CSCI531 SEMESTER PROJECT

Designing a Secure Electronic Voting System

Pooja Rajan

- Overall Description

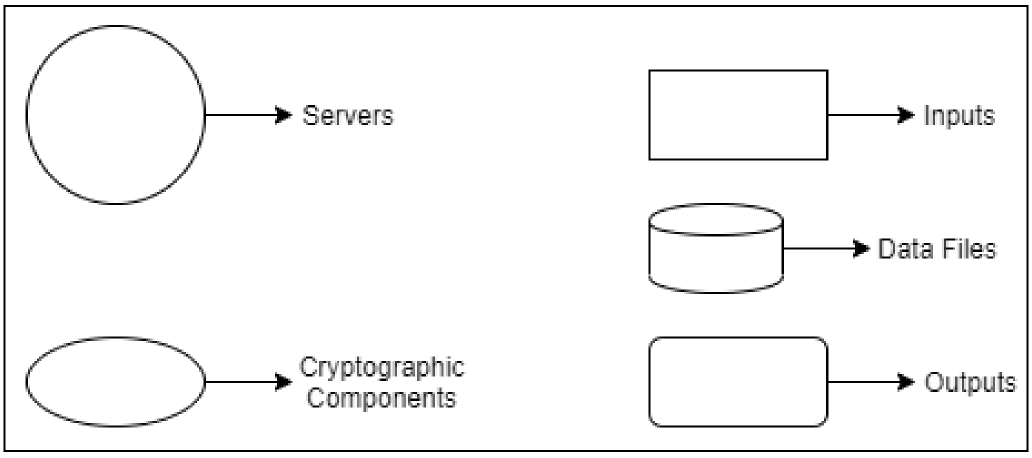
  Secure E-Voting System is a python based project. It handles the security of data during the registration of the voters while polling on election day, and counting and auditing of the votes to ensure an unbiased voting environment. It also ensures that the voter is a registered and unique voter who is eligible to vote and its ballot for voting is personalized and secured.

- Proposed System Architecture

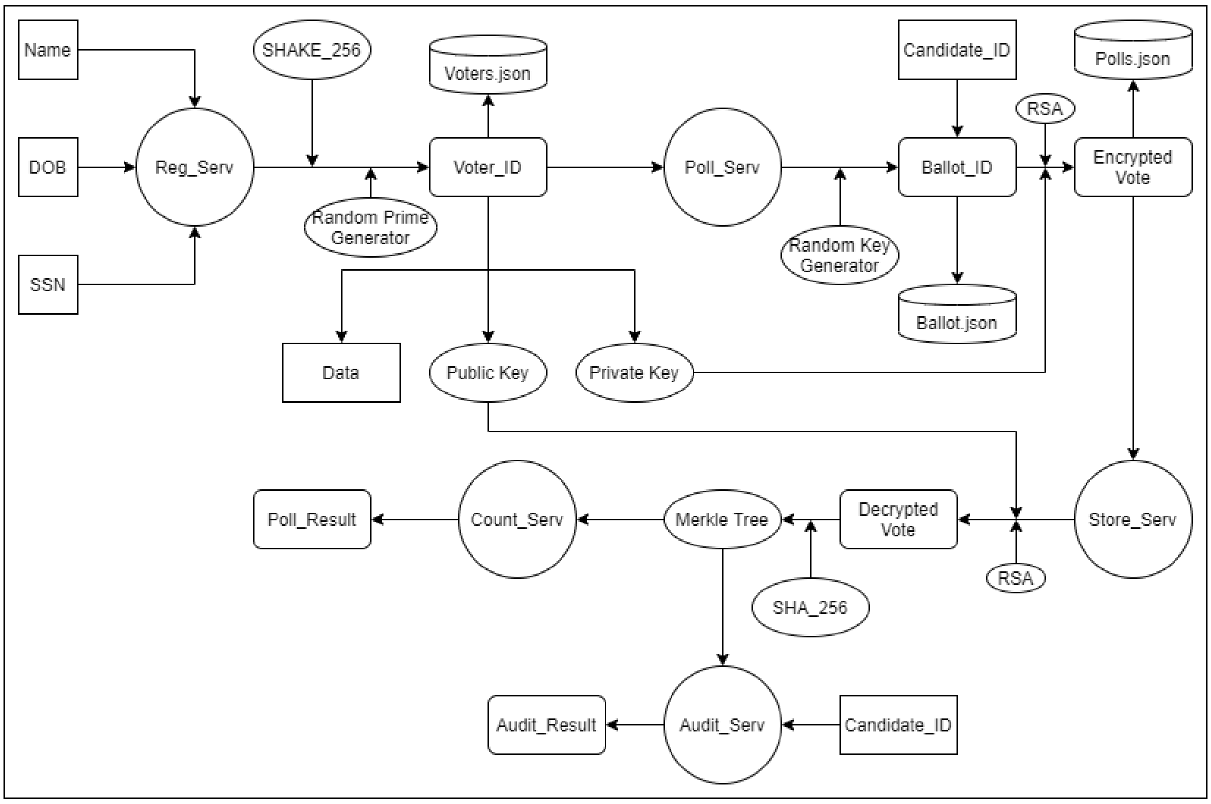
  The proposed architecture consists of five components:-

  - Registration Server - For registration of the voter and key generation to ensure confidentiality of the voter while or after voting.
  - Poll Server - For the creation of a unique ballot for each voter upon confirmation and to cast their votes securely and secretly.
  - Store Server - To authenticate the ballots passed after polling, and storing and managing the votes from all the anonymous(registered) voters.
  - Count Server - To count the votes for each candidate.
  - Audit Server - To conduct an audit to ensure that the votes counted are from unbiased voting methods only.

