RU eVoting

Gabriel Ajram
Erik Castro
Harsha Dantuluri
Apoorva Goel
Hrishit Joshi
Nirav Patel
Krishna Prajapati
Dominik Sepko
Osama Trabelsi
Mufeng Zhu

Big Voting Project Assignment
14:332:424: Information and Network Security [Fall 2020] - Dr. Wade Trappe
November 22, 2020

Introduction:

Due to the drastic improvements in technology in recent years, as well as the current pandemic situation, we've seen a huge increase in electronic usage. People are realizing that a lot of jobs and tasks do not require people to be face to face. As time passes, companies, organizations, and individuals alike are trying to take advantage of our current global situation to increase their productivity and efficiency. We are ditching obsolete, inefficient, traditional methods for faster, more accurate, modern methods.

The United States 2020 Presidential Election displays a prime example of a system that is not optimized for the modern age. We've seen very clearly how dangerous and troublesome in-person voting is. Additionally, we have noted the security flaws and inefficiencies of mail-in ballots. We have also seen the amount of time it takes for a result to be declared, as what had normally been an election day in previous years had easily turned into an election week.

Such problems occur not only with large scale elections such as the presidential election, but it also occurs with smaller scale, democratic-style decision making. Organizations and corporations can also utilize a more efficient method of voting for decisions, as they experience similar inefficiencies.

Thus, to solve these problems, our group proposes a form of digital voting scheme that not only solves the issues of in-person voting, but also upholds the security and integrity of any voting process.

Digital voting, when done properly, can be more efficient than the traditional ways of voting both in terms of speed of the process and the financial aspect. Counting ballots will be significantly faster online than doing it by hand or with a scanner, and election departments will require less man-power to manage the voting and counting process. Digital voting also can improve accessibility for disabled votes as they will be able to vote at home or any public disability friendly institute. All these advantages increase the voting turnout, especially as we reduce the digital divide. This will boost the strength of our democracy, as well as the efficiencies of our organizations.

However, virtualizing a traditionally in-person scheme comes with its own set of challenges and requirements. In this outline, we will discuss the old features of in-person voting that need to be maintained, the new challenges of digital voting that need to be addressed, and the approach towards solving these problems and creating a successful system.

Core Features & Requirements:

Username and Password with Government ID

The first requirement that comes to mind with a voting system is authentication (of the voter). We need a way to make sure that the person voting is who they claim they are in real life, and not impersonating someone else.

Each voter will pick a unique username and password upon registering. Registration involves signing up at one's local state website. In order to verify a person's identity, authentication requires either a government issued social security card or a valid driver's license for the state in which the voter wishes to register in. When the identification is verified, the voter creates a username and password that is unique to them, where the password must have at least 1 number, 1 symbol, and at least 1 uppercase letter.

Privacy & Security of Votes

Each vote is encrypted and sent to the polling server. No outside person will be able to intercept and decrypt the message, meaning that no one will be able to figure out who voted, or what they voted for. Only the polling server will have that information.

The polling server stores the voter's ballot information in their system, and encrypts this with their public key. In this manner, internal server workers cannot view or change votes easily, and if this information is stolen, there is no feasible way to decipher it.

Voter Authorization and Integrity

The polling system contains a list of all registered voters, along with the username and password of that voter. When a voter is trying to log in, the username is matched up with the username entered, which is then matched with the password entered. If both of these are the same, then the system lets the voter log in. If these do not match, then the system will not let the voter log in until the username and password match those that are on file.

If the user is able to log in, the system will also check to see if the user has already voted. If the user has already voted before, we will not allow the user to change their vote, nor vote again.

Candidate Visibility

To ensure a fair vote, no candidate or option should be left out. By having the polling server fetch and send candidate information from official sources, we protect against outside interferences that could change the visibility of each candidate.

Security Tools:

Option 1: public key crypto for authentication and encryption (which keys are being used for authentication and for encryption? → *server will have to sets of public/private keys*

Option 2: Using AES eliminates this problem. AES for encryption. AES is the most standard and widely used method for block cyphers.

Our group decided to use **option 1** due to the simple implementation and ease of use. Option 2 requires the server and client to securely come to an agreement on keys. This also means each client will have a different key.

Tools Needed:

PyCryptodome -- RSA package
https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html
For both confidentiality (encryption) and authentication (digital signatures).

hashlib
https://docs.python.org/3/library/hashlib.html
For securing hashes. (E.G. in the password storage implementation)

PyCryptodome -- Signature package
https://pycryptodome.readthedocs.io/en/latest/src/signature/signature.html
Algorithms for security/non-repudiation.

