Electronic Voting System

Technical Manual

[1. Introduction 2](#_Toc453112687)

[1.1. Document Identification 2](#_Toc453112688)

[1.2. System Overview 2](#_Toc453112689)

[1.3. Document Overview 2](#_Toc453112690)

[1.3.1. Reference Documents 2](#_Toc453112691)

[1.3.2. Acronyms and Abbreviations 2](#_Toc453112692)

[1.3.3. Modification Record 2](#_Toc453112693)

[2. System Description 3](#_Toc453112694)

[2.1. Introduction 3](#_Toc453112695)

[2.2. Operational Scenarios 3](#_Toc453112696)

[2.3. System Requirements 6](#_Toc453112697)

[2.3.1. Functional Requirements 6](#_Toc453112698)

[2.3.2. Non-functional Requirements 6](#_Toc453112699)

[2.4. Module Design 7](#_Toc453112700)

[2.5. Class Diagram Design 9](#_Toc453112701)

[3. Software Design 9](#_Toc453112702)

[3.1. Software Design Process 10](#_Toc453112703)

[3.1.1. Software Development Environment 10](#_Toc453112704)

[3.1.2. Software Implementation Stages 11](#_Toc453112705)

[3.2. Software Design Description 12](#_Toc453112706)

[3.2.1. Architecture 12](#_Toc453112707)

[3.2.2. Tables 16](#_Toc453112708)

[3.2.3. Software Interface 17](#_Toc453112709)

[3.2.3.1. Software Interface at Main Server 17](#_Toc453112710)

[3.2.3.2. Software Interface at Voter 19](#_Toc453112711)

[3.2.3.3. Software Interface at HV Server 22](#_Toc453112712)

[3.2.3.4. Software Interface at Tallying Server 25](#_Toc453112713)

[3.2.3.5. Main Server Public Interface 26](#_Toc453112714)

# Introduction

## Document Identification

This document describes the design of electronic voting system.

## System Overview

The purpose of this electronic voting system is to improve voting process. It is a privacy-aware electronic voting system which guarantees receipt-freeness as well as privacy, universal verifiability and robustness. A trusted third party called honest verifier (HV) is introduced which verifies the validity of the first ballot and provides the procedure parameter.

## Document Overview

In this document, I firstly give a general overview of the basis for the electronic voting system design, and use use-case diagram to describe how the system can be used. After that, I describe the system requirements module design and class diagram design. Then, I show the software implementation stages and use sequence diagrams to describe software design. Finally, I describe the software interface.

### Reference Documents

The present document is prepared on the basis of the following reference documents, and should be read in conjunction with them.

* LEE, B. & KIM, K. Receipt-free electronic voting through collaboration of voter and honest verifier. Proceeding of JW-ISC2000, 2000. Citeseer.
* WELDEMARIAM, K. & VILLAFIORITA, A. A Survey: Electronic Voting Development and Trends. Electronic Voting, 2010. 119-131.
* BHIOGADE, M. S. Secure socket layer. Computer Science and Information Technology Education Conference, 2002. 85-90.

### Acronyms and Abbreviations

Table 1.1 lists the acronyms and abbreviations used in this document.

Table 1.1 Acronyms and Abbreviations

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| HV | Honest Verifier |
| CA | Certificate Authority |
| PV1 | First Ballot |
| PK | Common Exponent |
| PV2 | Final Ballot |

### Modification Record

Table 1.2 lists the record of who and when modified this document.

Table 1.2 Modification record

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Name | Content |
| 1 | 21, May, 2016 | Danli Jiang | Create file |
| 2 | 28, May, 2016 | Danli Jiang | Modify section 2 |
| 3 | 5, June, 2016 | Danli Jiang | Modify section 3 |

# System Description

This section is intended to give a general overview of the basis for the electronic voting system design, and of its development and implementation.

## Introduction

System design performs the following activities: design the goals for subsystems from non-functional requirements, decompose the system into subsystems and mention strategies for subsystems.

The strategies include hardware, software, global control flow, data structures (persistent data storage), access control policy, boundary conditions and exception handling mechanisms.

## Operational Scenarios

In this section, I will describe how the system can be used. There may be several different ways that it can be used, perhaps involving different users, or classes of user. Each operational scenario is a part through a use case diagram. Use case diagrams are shown as follows.