A diagram has been included to further understand the working of this system.



Index for System Architecture



System Architecture of the System

A comprehensive working of each component, the data flow between them and other functionalities are described below:-

- Registration Server - In this module, a voter, using their SSN number or DL number, registers itself on the Voters List. An SSN or DL number is used to uniquely verify each voter and avoid any duplicity due to a change in location. Upon collecting details from the voter, a unique 10-digit hash is generated for the voter which acts as the voter's unique Voter ID. In case the voter tries to re-register itself, its already existing ID is retrieved and displayed. This voter ID is generated using SHAKE-256. The voter ID, along with other data, is also stored in a file for future tallying and verification. Also, while registering, a voter's year of birth is also used to verify that he/she is eligible to cast a vote. The default minimum age to cast a vote is presumed as 21 years. Along with ID, a public and private key is also generated for secure voting on the election day. The public key is passed on to the Election Committee.
- Poll Server - In this, a voter uses their voter ID for verification process that they are registered voters by tallying up against the voters' file.

When this verification is successful, the voter receives a random unique ballot that has its own ID which is stored in a separate file when a voting has been done using it. Using this ballot ID and the corresponding private key from the previous module, the voter casts a vote which will be encrypted using the private key. This way no one can view or change the vote in the ballot unless they have the voters' public key which is available only with the Election Committee. After the voting is done, both the ballot ID and the encrypted vote is sent to Store Server for authenticating and storing the data.

- Store Server - The data from the Poll Server is read and checked for any duplicity or tampering with the votes by checking it against the ballot file which has a list of all the voters who have voted. Once verified, the encrypted votes are decrypted using the public keys from the Election Committee. The ballot ID serves the purpose of identifying the voter ID to use the corresponding public keys for decryption. The votes are basically candidate IDs, which are then hashed again using SHA-256 hash algorithm and the binary hashes are stored in merkle tree. This merkle tree is then sent to the Election Committee.
- Count Server - The merkle tree is used to count the votes for each candidate. It also checks for any tampering with the votes. Any change in the votes will change the sister nodes and root of the merkle tree and hence will be alerted to the system, that there has been a tampering with the votes.
- Audit Server - This component checks for the consistency of votes received. This way we get to know if any new vote has been added or if any existing votes have been changed. The duplicity of voters is prevented during the registration itself. The uniqueness of the ballot used for voting is also achieved during the ballot generation.

➢ Cryptographic Components Used

The encryption schemes used are:-

- RSA Encryption
- Merkle Tree Hashing

The key management is done using:-

- SHAKE_256 hash algorithm
- SHA-256 hash algorithm
- Private key and Public key using GCD and Modular Inverse
- Random key generator.

Using voter data, SHAKE_256, a SHA3 + Keccak-256 combined hashing algorithm is used for generating a hash. SHAKE_256 is used because it has a somewhat better performance rate as compared to SHA-256. It also allows variable length of the hash key produced. This way the length extension attack can also be avoided. Also, it is easier for any voter to remember and use it easily. It also takes more hash bits, which means it can resist collision attacks for longer periods. SHAKE_256 is also known to be useful for OAEP padding which can be used with RSA Encryption.

The public and private key for each voter is used during voting, for encrypting the votes using RSA. The private key of the voter is used to encrypt the vote casted by the voter in his unique ballot. The votes are encrypted so as to avoid any tampering with the votes while sending and storing the votes. The Election Committee can decrypt the votes using the public key of the voter to collect all the votes for each candidate and then count the votes to announce the results later after auditing. RSA is known for sending sensitive data securely over insecure networks too because of its large prime number generation for GCD and Modular Inversions. A random key generator is used for producing the ballot IDs.

After the votes are decrypted, it is hashed again using SHA-256 to create a hashed leaf node for merkle tree, which is then sent for counting and auditing by the Election Committee. SHA-256 is used to generate a unique 32-byte hash of votes, to verify data integrity and any unintentional corruption in the votes stored. Although it has better security against collision attacks, it is still vulnerable to length extension attacks. But it still wins as one of the best secure hashing algorithms. The hashed leaf nodes produce sister hashes which then traverses up till the root in a bottom-up approach. Any change in the votes reflects a change in the hash data and hence the root is also changed. This way we can check for any discrepancies in the votes after the casting of votes also.

- Packages and Modules Used

  The packages used are datetime, hashlib, helpers, json, mymerkle, os, pickle, random, and sys. The modules used are argv[], dumps(), encode(), hexdigest(), MerkleTools(), now(), pow(), randrange(), sha256(), shake_256(), and urandom().