PyCryptodome -- SHA3-256
https://pycryptodome.readthedocs.io/en/latest/src/hash/sha3_256.html
Hash function on binary data that produces a 256 bit digest.

Design Outline:

We will have a server written in python to verify the identity of the users casting a vote, receiving votes, and counting the total votes. The users will have a username and password to verify their identity, which will be sent to the server which will check if the user is eligible to vote by checking their credentials saved within a hash chain. Once verified, the next step is to check applicability, meaning to check whether the voter is allowed to vote in the respective state/city/location by going through the whitelist of users saved within another hash chain. If the user is eligible, they will be prompted by a GUI/form of the candidates to choose and cast their votes. The votes will be collected and stored with the voters information (encrypted) to keep confidentiality. There will be a command to send to the server to retrieve the results of the election with the amount of votes for each candidate.

<u>Encryptions, Hashes, Timestamps, and Storage</u>

The server uses two sets of RSA public/private keys

- RSA public/private confidential key
    - Used only for encrypting votes and ballots sent to server
- RSA public/private authorization key
    - Used for all other server-client communication

Hashing Function

- Anytime "**hashing**" is mentioned, we will be referring to **SHA3-256**

Timestamp

- All messages will have a timestamp prepended to them to prevent replay attacks (not explicitly mentioned from here on)

Database

- The server will have 2 big files with encrypted data with new lines separating entries
    - One file will represent the list of users
    - The other will represent the list of voters
- While the server is running, the data will be stored on the program as a hash chain

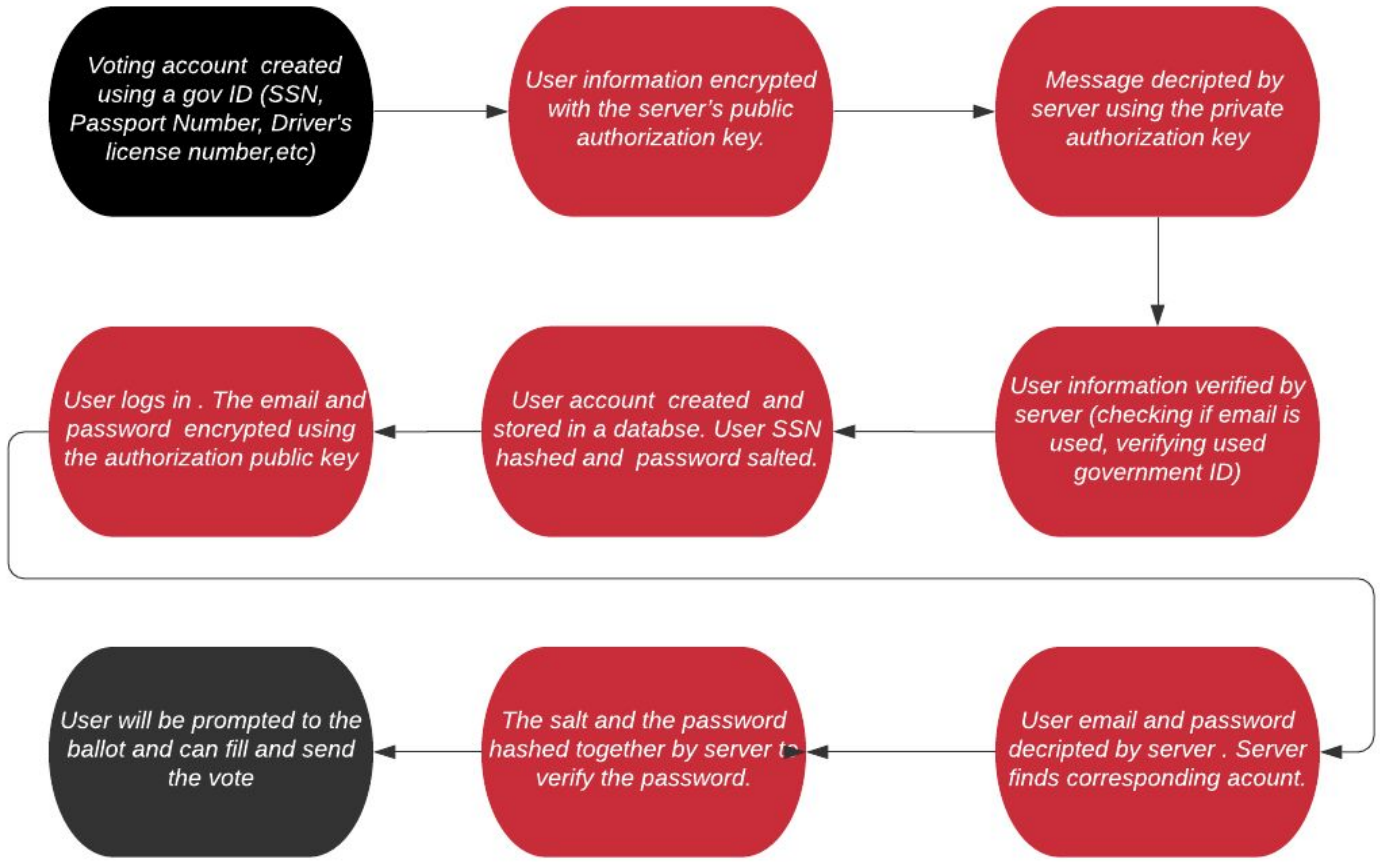
Username and Password with Government ID