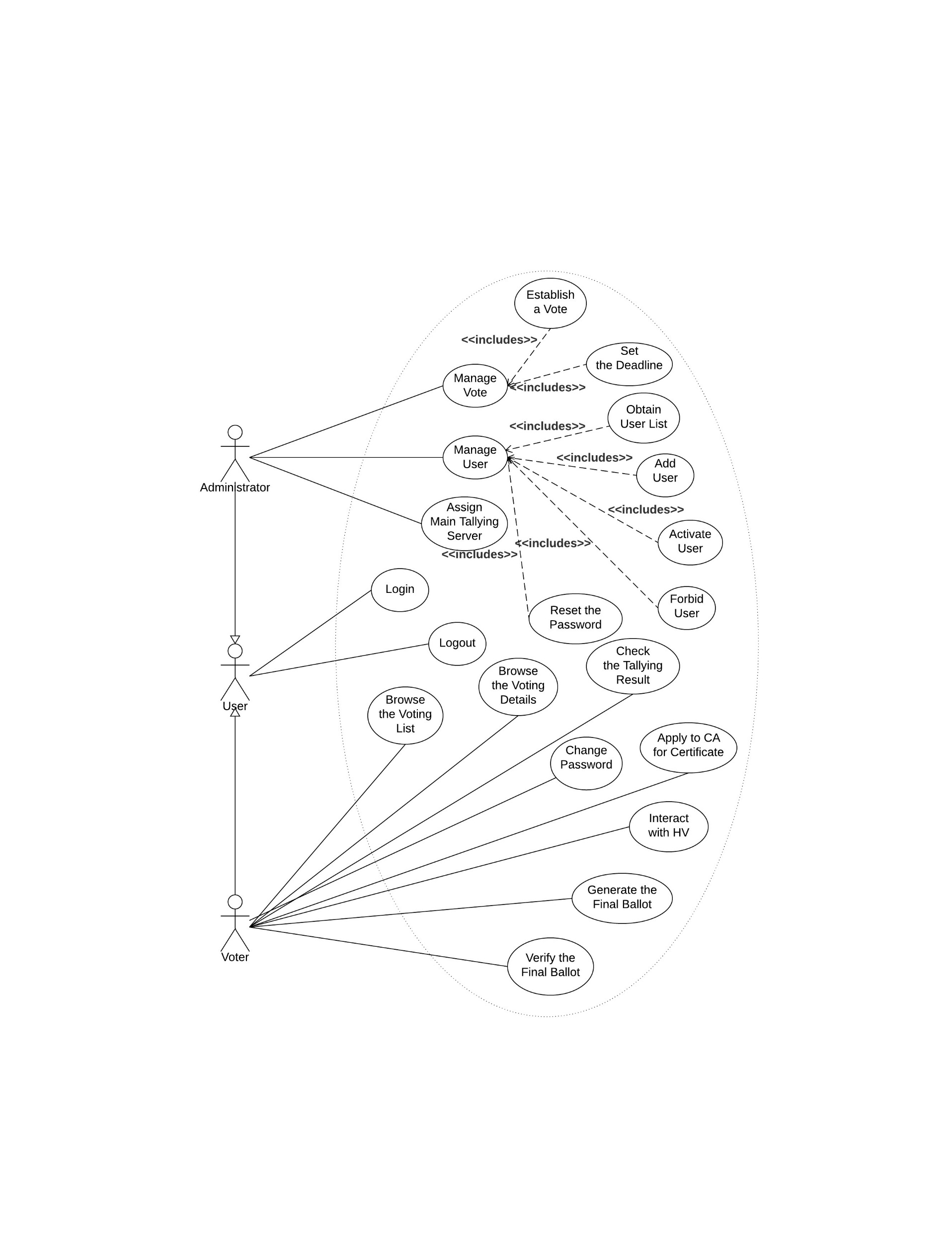


Figure 2.1 Use case diagram for user

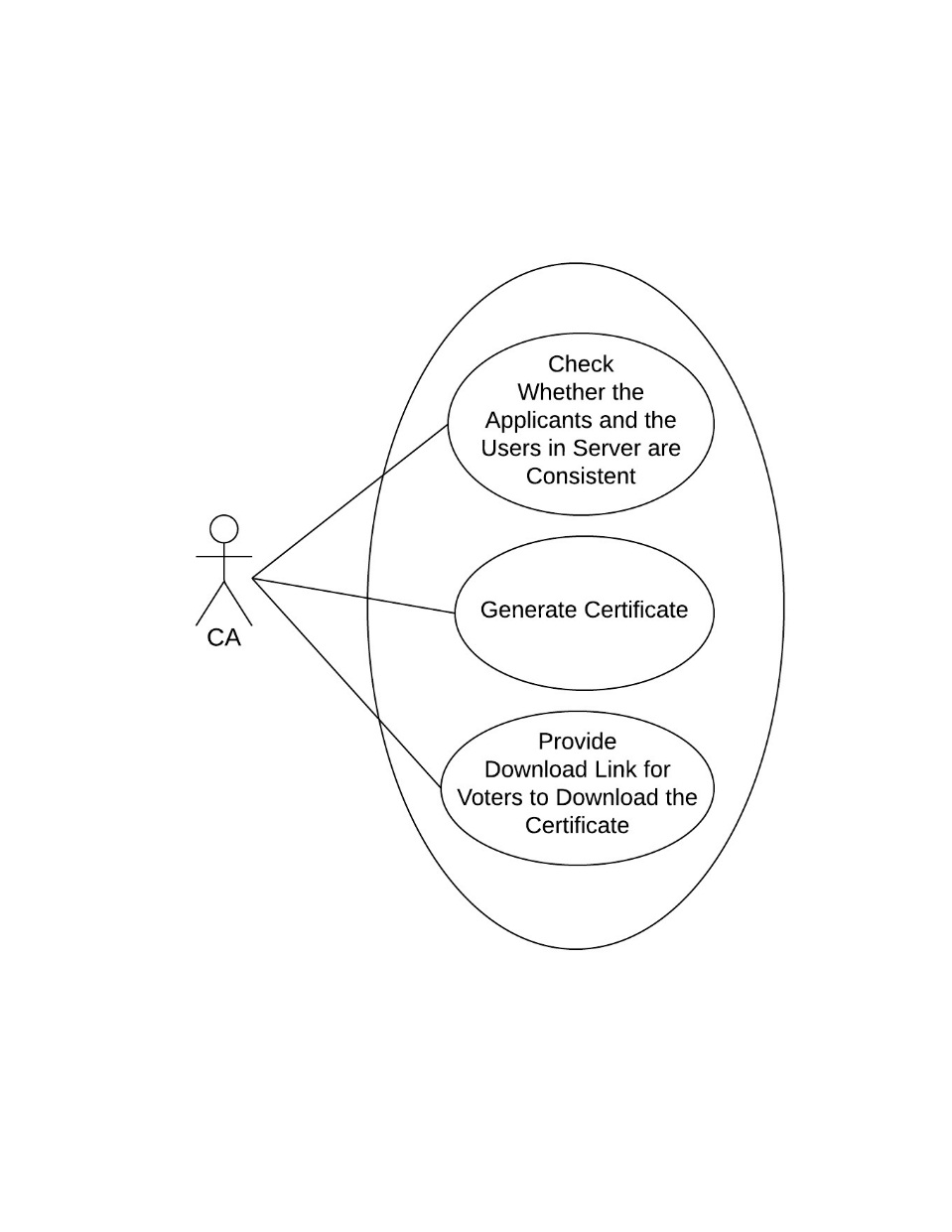


Figure 2.2 Use case diagram for CA

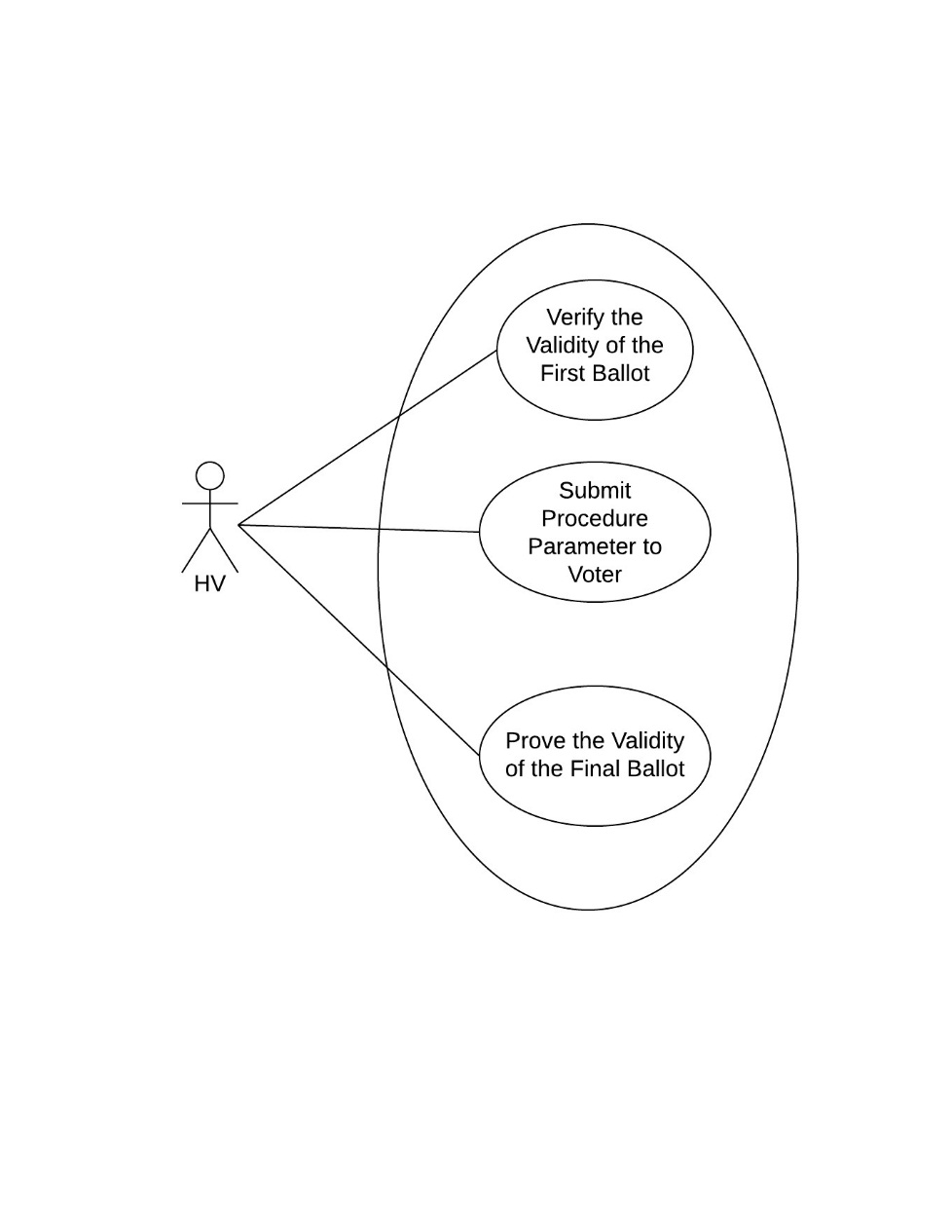


Figure 2.3 Use case diagram for HV

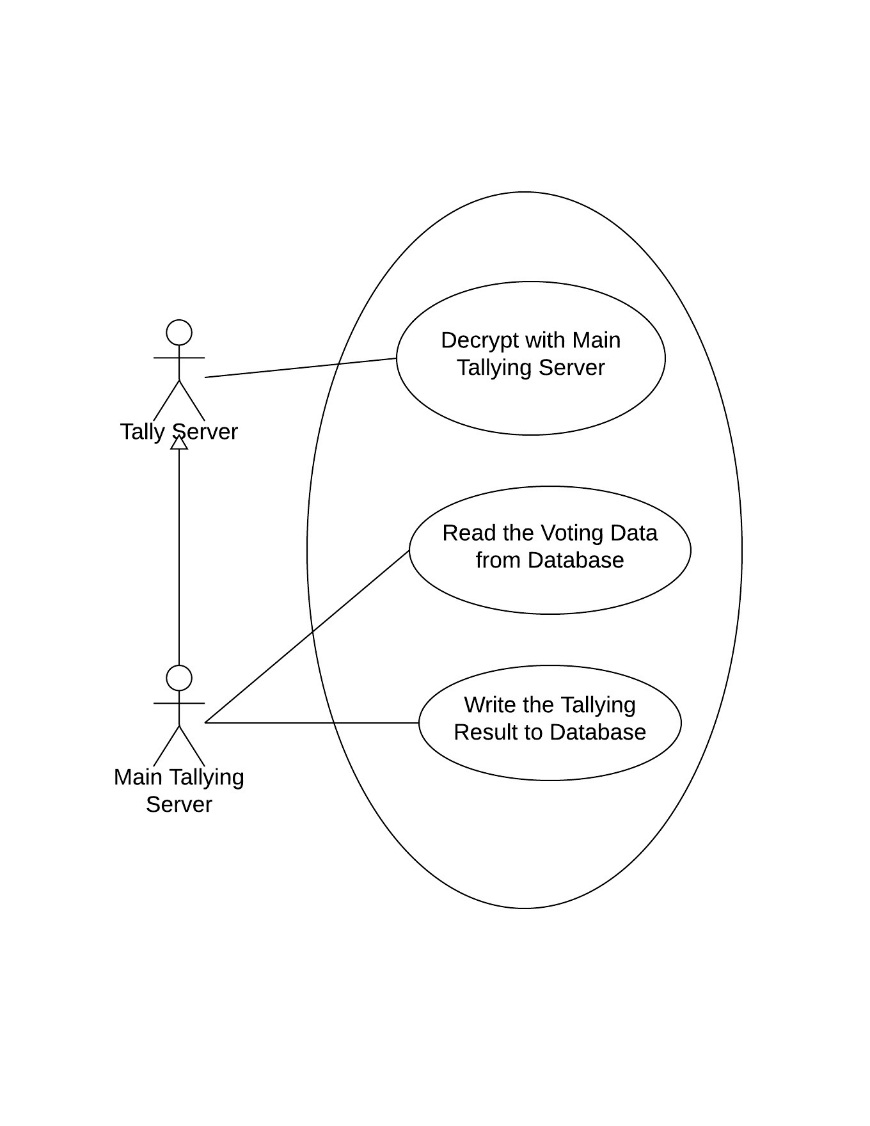


Figure 2.4 Use case diagram for Tallying Server

## System Requirements

### Functional Requirements

**Tallying Authorities:** generate system parameters, show some of voters’ information on the bulletin board including challenges, responses and the validity of the final ballots, collect all the valid ballots, encryption and send notifications, decryption and count the ballots, and announce the finally result after the deadline.

**Voters:** register, login, browse the newest vote and its available options, generate the first ballot. (participate through an interactive voting protocol), generate the final ballot, verify the validity of ballots, and browse the bulletin board.

**CA:** generate certificate for voters.

**HV:** validate the first ballot, generate the proof of validity of final ballot, post the validity of final ballot on the bulletin board.

**Vote administrator:** establish a vote, check if voters have casted before.

### Non-functional Requirements

Some important characteristics for this project must highlight the Privacy, Completeness, Soundness, Un-reusability, Eligibility, Fairness, Robustness, Universal verifiability and Receipt-freeness

Privacy – No one can gain any information about the voters.

Completeness – All valid votes are calculated correctly.

Soundness – Voting cannot be disrupted by the dishonest voter.

Un-reusability – No voter can vote second time.

Eligibility – Only eligible voters can cast the votes. Every voter can cast only one vote.

Fairness – No participant can gain any knowledge about the tally before the counting stage.

Robustness – Faulty behavior of any reasonably sized coalition of participants can be tolerated. No coalition of voters can disrupt the election and any cheating voter will be detected.

Universal verifiability – The validity of voting and tallying process can be verified by anyone.

Receipt-freeness – The vote cannot be proven to a buyer.

## Module Design

The system can be breakdown into six functional modules which includes main server, administrator, voter, certificate authority (CA), tallying server and honest verifier (HV).

