to produce a hash which is then cryptographically signed via RSA using the user's private key.

III. User Interface and Experience

A. Used Technologies

We use Bootstrap, along with standard HTML/CSS/Javascript. This is served using Python Flask, which provides basic routing capabilities that are similar to Angular.

B. Backend Node Selection

For any requests to and from the backend require one backend node to be selected to receive and send updates to. As is discussed elsewhere, any modifications to users and elections and votes in one backend node are propagated to the other nodes.

C. User Authentication

Users use a specific authentication scheme as follows. Each user has an RSA public-key and private-key pairing. They provide this public key when they register as can be seen in Figure 9. All requests to the backend nodes are digitally signed using the private RSA key, which is never shared with the backend nodes - but they can verify the provenance of the request by verifying the RSA signature.

D. Election Viewing, Creation, and Vote Casting

As can be seen in Figures 5,6,7, the interaction for viewing elections, creating them, and voting in them is relatively simple. Viewing elections is straightforward with a simple button press. For election creation and vote casting, the relevant properties are passed forth, along with a digital signature via the private key of the username in question.

E. User Viewing and Creation

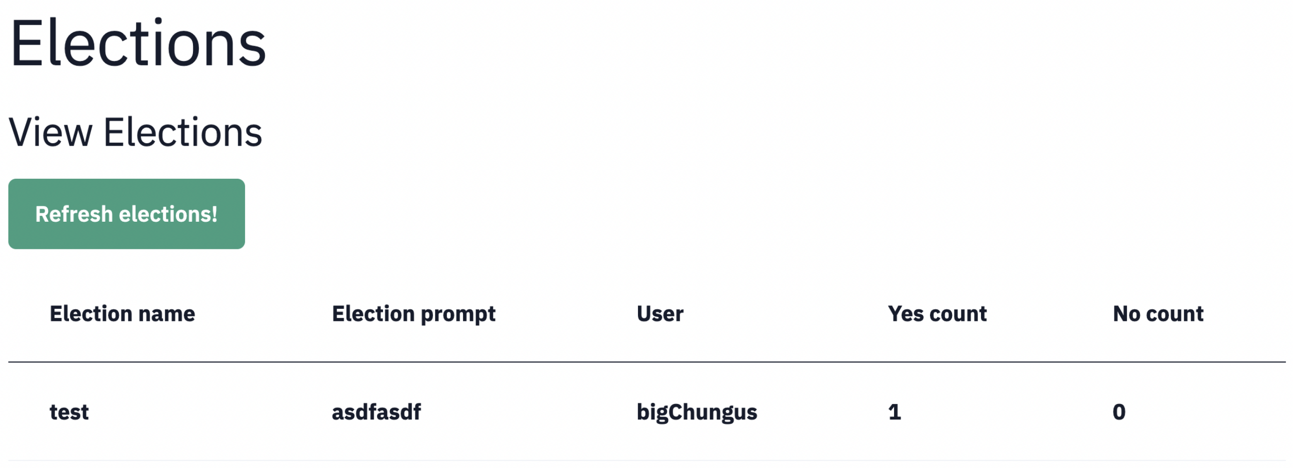
User viewing of course requires no authentication - all users are publically available information. For user creation, all that is needed is for the relevant user to provide their public key to the backend nodes.

IV. Discussion

A. General Limitations and Future Work

There are a few different limitations of this work that could potentially be addressed in future work:

1) Currently security in this scheme is a product of honest backend node CPU capacity, proof of work difficulty, and the number of transactions that come into the backend nodes. If any of these three factors is too low, the security of the system could be compromised. While proof of work difficulty and backend node CPU capacity might not be easily changed, since they are tied to real world resources, we could produce more ”do-nothing” blocks to artificially increase the number of minted blocks at any given time, if the number of votes/elections at any given time prove to be inadequate for security.
2) Unlike in Bitcoin [1], there is no incentive for backend nodes to verify transactions at all. One potential out of this problem is to bestow the ability to vote on elections upon those who mine, but that obviously gives more voting power to those with more CPUs, which might be considered unfair. While we do have a decentralized record-keeping scheme implemented, it is not clear how to incentivize people in real-life to contribute to vote/election verification in the first place.
3) We do not have decentralized discovery in this version of the app - rather, the backend nodes are generated and discovered based on a hard-coded YAML list. This currently defeats the purpose of decentralized nodes, since they are generated and referred to from a central source. P2P communication, perhaps akin to this decentralized P2P blockchain scheme [6], might be capable of addressing this in the future.