We let the user pick their username and password, and fill out their personal information. The username will be their email, which we will ask to verify later. The user's password will be checked on the client-side to make sure that it follows the security criteria. Finally, the user will also provide some form of government identification, such as SSN. The client would encrypt this information with the server's public authorization key before sending it to the server.

Upon receiving this information, the server will decrypt the message using the private authorization key. It will first make sure that the email address has not already been used by searching through the database of users. It will then send a message to a government site to ask for confirmation whether this person's SSN matches who they claim they are. If they are, then we will make sure that this person does not already have an account with the same SSN under a different email. If all of these checks are passed, then we can create the account for the user and store it.

We store the account information in the following manner. We keep the user's personal information, such as their name and a hashed SSN, to make sure that future accounts are not actually duplicates of this. We keep the user's email as well. Then, we will create a random "salt" for the password and store this. Finally, we will append the salt to the end of the password, and hash it together. We will store the hash as the user's salted password. All of this information will be stored in the database.

Once the user's information is verified and stored, the user can then log in and vote. When a user logs in, their email and password will be encrypted using the authorization public key, and then sent to the server. This is decrypted by the server, which then finds the account that corresponds to the email. The server will then hash the salt and the password together to see if it matches the stored value. If the hashes match, then the user will be prompted with their ballot, and can send in their vote.



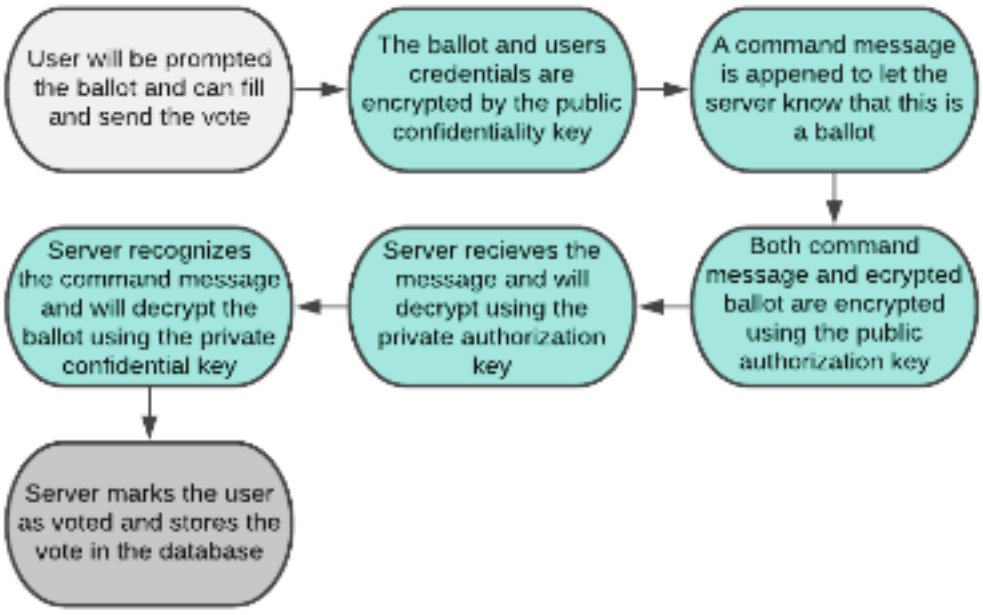
Username and Password with Government ID Process flowchart

Privacy & Security of Votes

Once a user has logged in, the server will let the user know if they have voted already or not. If not, the user has access to see the whole ballot and can select their vote. This vote is then sent together with the user's email and password, which should have been cached on the client side. The log-in and vote together are encrypted using the public confidential key. This message is then appended to a command message that lets the server know that this is a confidential ballot. The command and the encrypted ballot are then encrypted using the public authorization key.

