Project 2 Secure Election System

Vidal Zazueta, Nick Janes, Mingyang Liu

### Project Description

This project implements a secure and auditable electronic voting system leveraging two centralized authorities: the **Central Legitimization Agency (CLA)** and the **Central Tabulating Facility (CTF)**. The goal is to ensure that each voter can only vote once, while maintaining the integrity of the voting process by making votes anonymous and encrypted. The encryption method we used is RSA, where we used the same functions that were developed in Project 1 Secure Socket Layer. The Secure Election protocol must follow several key principles:

* Only authorized voters can vote before the election ends
* No one can vote more than once
* No one can determine for whom anyone else voted
* No one can duplicate anyone else’s vote
* Every voter can make sure that their vote has been taken into account in the final tabulation
* Everyone knows who voted and who didn’t

The **CLA** is responsible for **voter validation**. Before casting a vote, a voter must request a **validation number** from the CLA. This number acts as an anonymous token that certifies the voter's eligibility. Upon request, the CLA generates a unique random validation number and links it to the voter internally without revealing their identity to any other party. The CLA will add the voter to the CLA’s voter list for future use, which will be discussed further. Afterwards, the CLA sends the validation number to the CTF. The CLA will have 3 command arguments; the first argument is the CLA’s port number, the 2 argument is the CTF host address, and the 3 argument is the CTF port number. All three arguments will be numbers. The CLA requires listening on a port number because it acts as a server for the voter, while it needs the CTF host and port number because it will then act as a client and send the validation number to the CTF. A key feature in our implementation is that we attached a string “CLA” to the validation number that is sent to the CTF. The purpose is for the CTF to understand where it is getting the validation number from, to avoid any synchronization issues between voters and the CLA. The CLA will decrypt any incoming voter requests with its private keys and will send the encrypted validation numbers using the voter’s public keys. Various security checks are implemented, such as finding the voter’s file, checking for duplicate validation numbers, verifying that a voter’s password matches, and preventing a voter from registering twice.

The CTF is responsible for tallying the votes of the displayed candidates. The arguments for the CTF only consist of a port number where the CTF will listen for connections. The CTF will have a list of the validation numbers it received from the CLA and the list of candidates in the election. The CTF will decrypt any incoming packets with its private keys to get the plaintext. A key component in the CTF architecture is parsing the initial text of the plaintext to decide its next action as it is receiving packets from 2 clients. If the initial letters contain “CLA,” it indicates that the current packet came from the CLA; the CTF will call a method to get the validation number from the CLA. This method will iterate over the CTF’s validation number list to see if the received number is in the list or needs to be added. If the initial letters do not contain CLA, it indicates that the packet received is coming from a voter who cast their vote. The CTF will then call a method to process the vote using the validation number that the voter used to vote. The CTF will notify the voter if the validation number they used to vote is invalid and will also notify them if their vote has been successfully counted. Upon a successful vote, the CTF will cross out that validation number to prevent a verified voter from voting twice.

Finally, the voter is responsible for requesting a validation number from the CLA and sending their validation number and vote to the CTF to vote. The voter takes the arguments of the port and host numbers of the CTF and CLA, followed by the action. The action is the integral piece to our architecture as it determines what actions the voter will take. If the action is “getValidation,” the voter will use a method to get the validation number from the CLA. Once the user has their validation number, the next action they can use is “castVote candidate name,” which will call a method to cast their vote for the selected candidate. The voter will have to input their validation number in order to cast their vote. The voter encrypts it’s packets with the public keys of the CLA or the CTF, depending on the actio,n and decrypts packets using their private key.

At the conclusion of the election, the CTF publicly releases the final tally, including the number of votes per candidate. This separation of responsibilities between CLA and CTF ensures anonymity, prevents double voting, and enables transparent result publication.

### Development Environment

The development environment used Java as the programming language, followed by the IDE Visual Studio Code. The code was kept in a repository on GitHub, and all development was done by merging different branches with the main branch.

### How to Run the Program

To run the program, you must first open a command prompt and then navigate to the location where the folder Project 1 is stored. Upon entering the folder, you can run all of the programs after you compile them. To compile the files, input the command: javac <fileName>.java for each file. After compiling the files, you can run them as follows:

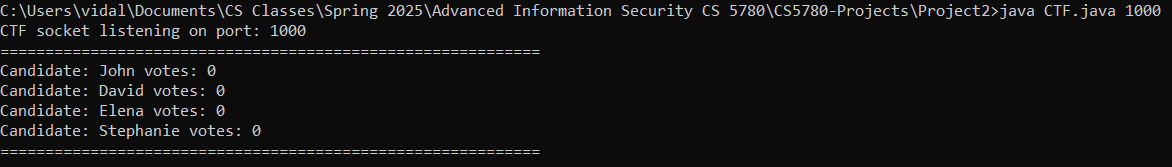
Starting with the CTF file, running the arguments **java CTF.java <port number>** will start the file. The port number is the user-defined port number where the CTF will listen to.

Moving forward with the CLA file, running the arguments **java CLA.java <port number> <CTF Host Address> <CTF port number>** will start the file. The port number in the first argument is the port where the CLA will listen. The following arguments are the host address, typically 127.0.0.1, and the port number are used to connect to the CTF.