1. main server
2. respond to the login request
3. receive the vote details and write it to database
4. respond to the voting list request
5. respond to the voting details request
6. receive the voting deadline and write it to database
7. receive user’s information and write it to database
8. read the user list from database and send it to user
9. change user’s status to activated in database
10. change user’s status to forbidden in database
11. initialize user’s password in database
12. change user’s password in database
13. change the tallying server status to main in database
14. remove the login status on the server side
15. receive the first ballot and write it to database
16. respond to the first ballot request
17. receive the final ballot and write it to database
18. respond to the final ballot request
19. receive the tallying result and write it to database
20. respond to the tallying result request
21. receive user’s certificate request and send it to CA
22. receive the procedure parameter from HV and write it to database
23. respond to the procedure parameter request
24. administrator
25. send the login request to server
26. assign the main tallying server
27. remove the login status on the client side
28. manage vote
29. send the vote details to server
30. send the voting deadline to server
31. manage user
32. send the user list request to server
33. send user’s information to server
34. activate user
35. forbid user
36. reset the password
37. voter
38. send the login request to server
39. send the voting list request to server
40. send the voting details request to server
41. send the tallying result request to server
42. change the password
43. apply to CA for certificate
44. send the procedure parameter request to server
45. send the first ballot to server
46. send the final ballot to server
47. verify the final ballot
48. remove the login status on the client side
49. CA
50. check whether the applicants and the users are consistent
51. generate the certificate
52. provide download link for voters to download the certificate
53. tallying server
54. read the voting data from database
55. execute the decryption protocol
56. send the tallying result to server
57. HV
58. verify the validity of the first ballot
59. send the procedure parameter to server
60. prove the validity of the final ballot