  - datetime - used for getting date and/or time
    - now() - to get the current date.
  - hashlib - used for hashing purposes.
    - hexdigest() - to return data in hexadecimal format.
    - sha256() - for producing hash key of a data using SHA-256.
    - shake_256() - for producing hash key of a data using SHAKE-256.
  - helpers - to perform read and write operations on a JSON file.
    - update_file - to read and write data according to r and w respectively. Also checks for duplicate records.
  - json - for reading and writing of simple data in a JSON file.
    - dumps() - to write data in a .json file.
    - load() - to read data from a .json file.
  - mymerkle - An external file used for formation of a merkle tree and data manipulation in it.
    - MerkleTools() - to implement a merkle tree.
  - os - for using system-generated data and functions.
    - urandom() - to generate cryptographically secure random keys.
  - pickle - for reading and writing of serialized data in any file.
    - dump() - to write serialized data in a file.
    - load() - to read serialized data from a file..
  - random - for random data generation.
    - randrange() - to give a range for the random number to generate between.
  - sys - for taking arguments from the command line

        - argv[] - to assign the argument from the command line to a variable, with the appropriate position number.
    - Miscellaneous -
        - pow() - for modular exponentiation calculations

- Assumptions, Limitations and Future Works
    a) The files in data.zip and mymerkle.zip must be downloaded as it is. The files inside it should not be downloaded separately.
    b) A voter's year of birth is considered here for ease of use. In an actual system, the whole Date of Birth can be used for fairness.
    c) The process of registering and polling is assumed to be known by the voter.
    d) The minimum age of the voter is assumed to be 21years.
    e) It is not necessary to use an actual SSN or DL number. Any dummy 4-5 digit number would work too.
    f) The number of candidates can be more than two.
    g) It is assumed that the Election Committee gets the public keys of all the voters during the registration.
    h) Merkle tree's pickled object should be binary for read/ write purposes.
    i) The server used here provides the vote count in a hashed format where the admin has to then match with the valid candidate ID.
    j) The audit doesn't check for all the valid candidates. Only a chosen candidate at a time is enquired for validity and consistency.
    k) An audit is not performed for uniqueness of the voter or the ballot in the audit server as it is already prevented from happening from the beginning.
    l) We can include timestamps for future works to track further, for any changes made in votes.
    m) We can explore other functionalities of SHAKE_256 like OAEP and use them in future models of this system.

- File Description

  There are 6 python files, namely :-
    - regserv.py - This file is the start of the system. In this, it takes 3 arguments from the command line; the name, the year of birth, and the

SSN number of the voter. Using the year, it checks if the voter is eligible to vote or not. The default age assumed is 21 years. An SSN is used to verify that an individual is uniquely identified. If a voter accidentally tries to re-register, he is assigned his previously generated voter ID along with new public and private keys. Once this verification is complete, a new 10-digit voter ID is generated using the SHAKE_256 on the SSN. Along with this, a hashed data of all the data provided is also created. A unique public and private key is also generated for future purposes of encryption of votes. All these data for each voter is mapped in a JSON file called data/voters.json. Every new voter data appends the existing data in it.

- pollserv.py - The polling process happens in this file. The voter enters the voter ID and the vote for his candidate by mentioning his candidate ID in the beginning in the command line. The voter ID is then used to verify if the voter is registered or not, by tallying it against voters.json file. If he is registered, then a unique ballot and ballot ID is generated for the voter to store the casted vote. The ballot ID along with its corresponding voter ID is stored in data/ballot.json file once the voting has been completed. After this, the vote is encrypted using the voter's private key for which data is loaded from voters.json . It is encrypted using the RSA Encryption Algorithm. The ballot ID and the encrypted vote is mapped in a data/polls.json file.
- storeserv.py - It reads data from all the files used for mapping and storing generated data. It uses the ballot.json to verify the ballot ID against the ballot ID present in polls.json file. It is also used to figure out the corresponding voter ID to any given ballot ID. Using the voter ID, we access the voter's public key to decrypt the encrypted vote. The vote comprises of a candidate ID. The sequence of candidate IDs is shown as a result. The sequence follows the same structure as the votes were stored. This sequence of candidate IDs is then hashed using SHA-256 and used as a leaf node to form a merkle tree. This tree is then dumped using pickle, in a file called Merkle.

- countserv.py - This file reads data from the Merkle file and counts the total votes for each candidate by using a counter for each of them while reading it from the Merkle file. The file initially reads from the pickled 'merkle' file which consists of the MerkleTools object. This object has the whole merkle tree in an immutable, binary form to read and process the leaves. The leaves of the Merkle Tree consists of the individual vote received for the candidate. The merkle tree helps in protecting the vote data from any kind of tampering and preserves the state of the voting sequence.
- auditserv.py - This file is used to verify the state of the merkle tree which is pickled from the 'Merkle' file. In this, a new candidate is taken from the argument in the command line. It is then checked against the Merkle file to check if it is a valid leaf node or sister node. If not, it will have a different root than the actual root. This way we can verify if the votes have ever been tampered at any time. Validating the proof of the target node provided in the argument makes the audit a success. If the node exists as leaf, then the tree is valid. Whereas it gives a false response on reading the tampered tree thereby verifying the voting sequence.
- helper.py - This file is mainly used for all the read and write data to and from a JSON file. It takes the name of the file, the data, and the type it is to be accessed. Then it dumps for writing and loads for reading a data from the .json file. It also checks if any duplicate voter is there and if someone is trying to tamper the votes, by always checking for a duplicity in the write data before appending the file.

