COSC 5010-08 Fall 2023 Project Report

Michael Elgin

December 3, 2023

Prototype repository: `https://github.com/Mikey-E/COSC-5010-08-Fall2023-Project`

1 Introduction

In the United States, elections are still held by requiring a citizen to vote in one of two ways: either vote in-person, or vote by mail. Each of these alternatives suffers from problems that current technology should be able to solve, elaborated upon in section 3. This project develops a prototype of a new voting system to improve upon these issues, additionally with distributed database synchronization. Section 2 discusses some past attempts to create such a system and highlights some downsides in those attempts and how the particular implementation in this project tries to improve upon them. These downsides and the improvements proposed over them span from security to flexibility of design (of the distributed architecture) to public understandability.

2 Related Work

There are some firms, such as *Voatz*, *Follow My Vote*, and *Votem* [1], which have been working on blockchain-based voting systems. There are some concerns with this. The first is that while blockchains are nice to have, they will not necessarily be needed for voting. Furthermore, while blockchain technology allows people to be pseudonymous as does the proposed design in section 5, it does not force people to register a pseudonym with their real identity except potentially in the case of a permissioned or private blockchain. Another disadvantage is that for a blockchain to be resilient to 51% attacks the network of miners must constitute an extreme amount of processing power, and most of this “work” in the proof-of-work process would go unrewarded since there is no coin to distribute for an election, nor would there be any reason to keep mining for the vast majority of the time since an election only occurs for a brief amount of time once every 2 or 4 or 6 years. For these reasons the design proposed does not use blockchain technology.

In [2], Tejedor-Romero et al. proposed a design based on Galois fields and mixnets. Their approach left some undesirable features, including the fact that only the voter could know their own vote. To improve upon this, the proposed solution here allows votes to be held public yet still linked to a pseudonym so that the voter can confirm their own vote. Similar to [2], this design allows “invidivual verifiability”, in this project’s case meaning the voter can verify the correctness of the election by summing totals from the recorded votes themselves, including seeing their own vote present in the totals.

In [3], Baudron et al. present a system which also makes use of RSA. Their implementation is extremely complex. This makes it somewhat less likely to appreciated and trusted by the voters, which is undesirable. Furthermore, their system creates a hierarchy between the national, regional, and local levels. This does not precisely map on to the American election system, and it creates needless constraints on the way the system can be used. The proposed design in this project seeks to abstract away much of the complexity by using reputable cryptographic protocols and also to maintain flexibility (i.e. absence of hierarchy) in the way an election can be structured.

In [4], a registration system involving establishing a shared AES session key is proposed. A huge risk is if the database of keys is ever leaked, since it contained AES keys for shared communication then every voter could be impersonated by the attackers. The design in the project for this report proposes that asymmetric cryptography is preferable because by keeping only public keys together in a single record (registrations.csv), if those public keys are leaked there is no significant consequence.

3 Problems - Open Issues, Goals - Proposed Solutions

The open problems are as follows. First, voting in-person suffers from congestion and wasted time due to traveling to the voting location and waiting in line. Second, voting by mail suffers from insecurity: anyone who handles the votes can tamper with them or destroy them altogether. Third, having all votes in one place with no replication is a problem because it is a risk for data loss.

This project aims to realize these goals with a new voting system to be developed, as a prototype, which eliminates the opportunity cost from in-person voting, protects votes from tampering with cryptography, and makes sure all databases in the network reflect the same state by replicating the set of votes collected across all databases.

To meet these goals, the following solutions are proposed. This system will allow people to vote in a way that is asynchronous, cryptographically secure, and replicates all votes across all databases in a network of databases that can accept votes from any registered voter originating through a client program. Section 5 describes how these solutions are met in detail.

4 Constraints

First, anyone who has been registered as a voter must be able to vote during the voting period. Second, votes must be able to be sent over an unsecured network (such as the internet) in plaintext, but include a message signature so that the votes cannot be tampered with in transmission and they can be verified upon receipt by the government. A third constraint is that there must be multiple distributed databases that can accept and synchronize vote tallies.

5 Design

It is assumed that the process starts from scratch. This means initially there are no databases, no results, no registrations, and no public or private keys made. The first step is that citizens must register to vote.