## Class Diagram Design

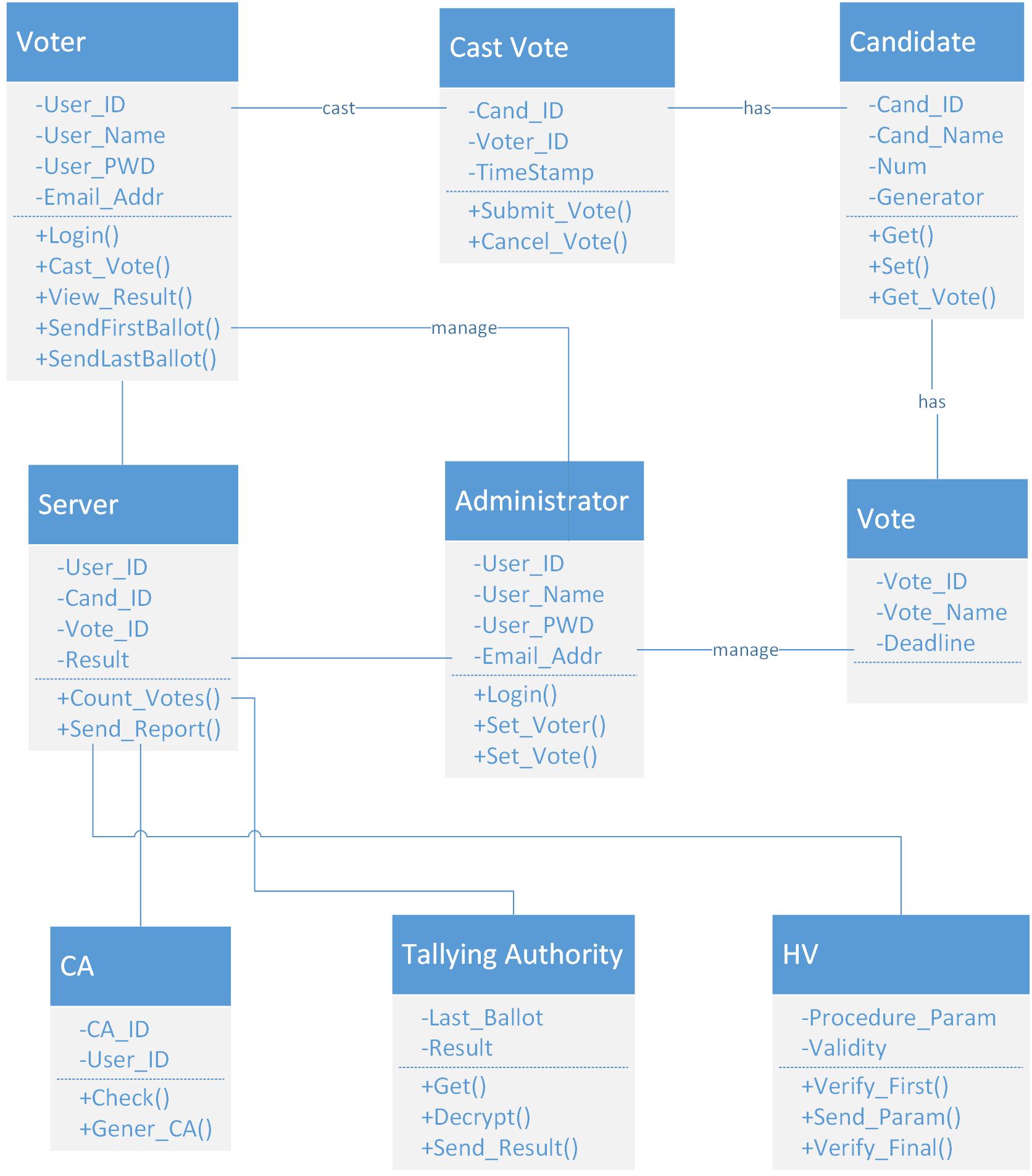


Figure 2.5 Class diagram

# Software Design

The software requirements and overview have been dealt with elsewhere in this document. The present section addresses the design and implementation of the software that forms the electronic voting system.

## Software Design Process

We used the waterfall model to design our system. We followed the following steps in order: state the requirements, analyze them, design a solution approach, architect a software framework for that solution, develop code, test, deploy, and maintain.

### Software Development Environment

**Server environment:**

We deploy web server based on Linux operating system which can effectively reduce the security risks, in order to ensure the security of the web server. We also use Node.js as Javascript running environment which is a platform based on the V8 engine.

**Database:**

We install the MySQL Community Server (the free version available1) for database server.

**Programming language:**

We programmed the system using C++ and Javascript.

**Library:**

We use Openssl 0.9.8zg, Rapidjson, boost 1.54 and Qt 5.5 as our c++ library. Moreover, we use the modules listed below as our node.js libraries:

1. bcrypt-nodejs: 0.0.3,
2. body-parser: 1.13.1,
3. connect-flash: 0.1.1,
4. cookie-parser: 1.3.5,
5. crypto: 0.0.3,
6. ejs: 2.3.2,
7. emailjs: 1.0.5,
8. express: 4.13.0,
9. express-session: 1.11.3,
10. jwt-simple: 0.5.0,
11. log4js: 0.6.36,
12. moment: 2.13.0,
13. morgan: 1.6.0,
14. mysql: 2.7.0,
15. node-schedule: 1.1.0

**Version control tool:**

We use svn as our version control tool.

**Data interchange format between C++ and Node.js:**

We use JSON as our data interchange format. JSON is a lightweight text-based data interchange format that is easy for both humans and computers to digest and consume.

**Developing machine environment:**

We use Macintosh as our developing machine and Mac operating system as our developing OS environment.

**Integrated developing environment:**

We use Qt creator and CLion as c++ IDE and Webstorm as node.js IDE. What’s more, we use qmake in Qt creator as the compile script language while using cmake in CLion. Last but not least, we apply npm as the package managing tool for node.js.

### Software Implementation Stages

• Requirements Analysis

Extracting the requirements of a desired software product is the first task in creating it. While customers probably believe they know what the software is to do, it may require skill and experience in software engineering to recognize incomplete, ambiguous or contradictory requirements.

• Specification

Specification is the task of precisely describing the software to be written, in a mathematically rigorous way. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.

• Software architecture

The architecture of a software system refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the product, as well as ensuring that future requirements can be addressed.

• Implementation

Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion.

• Testing

Testing of parts of software, especially where code by two different engineers must work together, falls to the software engineer.

• Documentation

An important task is documenting the internal design of software for the purpose of future maintenance and enhancement.

• Training and Support

A large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are occasionally resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, it’s very important to have training classes for the most enthusiastic software users (build excitement and confidence), shifting the training towards the neutral users intermixed with the avid supporters, and finally incorporate the rest of the organization into adopting the new software. Users will have lots of questions and software problems which leads to the next phase of software.

## Software Design Description

### Architecture

In this section, I will describe the high-level architecture of the software – that is, the top-level flow of control, and how the various functional modules communicate. The sequence diagrams are shown as follows.