➢ Implementation Example

The steps for command line interface of Windows 10 :-

```
>>>py regserv.py alice 1988 23047
You are registered. Your voter ID is: 1af6e92d0d. Do not disclose it to any unknown sources and third-parties.
>>> py regserv.py eva 2005 91475
Underage! You are not eligible to vote.
>>>py pollserv.py 1af6e92d0d 10
```

```
A unique ballot has been created for voter ID: 1af6e92d0d
Your vote has been casted successfully. Thank you!
>>> py pollserv.py ff7tb78y9p 5
Voter not found. Register!
>>>py pollserv.py 1af6e92d0d 10
You have already registered a vote. You cannot vote again.
>>>py storeserv.py
The sequence of votes stored:-
['5', '5', '6', '10', '10', '11', '10']
>>>py countserv.py
The count of votes for each candidate:-
{'ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d': 2,
'e7f6c011776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683': 1,
'4a44dc15364204a80fe80e9039455cc1608281820fe2b24f1e5233ade6af1dd5': 3,
'4fc82b26aecb47d2868c4efbe3581732a3e7cbcc6c2efb32062c08170a05eeb8': 1}
The candidate ID has been hidden for security reasons.
>>>py auditserv.py 6
True, candidate ID: 6 exists.
>>>py auditserv.py 16
Candidate ID: 15 doesn't exist.
```

To view the JSON files of stored data from the system:-

```
>>>type voters.json
```

[This will display the list of voter IDs, their hashed data, and the public and private keys for each one of them.]

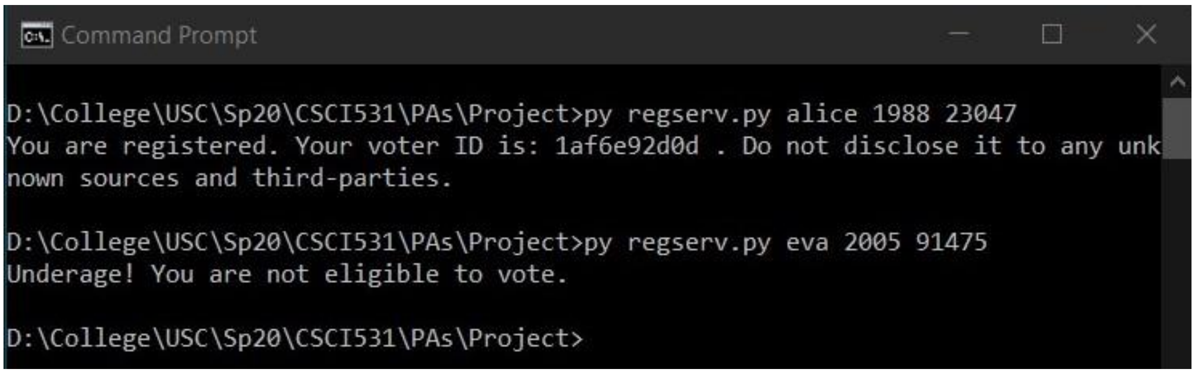
```
>>>type ballot.json
```

[This will display the list of ballots formed for every registered voter who has casted a vote.]

```
>>>type polls.json
```

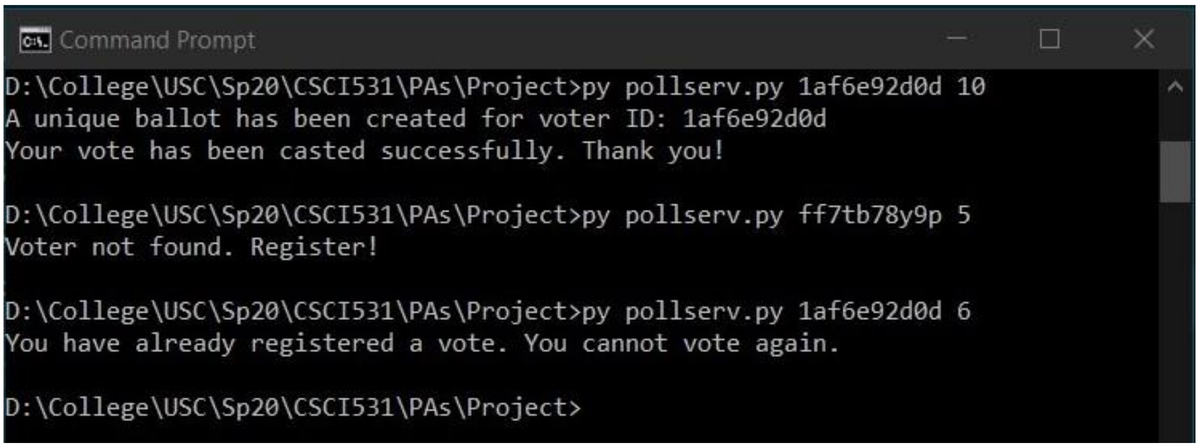
[This will display the RSA encrypted vote from every ballot used for voting.]