5.1 Registration

The process of registration here means that citizens must do the following actions only once: they must authenticate at a government office in-person by showing photo ID. At this time, or before, they must choose a pseudonym[1]and associate a public key with that pseudonym after generating a public-private key pair. When the citizen votes in future elections, this pseudonym may be updated with a vote choice in the database network. The citizen's private key allows them to attach a proper digital signature to their vote. This means that no one else can vote with that citizen's pseudonym and the vote cannot be altered in transmission, because the public key that the government will use to verify the signature will be the one associated with the pseudonym, and no one else has the private key to make the right signature. From time to time the registrations can be randomly sampled to ensure everyone in the sample is real i.e. there have not been fake voters introduced by someone.
The following is an example of the contents of a registration .csv file for the prototype:

```
pseudonym,public_key_file
voter0,public_key_voter0.pem
voter1,public_key_voter1.pem
voter2,public_key_voter2.pem
```

The following is an example of a key .pem file:

```
—–BEGIN PUBLIC KEY—–
MIIBIjANBgkqhkiG9w0BAQEFAA ... JcGwdgXr4GJQ3NaMr+anDZF/V jwIDAQAB
—–END PUBLIC KEY—–
```

5.2 Database

Next, the databases are initialized as separate processes that run in parallel. They know about each other since they keep track of what neighbors they have, which is all other databases since a fully connected topology of the network is required.

```
start cmd /c "py database.py vDB1 127.0.0.1 8081 8084 20 127.0.0.1:8082:8085 127.0.0.1:8083:8086 > results1.txt"
start cmd /c "py database.py vDB2 127.0.0.1 8082 8085 20 127.0.0.1:8081:8084 127.0.0.1:8083:8086 > results2.txt"
start cmd /c "py database.py vDB3 127.0.0.1 8083 8086 20 127.0.0.1:8081:8084 127.0.0.1:8082:8085 > results3.txt"
```

The first argument to database.py is the name of that particular votes database. The second is its IP address. The third is the port that the database *listens* on for any incoming connections. The fourth is the port that the database uses when acting as a client to synchronize votes with its neighbors, which each database uses to recognize a neighbor that is trying to synchronize votes. The fifth is the timeout in seconds, meaning how much time can pass with no votes before the database stops accepting new votes (and in this prototype waiting for such a length of time to pass is considered the end of the election).

[1] This means a fake name, but one that still uniquely identifies them.

The remaining arguments are each neighbor's IP address, server port, and client port in the format address:server_port:client_port. Figure 1 shows an example of how votes synchronize across the network.

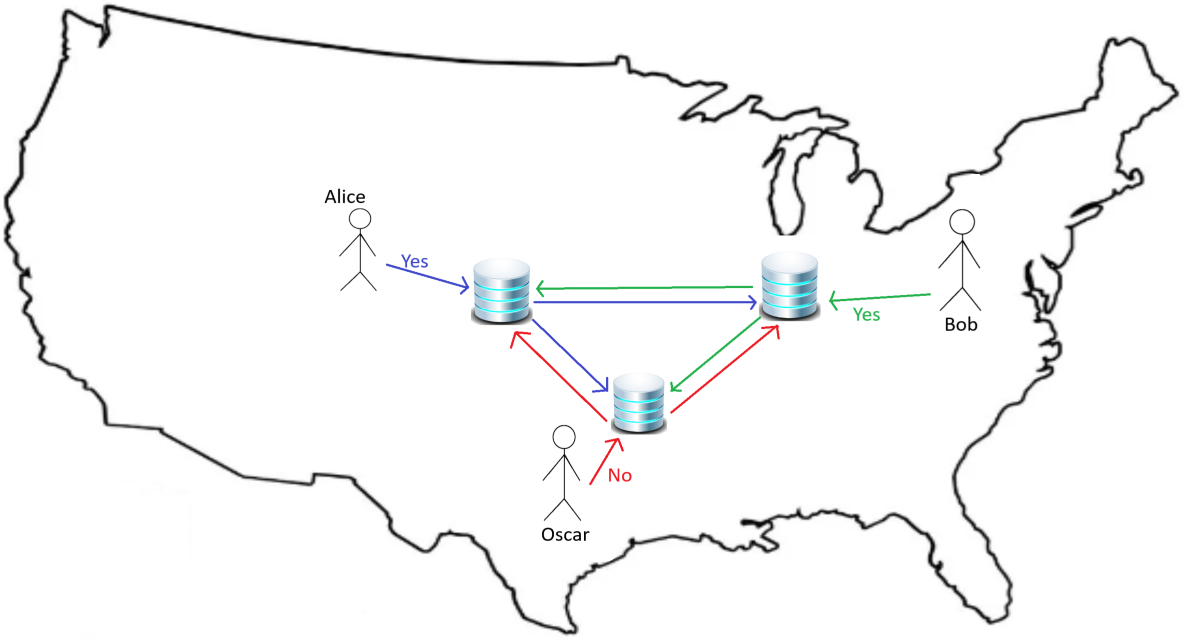

Figure 1: Replication Visualization

Table 1 shows the schema.