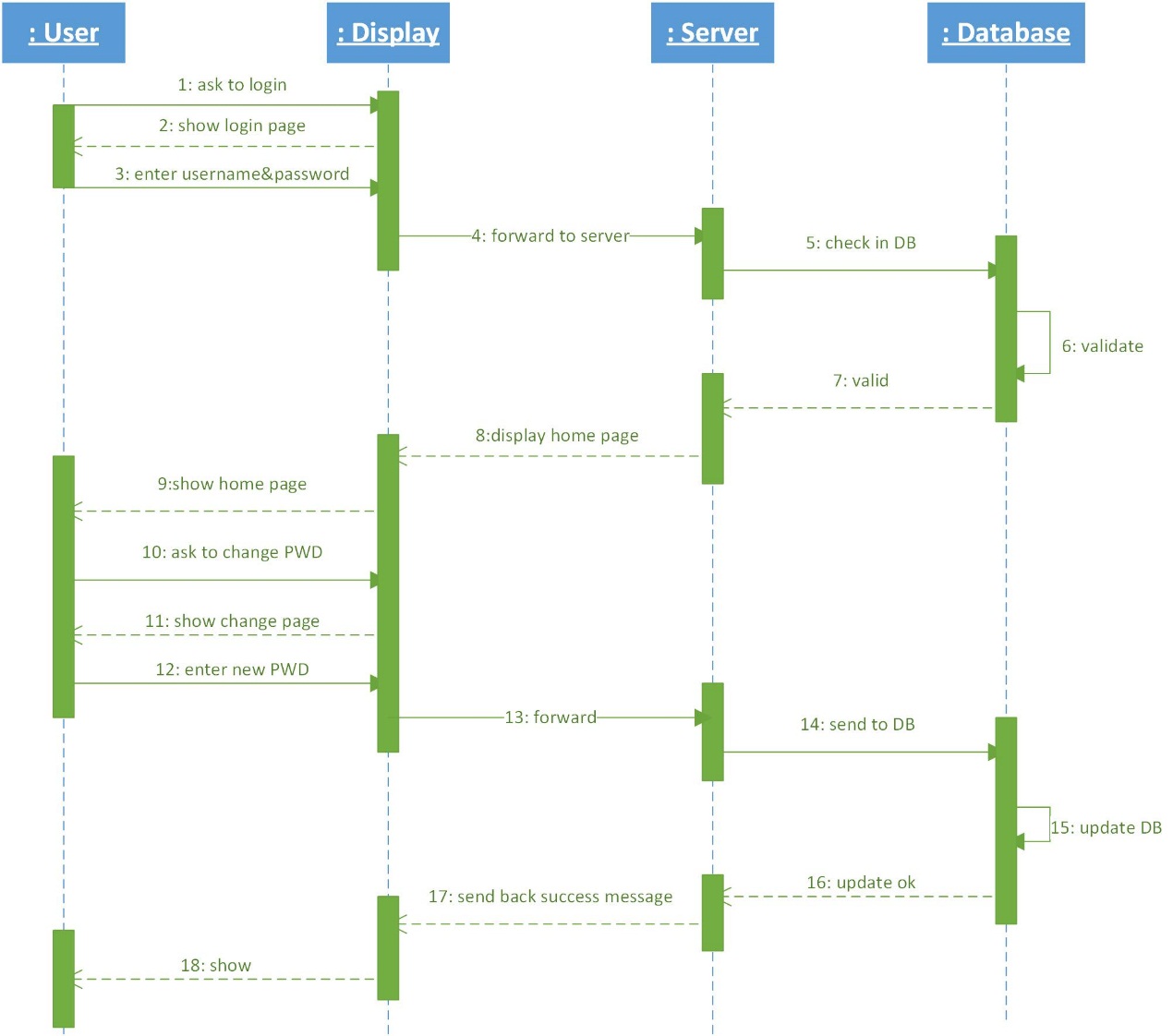


Figure 3.1 Sequence diagram for user’s login

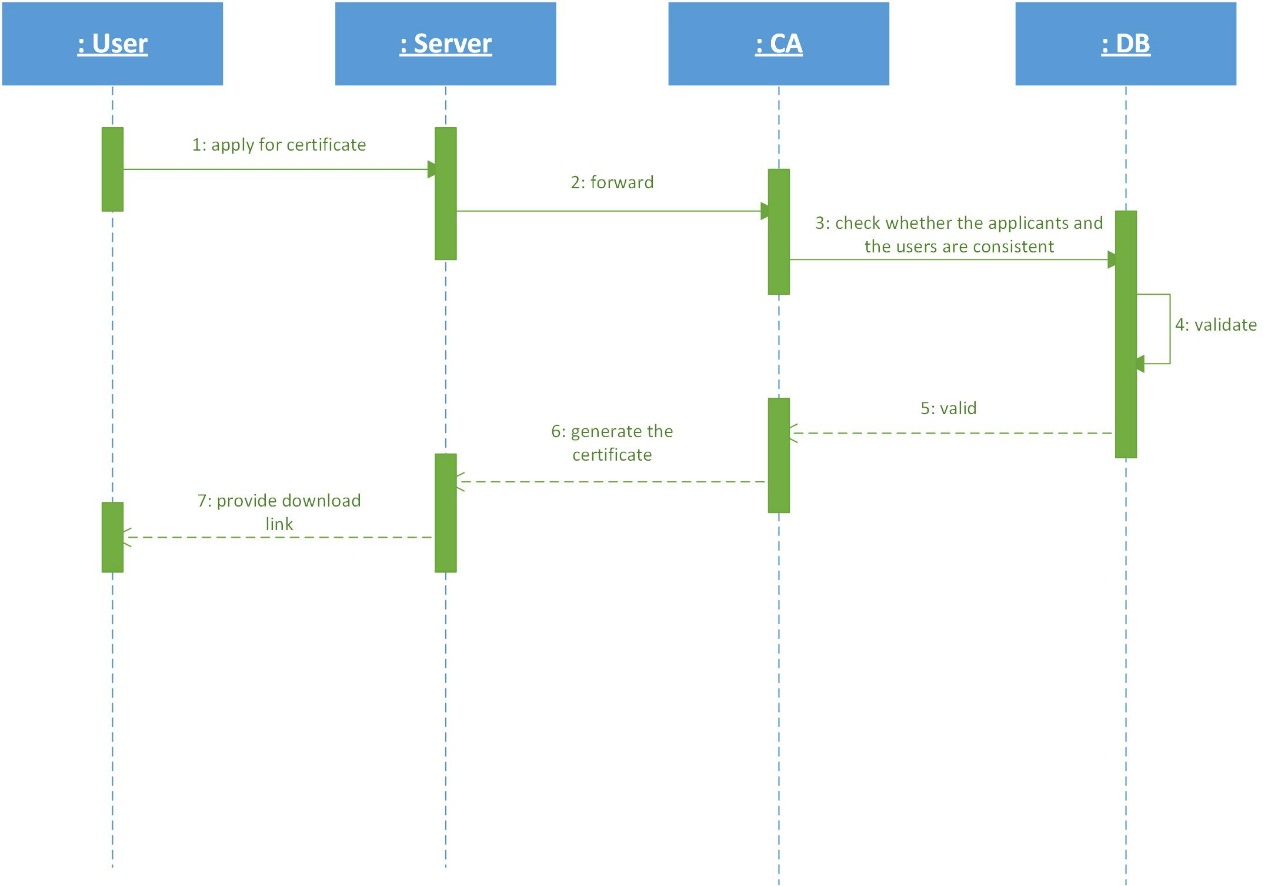


Figure 3.2 Sequence diagram for certificate authority

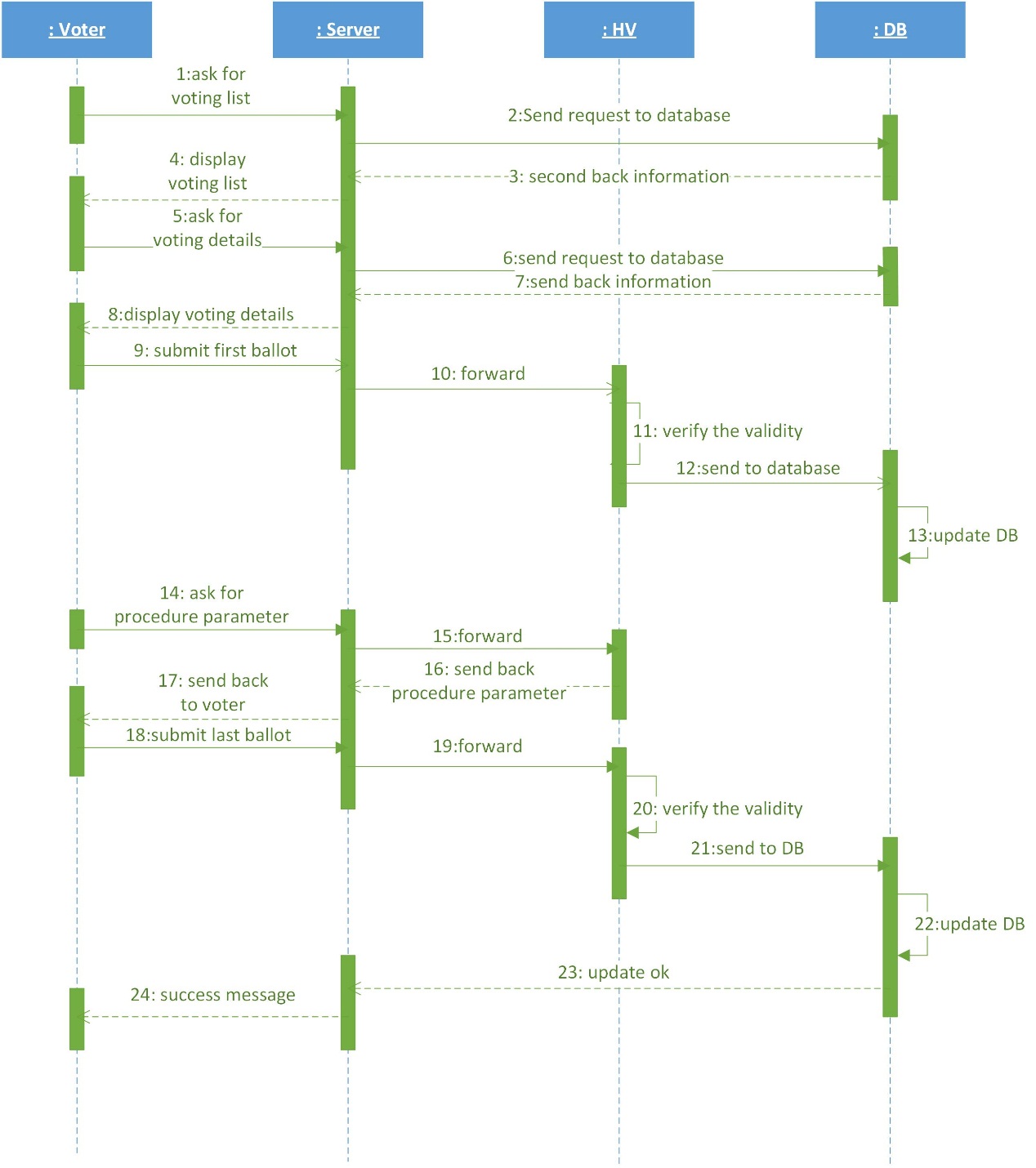


Figure 3.3 Sequence diagram for voting process

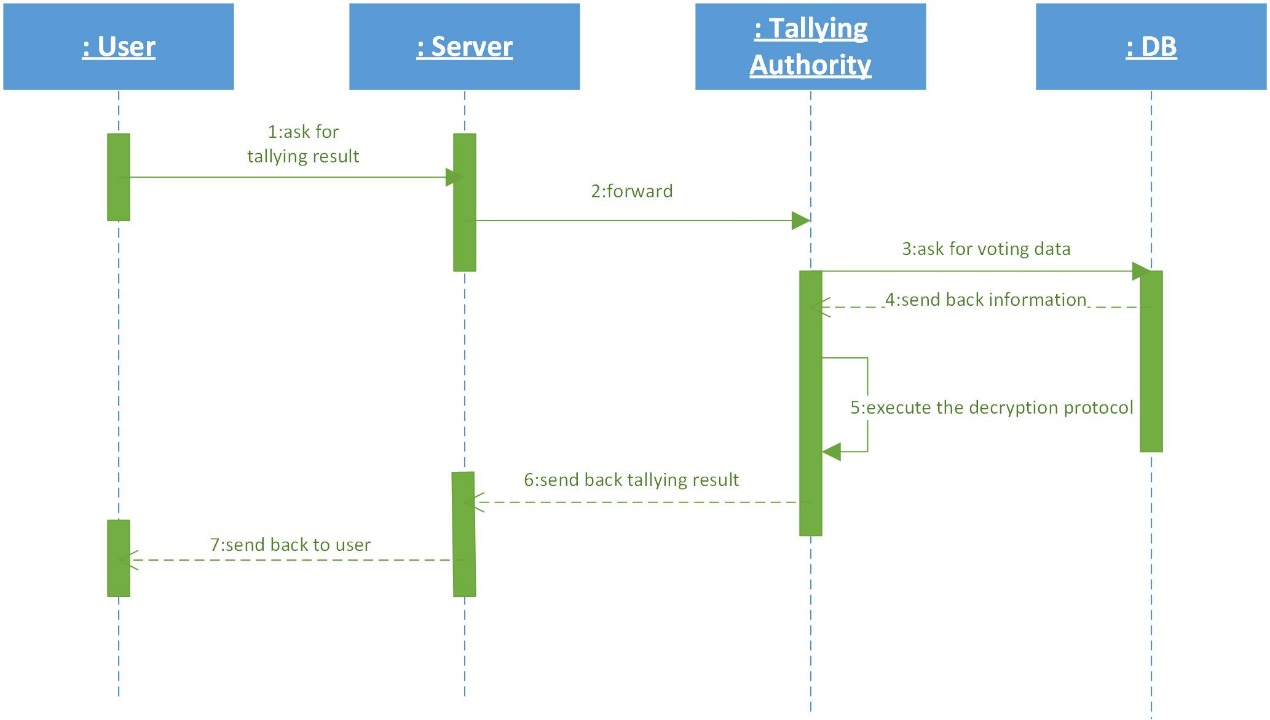