- Screenshots
  - regserv.py <name> <year_of_birth> <ssn>



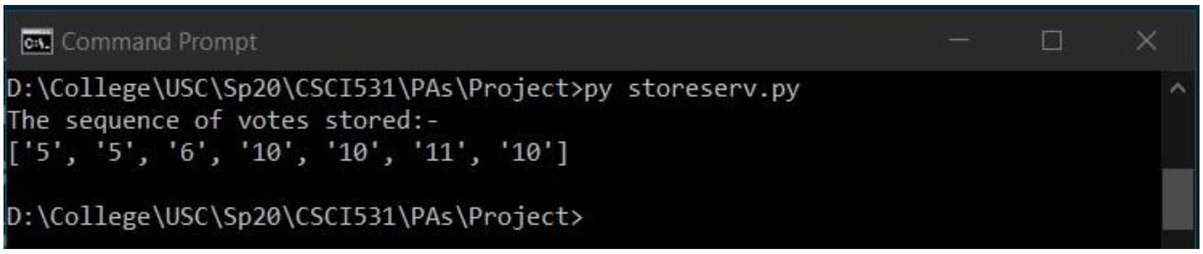
```
D:\College\USC\Sp20\CSCI531\PAs\Project>py regserv.py alice 1988 23047
You are registered. Your voter ID is: 1af6e92d0d . Do not disclose it to any unk
nown sources and third-parties.

D:\College\USC\Sp20\CSCI531\PAs\Project>py regserv.py eva 2005 91475
Underage! You are not eligible to vote.

D:\College\USC\Sp20\CSCI531\PAs\Project>
```

  - pollserv.py <voter_id> <candidate_id>



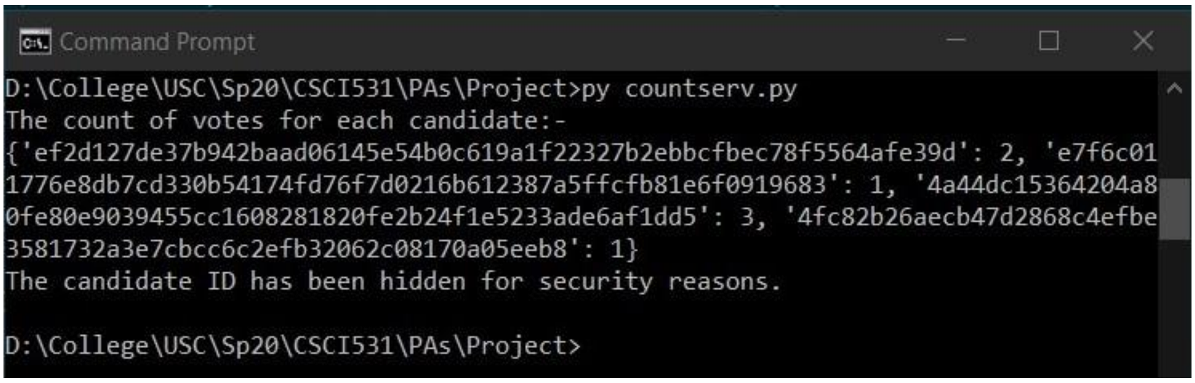
```
D:\College\USC\Sp20\CSCI531\PAs\Project>py pollserv.py 1af6e92d0d 10
A unique ballot has been created for voter ID: 1af6e92d0d
Your vote has been casted successfully. Thank you!

D:\College\USC\Sp20\CSCI531\PAs\Project>py pollserv.py ff7tb78y9p 5
Voter not found. Register!

D:\College\USC\Sp20\CSCI531\PAs\Project>py pollserv.py 1af6e92d0d 6
You have already registered a vote. You cannot vote again.

D:\College\USC\Sp20\CSCI531\PAs\Project>
```

  - storeserv.py



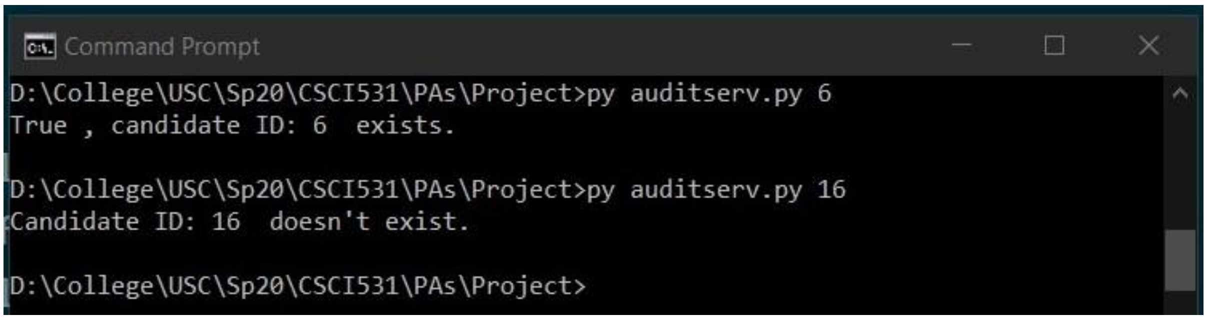
```
D:\College\USC\Sp20\CSCI531\PAs\Project>py storeserv.py
The sequence of votes stored:-
['5', '5', '6', '10', '10', '11', '10']

D:\College\USC\Sp20\CSCI531\PAs\Project>
```