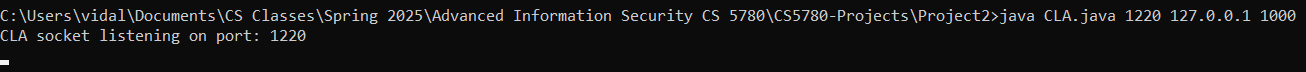
Finally, we have the voter file. The voter file will use different methods depending on the action given by the voter, which is either “getValidation” or “castVote.” The arguments **java Voter.java <CLA\_Host> <CLA\_Port> <CTF\_Host> <CTF\_Port> <Voter Name> <Action = getValidation>** is one way the voter can get the validation number. With the arguments above, the voter will need to input the CLA and CTF host address and port numbers to be able to communicate with them. The action “getValidation” followed by the voter's name, will communicate with the CLA to receive a validation number with the associated voter name. Another set of arguments the voter can use is **java Voter.java <CLA\_Host> <CLA\_Port> <CTF\_Host> <CTF\_Port> <Voter Validation Number> <Action = castVote> <Candidate Name.** With the arguments above, the voter will need to port and host addresses of the CLA and CTF to send its vote to the CTF. The voter will need to input the validation number they received from the CLA and use the action “castVote” to vote on their preferred candidate.

### Screenshots

For the input **java CTF.java 1000,** we get the output:

****

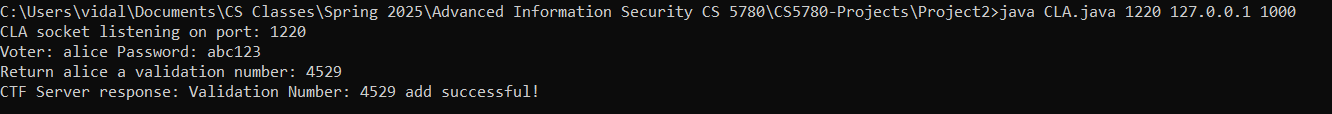
For the input **java CLA.java 1220 127.0.0.1 1000,** we get the output:



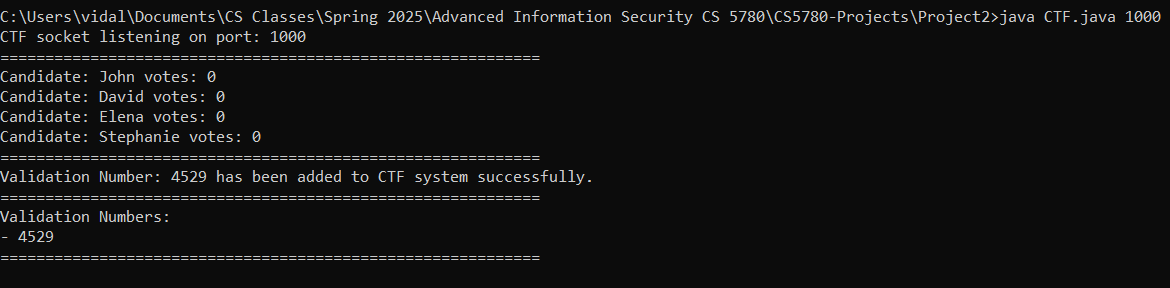
For the input **java Voter.java 127.0.0.1 1220 127.0.0.1 1000 alice getValidation,** we get the output on the Voter side:



On the CLA side, we get the output:



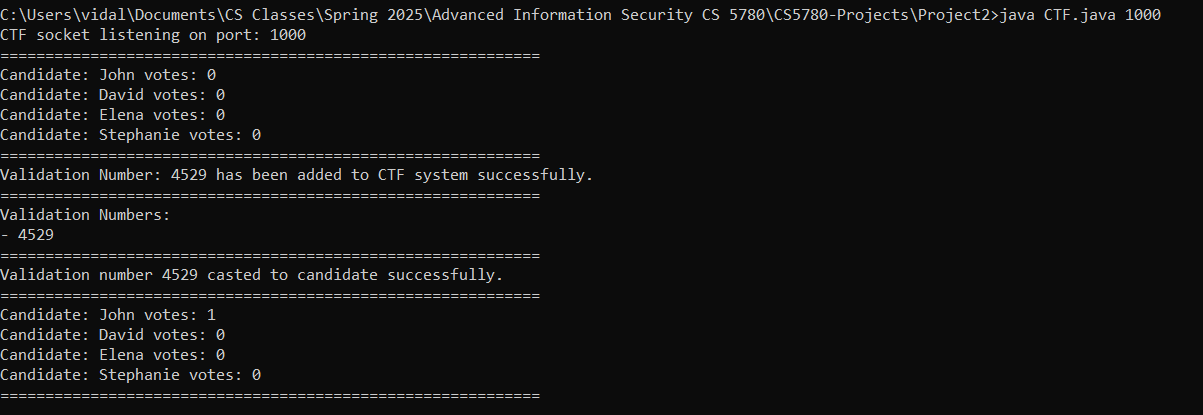
On the CTF side, we get the output:



For the input **java Voter.java 127.0.0.1 1220 127.0.0.1 100 4529 castVote John** we get the output on the Voter side:



On the CTF side we get the output:



### Member Contribution

All members contributed equally