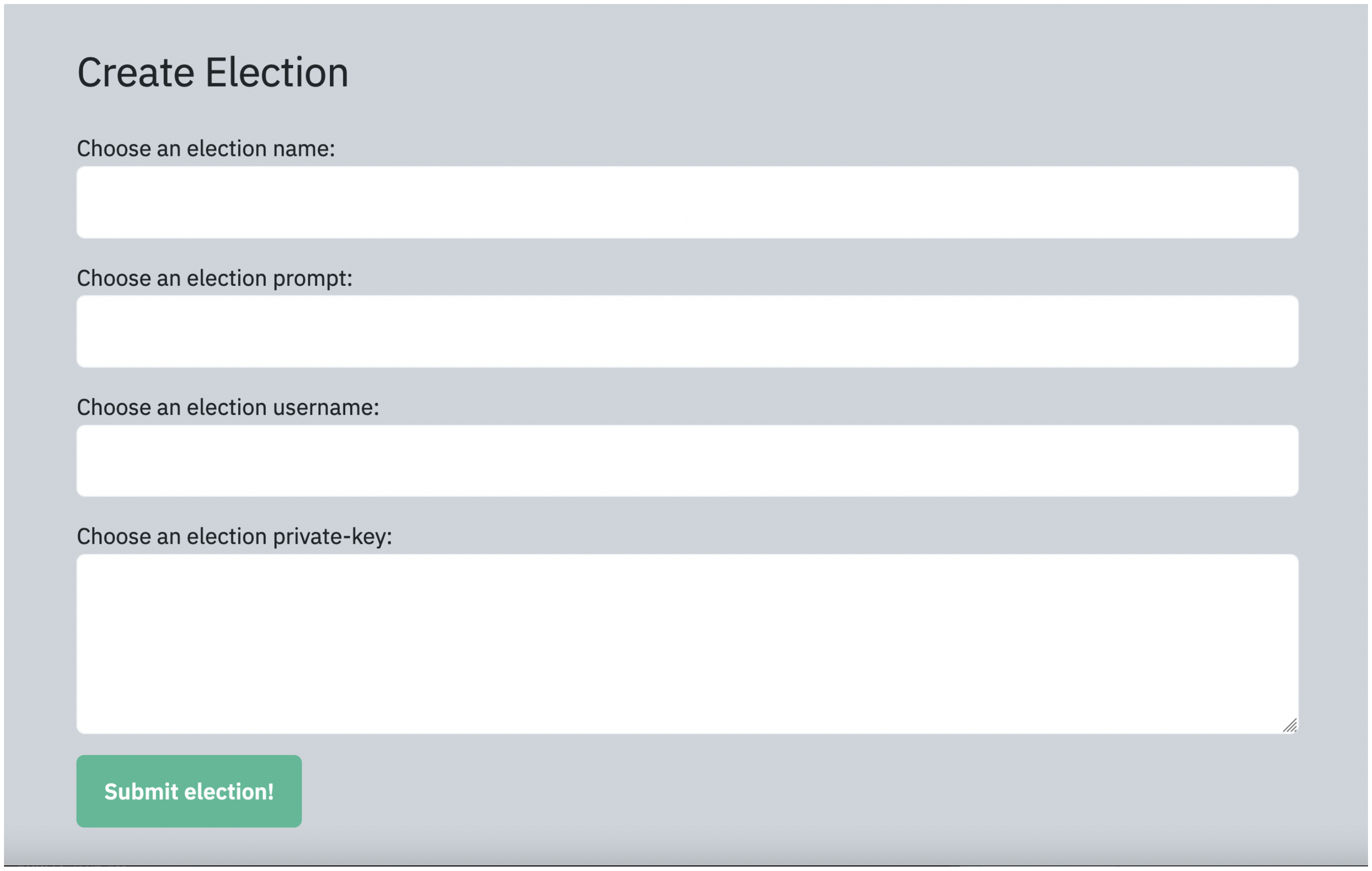


Elections

View Elections

Election name	Election prompt	User	Yes count	No count
test	asdfasdf	bigChungus	1	0

Fig. 5: View election interface. As one can see we have the election name, a prompt, the user who created it, and how many said yes vs. no.