  - countserv.py



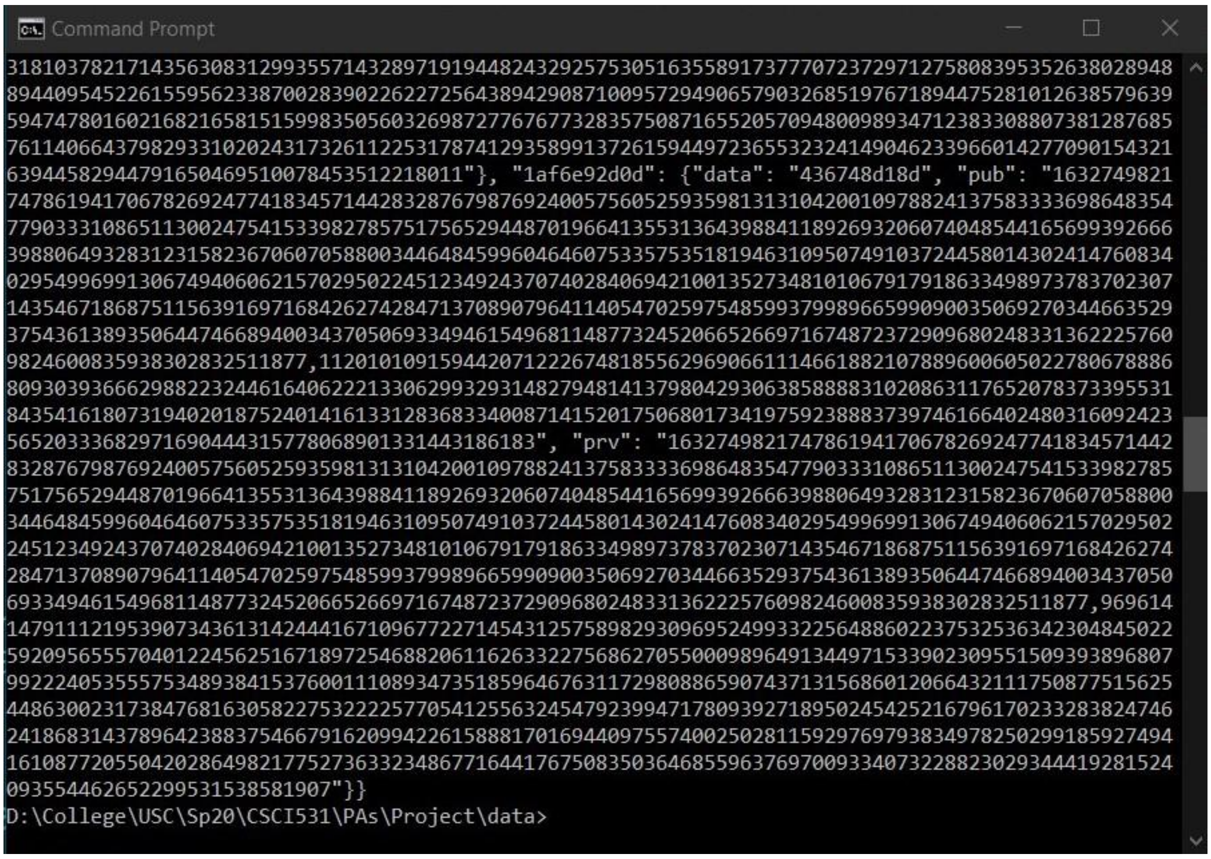
```
D:\College\USC\Sp20\CSCI531\PAs\Project>py countserv.py
The count of votes for each candidate:-
{'ef2d127de37b942baad06145e54b0c619a1f22327b2ebbcfbec78f5564afe39d': 2, 'e7f6c01
1776e8db7cd330b54174fd76f7d0216b612387a5ffcfb81e6f0919683': 1, '4a44dc15364204a8
0fe80e9039455cc1608281820fe2b24f1e5233ade6af1dd5': 3, '4fc82b26aecb47d2868c4efbe
3581732a3e7cbcc6c2efb32062c08170a05eeb8': 1}
The candidate ID has been hidden for security reasons.

D:\College\USC\Sp20\CSCI531\PAs\Project>
```