Figure 3.4 Sequence diagram for checking result

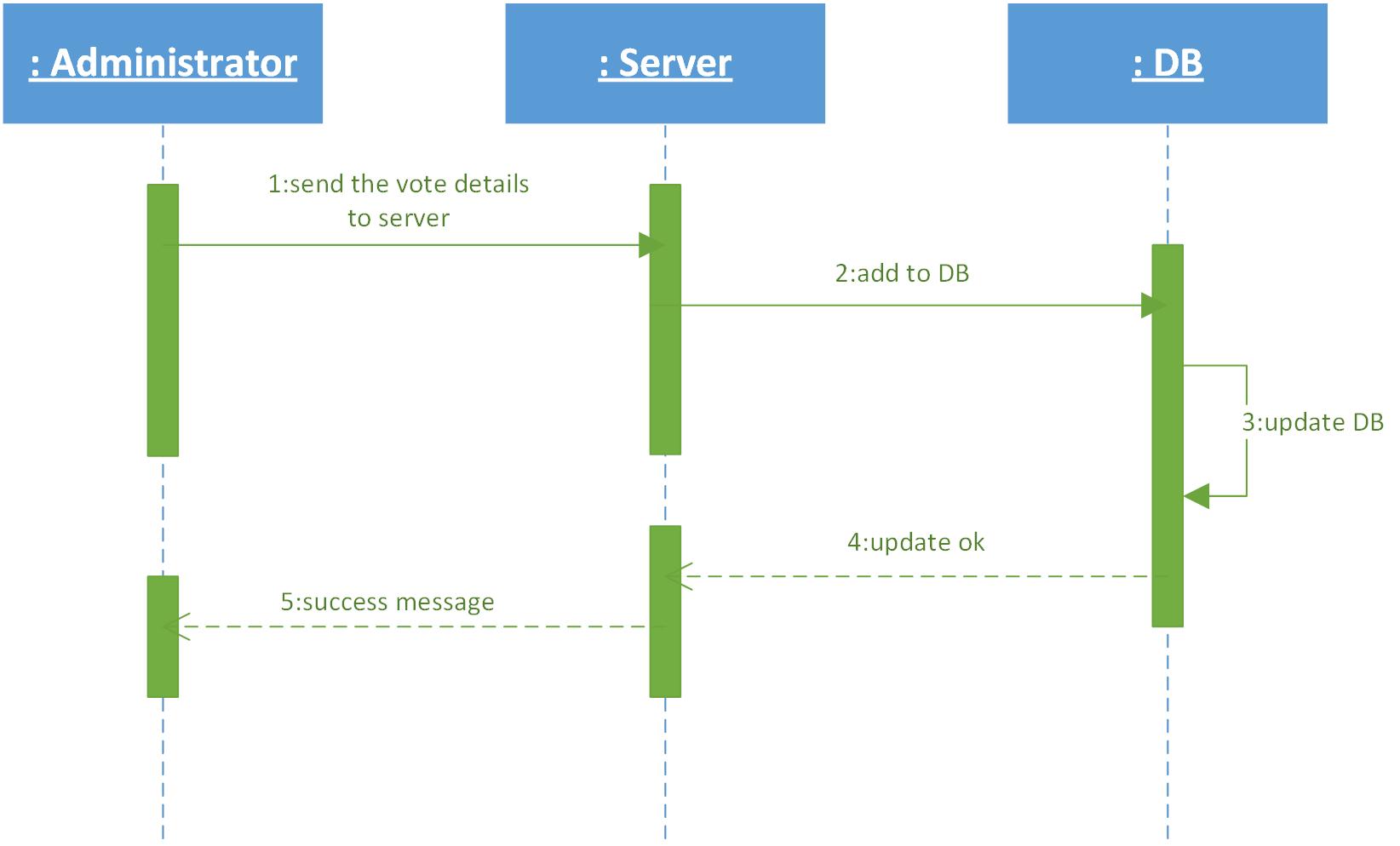


Figure 3.5 Sequence diagram for vote management

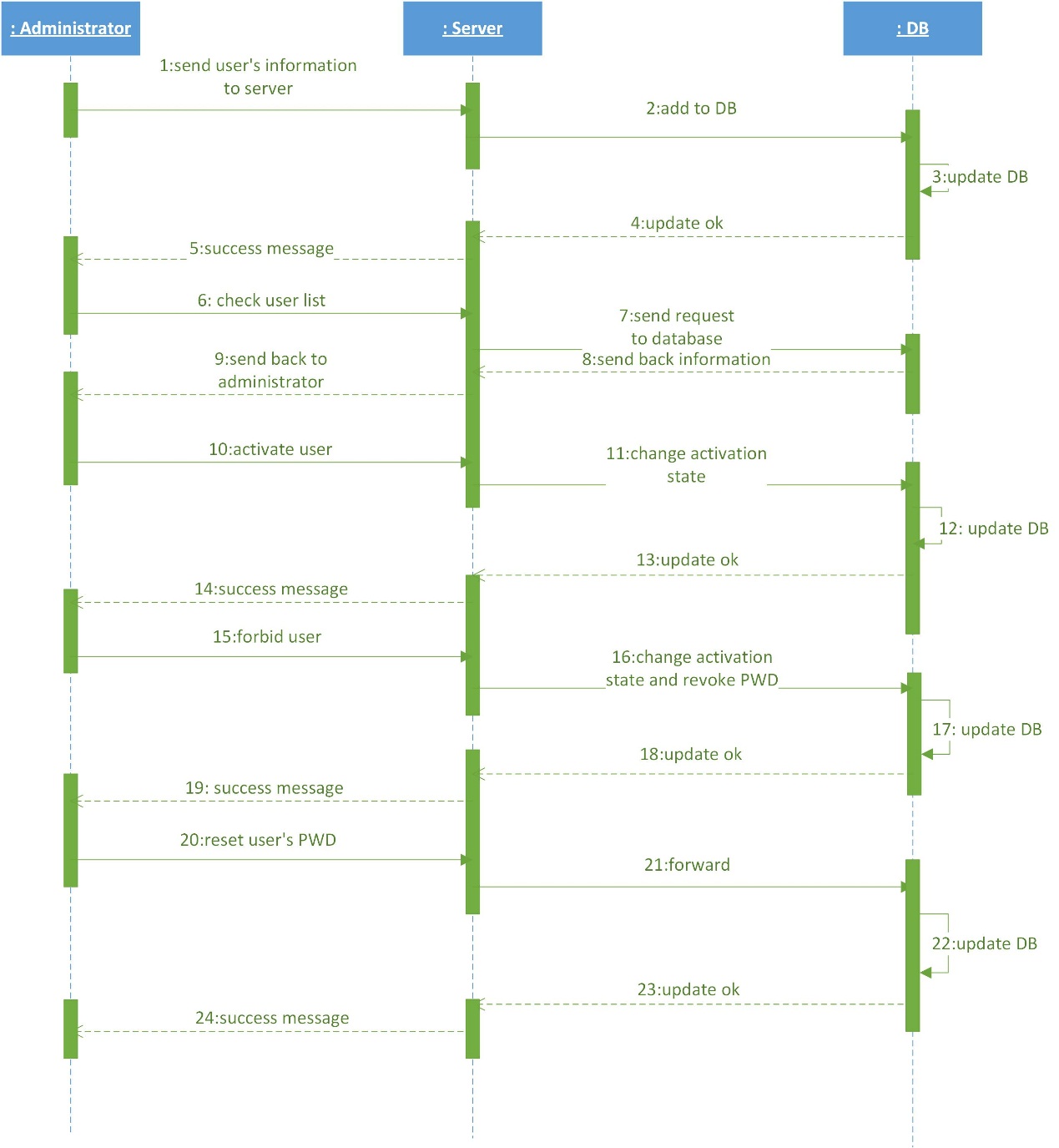


Figure 3.6 Sequence diagram for user management

### Tables

We have 8 tables in database in server to maintain user, vote and parameter information. All tables can be