Create Election

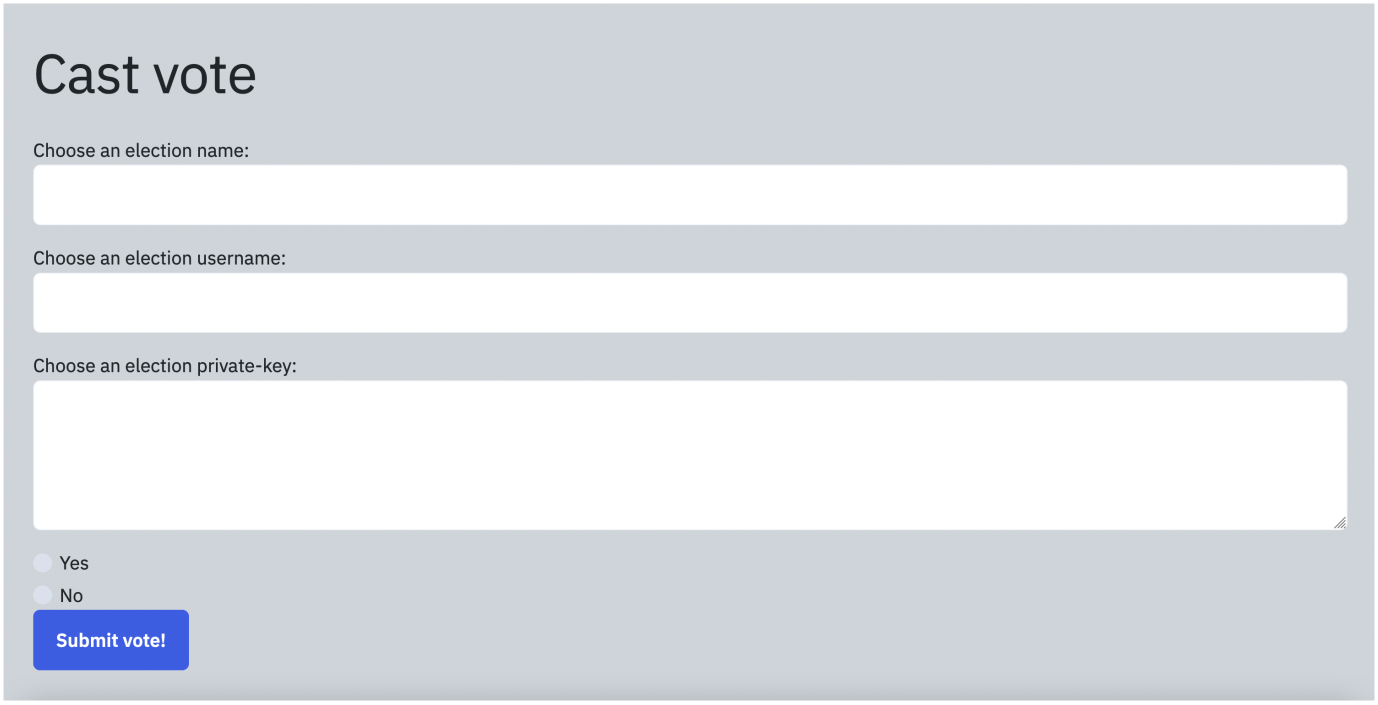
Choose an election name:

Choose an election prompt:

Choose an election username:

Choose an election private-key:

Fig. 6: Election creation interface. Interesting thing to keep in mind is the private-key for the user that they themselves have, that is used to digitally sign the request.