- auditserv.py <candidate_id>



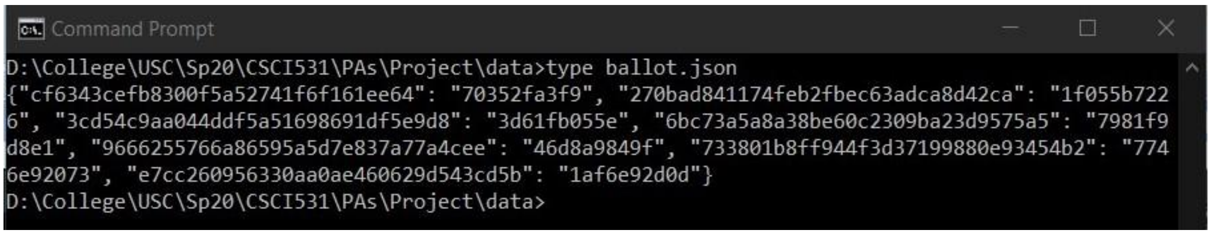
```
D:\College\USC\Sp20\CSCI531\PAs\Project>py auditserv.py 6
True , candidate ID: 6  exists.

D:\College\USC\Sp20\CSCI531\PAs\Project>py auditserv.py 16
Candidate ID: 16  doesn't exist.

D:\College\USC\Sp20\CSCI531\PAs\Project>
```

- voters.json



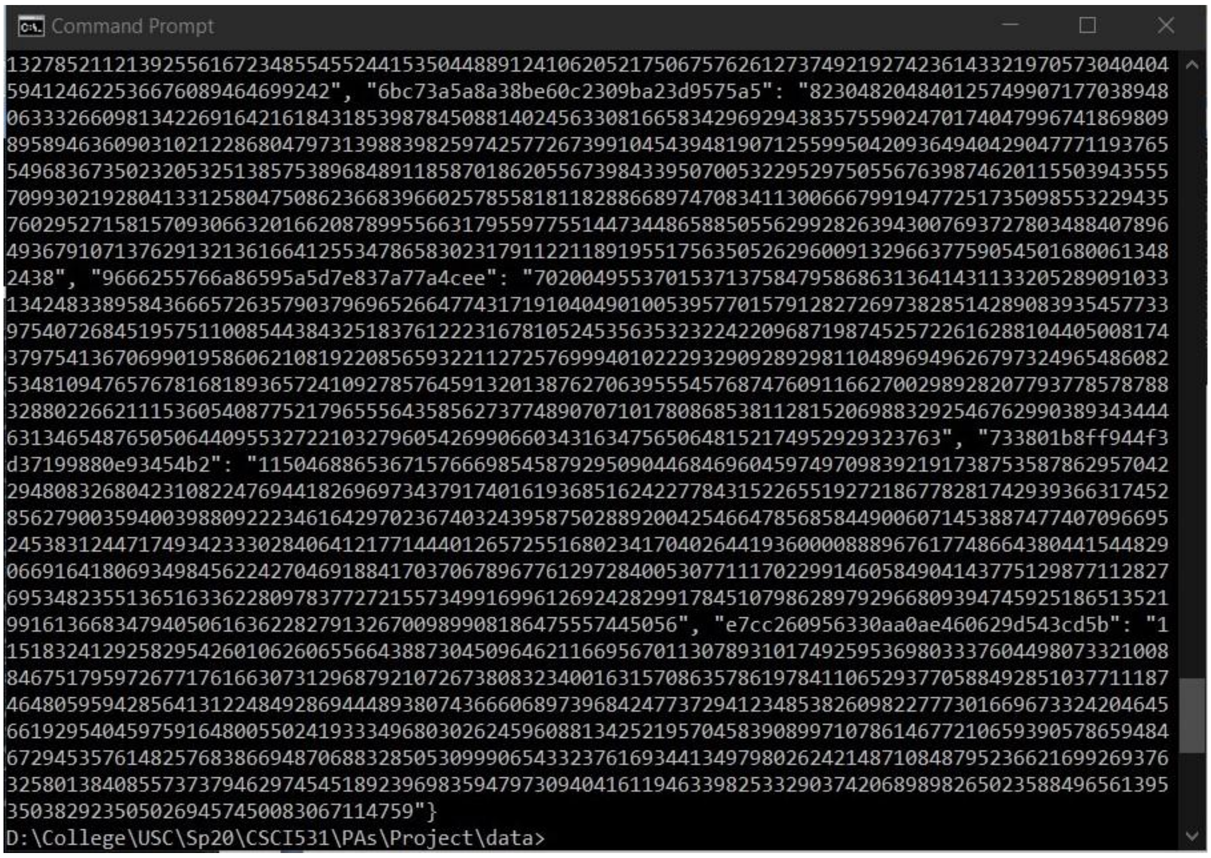
```
3181037821714356308312993557143289719194482432925753051635589173777072372971275808395352638028948
8944095452261559562338700283902262272564389429087100957294906579032685197671894475281012638579639
5947478016021682165815159983505603269872776767732835750871655205709480098934712383308807381287685
7611406643798293310202431732611225317874129358991372615944972365532324149046233966014277090154321
63944582944791650469510078453512218011"}, "1af6e92d0d": {"data": "436748d18d", "pub": "1632749821
7478619417067826924774183457144283287679876924005756052593598131310420010978824137583333698648354
7790333108651130024754153398278575175652944870196641355313643988411892693206074048544165699392666
3988064932831231582367060705880034464845996046460753357535181946310950749103724458014302414760834
0295499699130674940606215702950224512349243707402840694210013527348101067917918633498973783702307
1435467186875115639169716842627428471370890796411405470259754859937998966599090035069270344663529
3754361389350644746689400343705069334946154968114877324520665266971674872372909680248331362225760
9824600835938302832511877,11201010915944207122267481855629690661114661882107889600605022780678886
8093039366629882232446164062221330629932931482794814137980429306385888831020863117652078373395531
8435416180731940201875240141613312836833400871415201750680173419759238883739746166402480316092423
56520333682971690444315778068901331443186183", "prv": "163274982174786194170678269247741834571442
8328767987692400575605259359813131042001097882413758333369864835477903331086511300247541533982785
7517565294487019664135531364398841189269320607404854416569939266639880649328312315823670607058800
3446484599604646075335753518194631095074910372445801430241476083402954996991306749406062157029502
2451234924370740284069421001352734810106791791863349897378370230714354671868751156391697168426274
2847137089079641140547025975485993799896659909003506927034466352937543613893506447466894003437050
693349461549681148773245206652669716748723729096802483313622257609824600835938302832511877,969614
1479111219539073436131424441671096772271454312575898293096952499332256488602237532536342304845022
5920956555704012245625167189725468820611626332275686270550009896491344971533902309551509393896807
9922240535557534893841537600111089347351859646763117298088659074371315686012066432111750877515625
4486300231738476816305822753222257705412556324547923994717809392718950245425216796170233283824746
2418683143789642388375466791620994226158881701694409755740025028115929769793834978250299185927494
1610877205504202864982177527363323486771644176750835036468559637697009334073228823029344419281524
093554462652299531538581907"}}
D:\College\USC\Sp20\CSCI531\PAs\Project\data>
```