When the server receives the message, it will decrypt using the private authorization key. It will note the command and understand that the encrypted message is a ballot. The server will decrypt the ballot using the private confidential key. It will once again find the user and mark that they have already voted, and then the server notifies the user that their vote has been counted.

The server then stores the vote in the database(s). The vote and the hashed SSN of the user are encrypted together using the public confidential key. This way, the server can identify which users have voted without an outsider being able to tamper or view the ballots, since they do not have the private key. Encrypting the vote and hashed SSN together also prevents identifying which voter voted for which candidate.

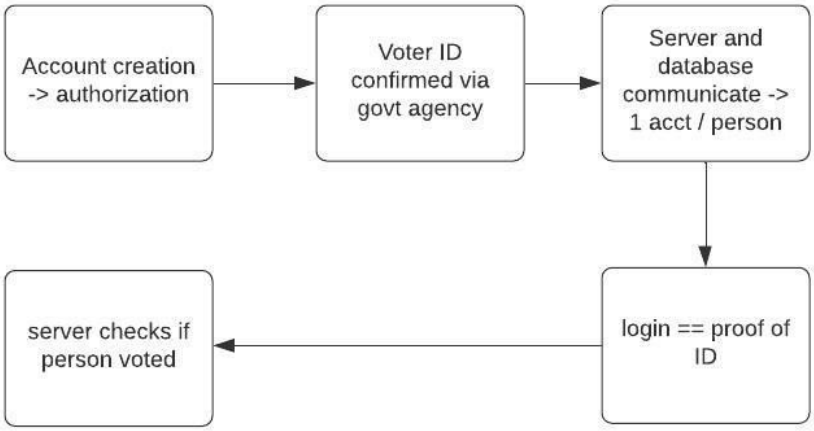


Voter Authorization and Integrity

As mentioned previously, the voter is authorized when they create the account. The voter's identity is confirmed through a government agency. The server checks through the database to see if the person already has an existing account, which ensures that each voter can only have 1 account.

We use the voter's log-in as proof of their identity, since only they should know their email and password combination. Therefore, we can confirm that every action that has a proper log-in was an action done by that user.

Finally, the server checks to see if the person has already voted, and does not send any ballot information or accept new ballots from this user if they have. This ensures 1 vote per voter.



Candidate Visibility

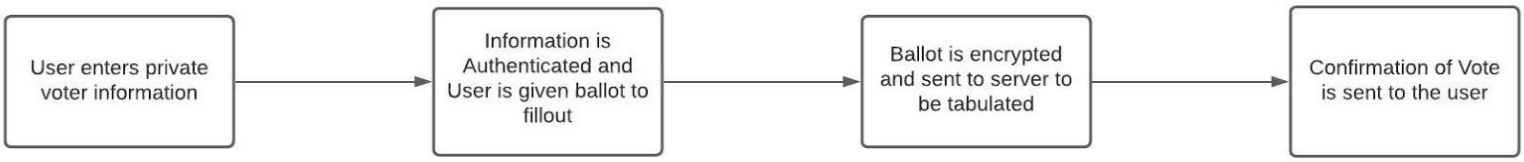
Ballots will be pulled from an official government source when it needs to be sent to a user. This information is only sent to a user who has not voted yet and can successfully log in. This information is encrypted by the server using the private authorization key. The client will decipher this information using the public authorization key.

Since the candidate information is public information, we are not concerned with someone intercepting this message. While they can decrypt with the public key, they cannot change the information that is being sent to the client. Additionally, by encrypting with the private key, the client has the server's digital signature and can confirm that the candidate information is coming directly from the server.

Counting Votes

The server will go through the poll database and decrypt the information using the private confidential key. It will simply tally each vote, and then display the results.

Overall Flowchart from the User's Perspective:



Work Plan:

Socket/Security Tools/Set-up-
Krishna Prajapati
Erik Castro

Create/Read RSA Keys-
Osama Trabesli
Dom Sepko

Create/Write Users & Votes to Database-
Harsha Dantuluri
Mufeng Zhu

Client-side User/Pass/SSN -
Harsha Dantuluri
Gabriel Ajram

Server-side User/Pass/SSN -
Krishna Prajapati
Apoorva Goel

Read Votes & Count-
Nirav Patel
Hrishit Joshi

Sending Candidate Info & Sending Encrypted Vote-
Apoorva Goel
Gabriel Ajram

REFERENCES:

Introduction to Cryptography with Coding Theory, by Trappe and Washington, Prentice Hall, 3rd edition.
https://en.wikipedia.org/wiki/Salt_(cryptography) , 2020