Cast vote

Choose an election name:

Choose an election username:

Choose an election private-key:

Fig. 7: Interface for casting a vote for an election as either yes or no. Note the private key for doing the digital signing.

4) For any election, we only support ”yes or no” answers, rather than multiple choice varieties. This is easy to incorporate into the blockchain, fortunately.
5) There is no secret ballot - all votes are part of the public record. If users are not using pseudonyms, this means that it is explicitly known how everyone voted on everything.

B. Blockchain technology for reliable record keeping - an aside

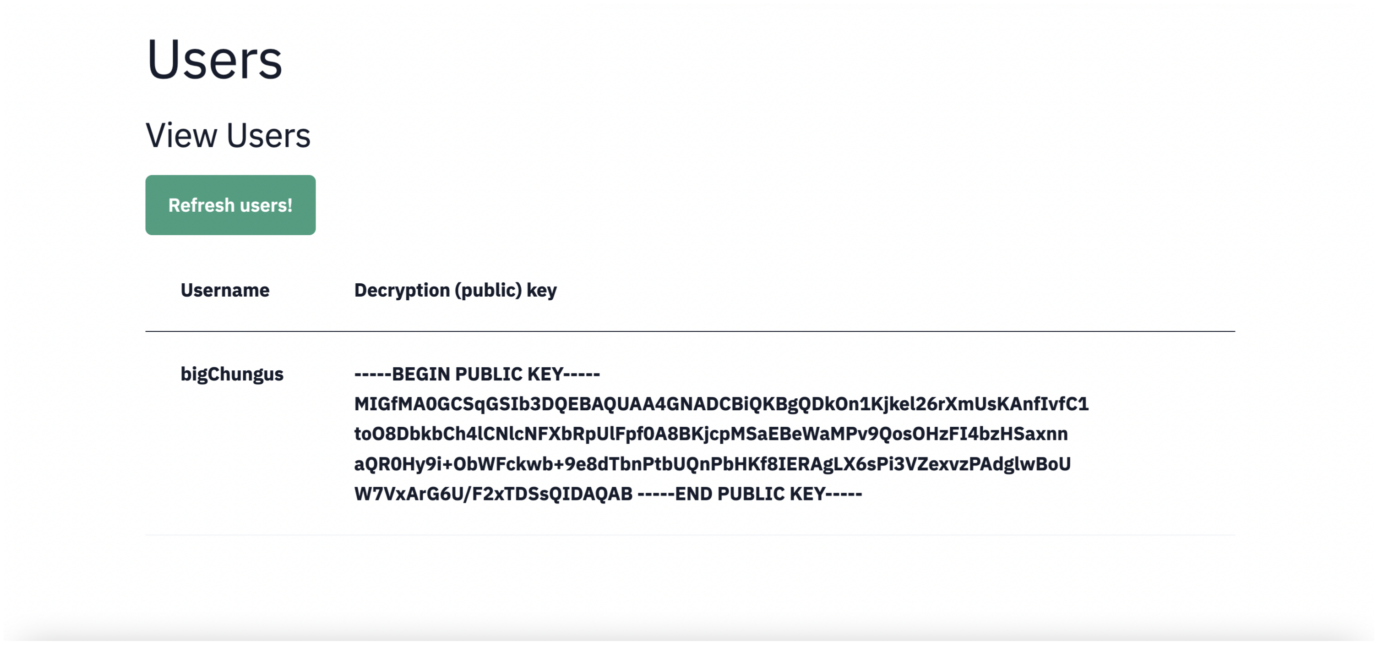
While blockchains are, in the present, capable of maintaining a ”correct” version of a given set of records, it requires continuous computational effort to avoid a new fabricated history from being competitive with the real history. In a way, this suffers from the same problem that all records of history have suffered from - the need for continuous applied effort. This is disappointing, while at the same time illuminating of what is needed to preserve reliable knowledge over time.