- ballot.json



```
D:\College\USC\Sp20\CSCI531\PAs\Project\data>type ballot.json
{"cf6343cefb8300f5a52741f6f161ee64": "70352fa3f9", "270bad841174feb2fbec63adca8d42ca": "1f055b722
6", "3cd54c9aa044ddf5a51698691df5e9d8": "3d61fb055e", "6bc73a5a8a38be60c2309ba23d9575a5": "7981f9
d8e1", "9666255766a86595a5d7e837a77a4cee": "46d8a9849f", "733801b8ff944f3d37199880e93454b2": "774
6e92073", "e7cc260956330aa0ae460629d543cd5b": "1af6e92d0d"}
D:\College\USC\Sp20\CSCI531\PAs\Project\data>
```

- polls.json



```
1327852112139255616723485545524415350448891241062052175067576261273749219274236143321970573040404
594124622536676089464699242", "6bc73a5a8a38be60c2309ba23d9575a5": "823048204840125749907177038948
0633326609813422691642161843185398784508814024563308166583429692943835755902470174047996741869809
8958946360903102122868047973139883982597425772673991045439481907125599504209364940429047771193765
5496836735023205325138575389684891185870186205567398433950700532295297505567639874620115503943555
7099302192804133125804750862366839660257855818118288668974708341130066679919477251735098553229435
7602952715815709306632016620878995566317955977551447344865885055629928263943007693727803488407896
4936791071376291321361664125534786583023179112211891955175635052629600913296637759054501680061348
2438", "9666255766a86595a5d7e837a77a4cee": "70200495537015371375847958686313641431133205289091033
1342483389584366657263579037969652664774317191040490100539577015791282726973828514289083935457733
9754072684519575110085443843251837612223167810524535635323224220968719874525722616288104405008174
3797541367069901958606210819220856593221127257699940102229329092892981104896949626797324965486082
5348109476576781681893657241092785764591320138762706395554576874760911662700298928207793778578788
3288022662111536054087752179655564358562737748907071017808685381128152069883292546762990389343444
631346548765050644095532722103279605426990660343163475650648152174952929323763", "733801b8ff944f3
d37199880e93454b2": "1150468865367157666985458792950904468469604597497098392191738753587862957042
2948083268042310822476944182696973437917401619368516242277843152265519272186778281742939366317452
8562790035940039880922234616429702367403243958750288920042546647856858449006071453887477407096695
2453831244717493423330284064121771444012657255168023417040264419360000888967617748664380441544829
0669164180693498456224270469188417037067896776129728400530771117022991460584904143775129877112827
6953482355136516336228097837727215573499169961269242829917845107986289792966809394745925186513521
99161366834794050616362282791326700989908186475557445056", "e7cc260956330aa0ae460629d543cd5b": "1
1518324129258295426010626065566438873045096462116695670113078931017492595369803337604498073321008
8467517959726771761663073129687921072673808323400163157086357861978411065293770588492851037711187
4648059594285641312248492869444893807436660689739684247737294123485382609822777301669673324204645
6619295404597591648005502419333496803026245960881342521957045839089971078614677210659390578659484
6729453576148257683866948706883285053099906543323761693441349798026242148710848795236621699269376
3258013840855737379462974545189239698359479730940416119463398253329037420689898265023588496561395
3503829235050269457450083067114759"}
D:\College\USC\Sp20\CSCI531\PAs\Project\data>
```

-----X-----X-----X-----