related to each other via id such as vote\_id, user\_id and cand\_id. The figure of database physical design is shown as follows.

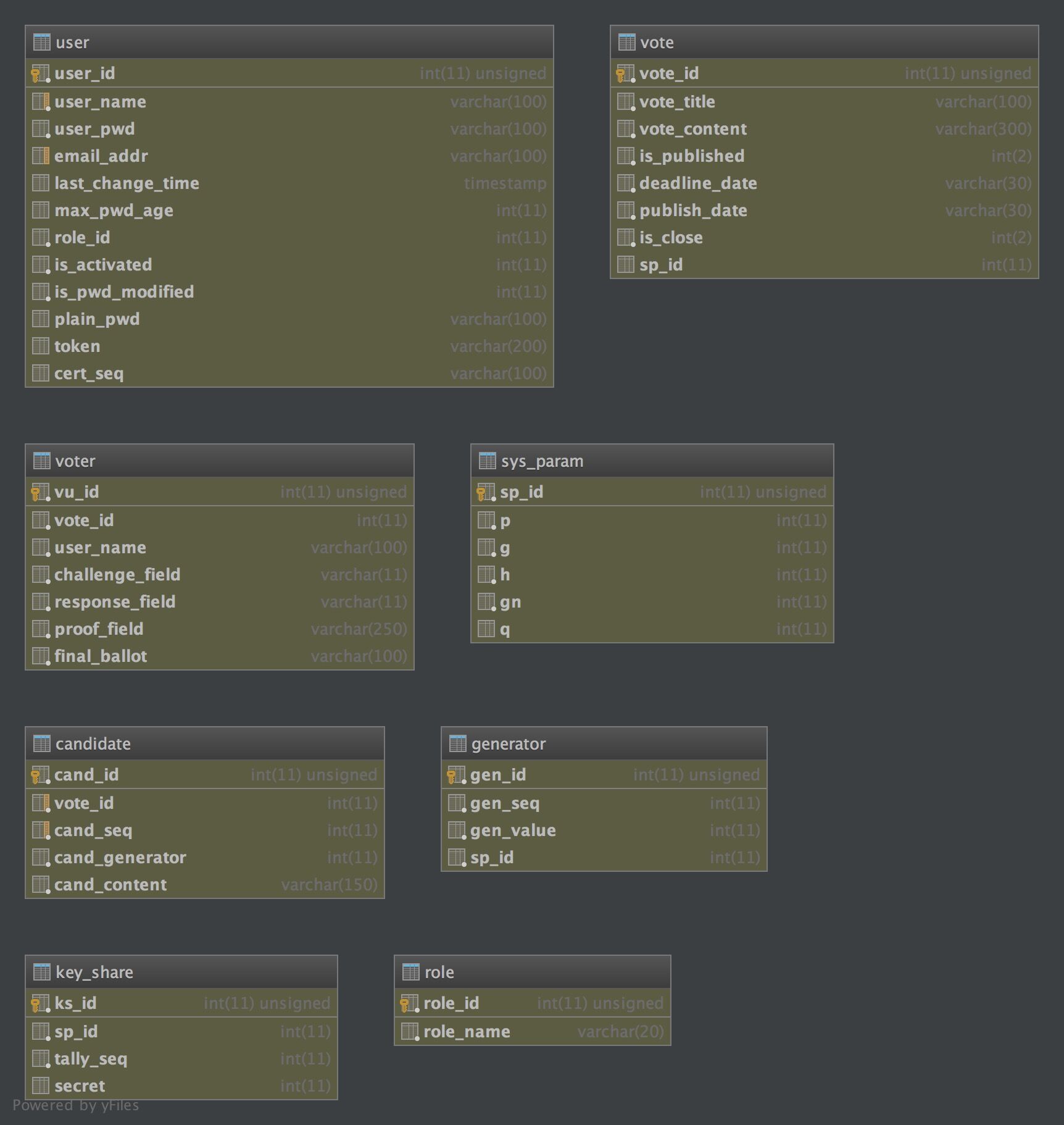


Figure 3.7 Database Physical Design

### Software Interface

#### Software Interface at Main Server

-----------System setup Start--------------

1.getGN

input:

-P: <long long int>: system prime number

outout:

GN: <long long int>: number of generators of Gp

c++:

GenNo

2.getGs

input:

-P: <long long int>: system prime number

outout:

Gs: <long long int Matrix[GN][2]>: generator list of Gp

c++:

Genlist

3.getg

input:

-Q: <long long int>: system prime number

outout:

g: <long long int>: a generator of Gq

c++:

sp\_g

4.geth

input:

-Q: <long long int>: system prime number

-P: <long long int>: system prime number

-s: <long long int>: system secret key

outout:

h: <long long int>: system public key

c++:

g\_h

5.getGset

input:

-GN: <long long int>: number of generators of Gp

-Gs: <Matrix[GN][2]>: generators set of Gp

-K: <long long int>: number of candidates

outout:

Gset: <long long int matrix[K][2]>: generators for candidates

c++:

GenD

-----------System setup End--------------

-----------Shamir secret share Start--------------

1.getsh

input:

sh = sp.split(s, sp.Available, sp.Needed, sp.Q);

-s: <long long int>: system secret key

-Available: <int>: (k,n)-->n

-Needed: <int>: (k,n)-->k

-Q: <long long int>: system prime number

outout:

sh: <long long int matrix[Available][2]>: secret shares

c++:

split

-----------Shamir secret share End--------------

#### Software Interface at Voter

-----------PV1 BEGIN--------------

1.getGi

input:

-item number,

-set Gset array 1

outout:

Gi

C++:

GetGi

2.getAlpha

input:

-Q

output:

alpha

C++:

PV1\_1\_arf

3.genFirstBallot

input:

-P from sys,

-g from sys,

-h from sys

-Gi

-alpha

output:

xy array 1

C++:

PV1\_1

4.sendFirstBallotToHV

input:

-xy

output:

success bool

5.receiveC1fromHV

input:

output:

c1

6.genW1

input:

-Q from sys

output:

w1

C++:

PV1\_3\_w1

7.genDR

input:

-c1,

-w1,

-Q from sys,

-K (candidate amount)

-item number

-alpha

output:

dr array 2 Kx2

C++:

PV1\_3\_dr

8.genAB

input:

-P from sys,

-K(candidate amount)

-item number,

-g from sys,

-h from sys,

-xy

-Gset

-w1

-dr

output:

-AB array 2 Kx2

C++:

PV1\_3\_AB

9.sendDRandABtoHV

input:

-dr

-AB

output:

success bool

10.sendDRtoServer

input:

-dr

output:

success bool

-----------PV1 END--------------

-----------PK BEGIN--------------

1.receiveabandUV

input:

output:

-ab array 1

-uv array 1

2.genC2

input:

-Q from sys

output:

-c2

C++:

PK\_2\_c2

3.sendC2toHV

input:

-c2

output:

success bool

4.receiveRfromHV

input:

output:

-r(response)

5.verifyPK

input:

-P from sys,

-ab,

-uv,

-r,

-c2,

-g from sys,

-h from sys

output:

success bool

C++:

PK\_4

-----------PK END--------------

-----------PV2 BEGIN--------------

1.receiveSTfromServer

input:

output:

-st

2.genFinalBallot

input:

-P from sys,

-uv,

-xy

output:

Fballot matrix 1x2

C++:

PV2\_2

3.verifyFinalBallot

input:

-P from sys,

-g from sys,

-h from sys,

-dr,

-Fballot,

-Gset,

-st

-K

-uv

-Fballot

output:

success bool

C++:

PV2\_3

4.sendFBallotToServer (if verification passed)

input:

-Fballot

output:

success bool

#### Software Interface at HV Server

-----------PV1\_BEGIN--------------

1.receiveFirstBallot

outout:

first ballot

input:

2.genC1

input:

-Q from sys\_param

outout:

challenge value c1

c++:

PV1\_2

3.sendC1toVoterAndServer

outout:

success bool

input:

c1

4.verifyFirstBallot

input:

-xy first ballot array 1,

-dr, ab array 2 from sys\_param,

-K candidate amount

-g from sys

-P from sys

-Q from sys

-h from sys

-subset of generator Gset

outout:

verification result of first ballot(bool)

c++:

PV1\_4

-----------PV1\_END--------------

-----------PK\_BEGIN--------------

1.genBeta

input:

-Q from sys

output:

beta

c++:

PK\_1\_bta

2.genW2

input:

-Q from sys

output:

w2

c++:

PK\_1\_w2

3.genUV

input:

-P from sys,

-g from sys,

-h from sys,

-beta

output:

uv array 1

C++:

PK\_1\_uv

4.genAB

input:

-P from sys,

-g from sys,

-h from sys,

-w2

output:

ab array 1

C++:

PK\_1\_ab

5.sendUVandABtoVoter

input:

ab,

uv

output:

success bool

6.receiveC2fromVoter

input:

output:

c2

7.genResponse

input:

-c2,

-P from sys,

-beta,

-w2

output:

r (response) long long

C++:

PK\_3\_r

8.sendResponse

input:

-r

output:

success bool

-----------PK\_END--------------

-----------PV2\_BEGIN--------------

1.genST

input:

-P from sys,

-K (candidate amount)

-uv,

-ab,

-dr

output:

st array 2 Kx2

C++:

PV2\_1

2.sendSTtoServerandHV

input:

-st

ouput:

success bool

#### Software Interface at Tallying Server

-----------step 1: ballots management--------------

1.genXYs

input:

-P from sys\_param

-times Nb/4

-res Nb%4

-Allballots from sys\_param

outout:

XYs: ballots product Matrix times+1 x 2

c++:

t\_msg

-----------step 2: secrete share scheme--------------

1.NumberofTallyingAuthorities

input:

-Available from sys

-Needed from sys

outout:

Nt

c++:

NumT

2.chooseTA

input:

-Nt

-Available from sys

outout:

ptp: Matrix 1 Nt

c++:

PT

3.genlm

input:

-Nt

-Q from sys

-ptp

outout:

lm: Matrix 1 Nt

c++:

lmta

-----------step 3: distributed calculation and tallying-------------

1.centralTallying

input:

-XY matrix 1 2 from central tallying

-cm matrix 1 Nt wi from sys

-K from sys

-voters long long int ballots number

-P from sys

-Gset from sys

output:

t\_r matrix 1 K tallying result

c++:

ctly.tallying

-----------step 3: distributed calculation and tallying-------------

1.DistributedCalculation

input:

-XY matrix 1 1 x 2 from central tallying

-P from sys

-Q from sys

-sh long long int from central tallying

-lm long long int from central tallying

output:

cm[i]: Wi

c++:

part\_tallying

#### Main Server Public Interface

-----------Administrator\_BEGIN--------------

1. /login

input:

- username, - password

output:

- success bool

2. /add\_user

input:

- username, - email

output:

- success bool

3. /get\_user\_list

input:

output:

- success bool

4. /activate\_user

input:

- username

output:

- success bool

5. /disable\_user

input:

- username

output:

- success bool

6. /add\_users

input:

-user\_list: [{username, email}, {username, email}, {username, email}]

output:

- success bool

7. /system\_init

input:

output:

- success bool

8. /propose\_vote

input:

- title

-content,

-deadline,

-publish\_date,

-candidates:[{ candname}, {candname}, {candname}]

-eligible\_voters:[{username}, {username}]

output:

success bool

9. /delete\_vote

input:

- vote\_id

output:

- success bool

10. /get\_vote\_list\_m

input:

output:

- success bool

11. /get\_vote\_m

input:

- vote\_id

output:

- success bool

----------- Administrator \_END--------------

-----------Voter\_BEGIN--------------

1. /modify\_pwd

input:

- username, - password

output:

- success bool

2. /get\_vote\_list

input:

output:

- success bool

3. /get\_vote

input:

-vote\_id

output:

- success bool

----------- Voter\_END--------------