V. Conclusion

In this work we presented Simayi, which is a blockchain-based online voting system with decentralized capability for reliable record keeping of elections, votes, and users. While the current system only works as a proof of concept, it oculd conceivably be iterated on to produce a cryptographically secure method of coming to collective consensus on arbitrary topics, and through its proof of work, can have its cryptographic flexibility turned up and down. However, the security of this scheme is contingent on enough ”honest CPUs” and votes/new elections/new users and other such transactions for the record-keeping to remain resistant to tampering.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
[2] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, 2014.
[3] R. Taş and Ö. Ö. Tanrıöver, “A systematic review of challenges and opportunities of blockchain for e-voting,” *Symmetry*, vol. 12, no. 8, p. 1328, 2020.
[4] S. Park, M. Specter, N. Narula, and R. L. Rivest, “Going from bad to worse: from internet voting to blockchain voting,” *Journal of Cybersecurity*, vol. 7, no. 1, p. tyaa025, 2021.
[5] E. Milanov, “The rsa algorithm,” *RSA laboratories*, pp. 1–11, 2009.
[6] K. Khacef and G. Pujolle, “Secure peer-to-peer communication based on blockchain,” in *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33*. Springer, 2019, pp. 662–672.

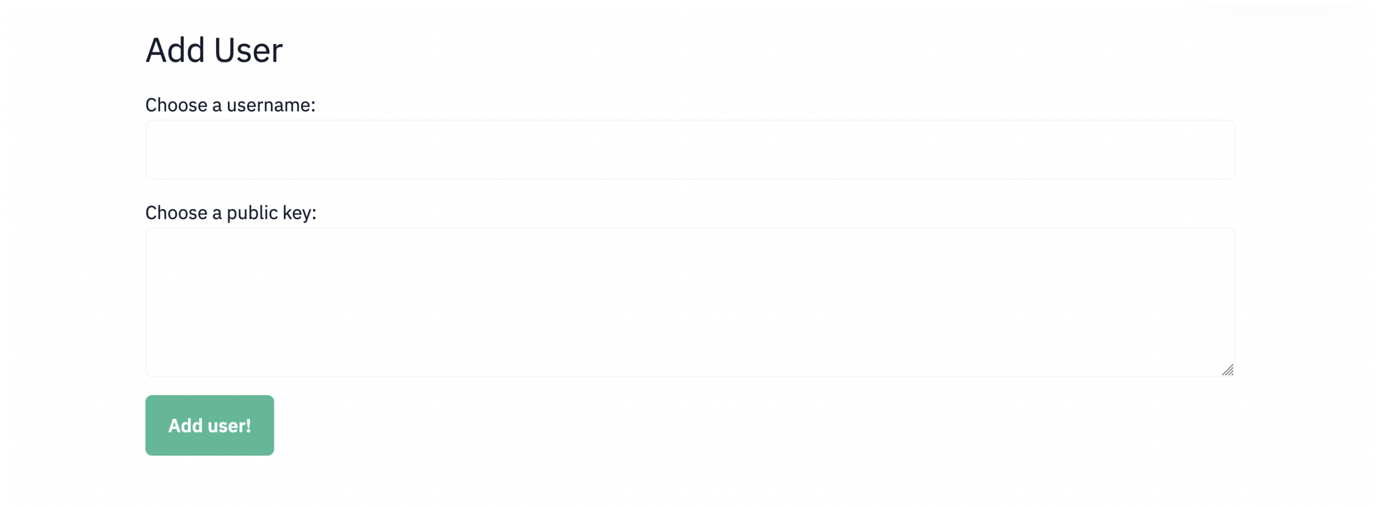


Users

View Users

Username	Decryption (public) key
bigChungus	-----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDkOn1Kjkel26rXmUsKAnfIvfC1 toO8DbkbCh4lCNlcNFXbRpUlFpf0A8BKjcpMSaEBeWaMPv9QosOHzFI4bzHSaxnn aQR0Hy9i+ObWFckwb+9e8dTbnPtbUQnPbHKf8IERAgLX6sPi3VZexvzPAdglwBoU W7VxArG6U/F2xTDSsQIDAQAB -----END PUBLIC KEY-----

Fig. 8: View Users interface.



Add User

Choose a username:

Choose a public key:

Fig. 9: One adds users via this interface. Note the public key that is provided such that requests can be verified as belonging to the user in question.