Table 1: Schema

Variable	Type
pseudonym	TEXT
choice	TEXT

Upon startup, each database will create a table of the schema, and insert a record for every pseudonym with a null vote. This is so that in a final vote tally, the amount of registered voters who did not vote can also be seen.

```
cursor.execute(”CREATE TABLE votes(pseudonym text, choice text)”)
for _, pseudonym in self.registrations['pseudonym'].items():
    cursor.execute(”INSERT INTO votes VALUES ('” + pseudonym + ”', 'null')”)
```

5.2.1 Distributed Synchronization

The synchronization follows a custom implementation of master-master replication. When a vote arrives to the database, if it came from a neighbor's IP address and client port, the database knows that this is a vote for synchronization, and will not forward it further because it knows the other databases are getting it as well. If it comes from any other IP address and source, it must be from a voter so it must be synchronized.

```
if ip_port not in ip_port_neighbors: #[(item[0], item[2]) for item in self.neighbors]:
    self.sync_vote(vote)
```

In both of those cases, the voter's record will be updated with their choice.

```
cursor.execute("UPDATE␣votes␣SET␣choice␣=␣'" + choice.decode() + "'␣WHERE␣pseudonym␣=␣'" + pseudonym.decode() + "'")
```

As part of this process, the database will verify the signature. If it does not match what is expected for the registered pseudonym and public key, the vote will be rejected. At the end of the election, the database will display its results including sum totals. This is so that everyone can see each pseudonym and know that their vote was recorded correctly, as well as verify the totals for themselves.

5.3 Client

The process of a client (voter) sending a vote involves them specifying their pseudonym, again for which they are the only one that has the private key for the corresponding public key that the vote message signature will test against. They then specify their private key file, their choice, and the IP and server port of the database they wish to vote in, which can be any one since the vote will synchronize across all databases.

5.4 Cryptography

The encryption is based on RSA. The hash function used is SHA256. To create the signature, then message is hashed, then the hash is signed with the private key. To verify the signature, the message is hashed again, and then this hash is compared to the signature that has been unsigned with the public key. The following is an overview of the client constructing message signature and attaching it to the vote.

```
vote = (self.pseudonym + delimeter + self.choice).encode()
signature = self.private\_key.sign(
    vote,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt\_length=padding.PSS.MAX\_LENGTH
    ),
    hashes.SHA256()
)
vote = vote + delimeter.encode() + signature
```

The following is an example of the database verifying a vote by it's signature.

```
sections = vote.split(delimeter.encode())
pseudonym = sections[0]
choice = sections[1]
signature = sections[2]
...
public_key.verify(#will throw an InvalidSignatureException if message or signature has been altered.
    signature,
    pseudonym + delimeter.encode() + choice,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
```

6 Conclusion - Results From Prototype

An example of the results from a simulated election follows. This shows a log of the events a database experienced during the voting period, as well as the final votes and totals. During this time, the database first receives a vote from a non-neighbor (a client), so it synchronizes it with its neighbors. Additionally, it receives a vote from each of its neighbors since its neighbors are synchronizing their own votes with it, so it knows those are votes it does not need to foward onward. By displaying every vote choice next to the pseudonym of the voter, everyone can see the votes themselves to confirm that their vote is present and to confirm that the totals are as expected. For a real run of the system, 127.0.0.1 would of course be replaced by the real IP addresses of the databases.

```
Received vote from ('127.0.0.1', 65380)
Synchronizing vote with neighbor: ('127.0.0.1', 8082, 8085)
Synchronizing vote with neighbor: ('127.0.0.1', 8083, 8086)
Vote recorded: voter0, yes
Received vote from ('127.0.0.1', 8085), vote is from a recognized neighbor so it will not be forwarded.
Vote recorded: voter1, yes
Received vote from ('127.0.0.1', 8086), vote is from a recognized neighbor so it will not be forwarded.
Vote recorded: voter2, no
Timed out after 20.0 seconds
———-Votes————
pseudonym: voter0, choice: yes
pseudonym: voter1, choice: yes
pseudonym: voter2, choice: no
———-Totals————
yes: 2
no: 1
```

References

[1] S. Daley, *Blockchain voting: The future of elections?* `https://builtin.com/blockchain/blockchain-voting-future-elections`, Updated by Brennan Whitfield on September 21, 2022, 2022.

[2] M. Tejedor-Romero, D. Orden, I. Marsa-Maestre, J. Junquera-Sanchez, and J. M. Gimenez-Guzman, "Distributed remote e-voting system based on shamir's secret sharing scheme," *Electronics*, vol. 10, no. 24, 2021, ISSN: 2079-9292. DOI: `10.3390/electronics10243075`. [Online]. Available: `https://www.mdpi.com/2079-9292/10/24/3075`.

[3] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, "Practical multi-candidate election system," in *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 2001. DOI: `10.1145/383962.384044`. [Online]. Available: `https://people.csail.mit.edu/rivest/voting/papers/BaudronFouquePointchevalPoupardStern-PracticalMultiCandidateElectionSystem.pdf`.

[4] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," Cornell University, Tech. Rep., 2008. [Online]. Available: `https://www.cs.cornell.edu/andru/papers/civitas-tr.pdf`.