Electronic Voting

Mohammed Almeshekah, Sam Kerr

April 24, 2015

Contents

1 Introduction **4**

2 Scenario Description **5**

3 System Description **6**

3.1 Assumptions 6

3.1.1 Physical Facilities 6

3.1.2 Network Security 6

3.1.3 Communication Robustness 6

3.2 The Voting protocol 7

3.3 Detailed Protocol Steps 8

3.4 The Networking Specification 9

3.4.1 The Lexical Blocks 9

3.4.2 Voter to CTF Messages 10

3.4.3 CTF to Voter Messages 10

4 Security Analysis of the Protocol **11**

4.1 The Preliminary Phase 11

4.2 The Voting Phase 12

4.3 The Results Phase 12

4.4 Summary of the Security Analysis 13

5 Implementation **14**

5.1 Cryptographic Classes 14

5.2 Communication Class 14

5.3 The Database Class 15

5.4 The CTF 15

5.4.1 CTF Local Data 15

5.4.2 CTF Processes 15

5.5 The Voter 16

5.5.1 Voter's Local Data 16

5.5.2 Voter Processes 16

6 Performance Analysis **17**

6.1 Client Analysis 17

6.2 CTF Analysis 17

7 Testing **18**

7.1 Networking Interface 18

7.2 Client Testing 19

7.2.1 Voting Stage 20

7.2.2 Vote Revealing Stage 21

7.3 CTF Testing 22

7.3.1 Registration Stage 22

7.3.2 Voting Stage 24
7.3.3 Vote Revealing Stage 25

8 Limitations **28**

9 Future Work **29**

10 Conclusion **30**

11 Appendices **31**
11.1 Profiling Images 31
11.2 SQL Schema 32

1 Introduction

Electronic voting is one of the most interesting applications of modern cryptography. It integrates a wide selection of cryptographic tools into one coherent secure system that serves millions of users. We have decided to delve deep in implementing this project and examine how different parts of cryptography can build on each other to solve a real world problem.

2 Scenario Description

The nature of the scenario allows us to make various assumptions and rules out certain options. As such, it is important to explicitly describe how we envision our system being used.

Our system will consist of a central tabulating facility (CTF) and various different polling places throughout the country. Voters will go to a polling place to register and to vote, in a similar way to current elections.

Voters will go to their polling place to register. Upon registering, they will be given a hardware voting token that contains some storage space. It is important to note that these tokens are all blank before they are used. Every token is identical before the election and the voter can delete it's contents if necessary as well. The voter then uses his token, goes to a polling place, and performs the registration process with the CTF. This involves sending his full name and social security number, so the CTF can record who has registered and who has not. The token then selects an ID number from a list of available IDs transparently to the voter. Note that this ID number does not correspond to an actual user. The voter then goes home until the voting period begins.

When the voting period begins, the voter returns to his polling place with his voting token. He does not check in or identify himself in anyway to the staff, but rather, simply goes to a voting booth, inserts his token, and completes his vote. The votes are stored under encryption, so it is not possible for the CTF to know the progress of the election until it is over. Each voter is also given a receipt, signed by the CTF, acknowledging the vote. Note that it is possible for a voter to vote more than once at this point, since the CTF will detect his ID colliding with another users ID number and issue him a new one. However, this will be detected and dealt with later. After the voter has voted, he goes home.

During the voting period, if the voter decides he would like to change his vote, he returns to the polling place and changes his vote using his voting token. He can do this as often as he wants during the voting period. Also, since the CTF cannot decrypt the votes yet, a user wouldn't be able to find out how everyone else is voting and change his own vote based on that.

After the voting period has ended, the CTF needs to collect decryption keys to tally the results. To do this, all users will return their voting tokens, which would still contain their ID numbers and decryption keys. This could be done either by mailing the tokens in or return them to the polling place. Note that since a user has only one identity token, even if he has send more than one vote, he cannot decrypt more than one vote, which means that a single user cannot vote multiple times. Once the CTF has received the voting tokens from all the users, it uses the stored decryption keys to decrypt the votes and tally the results. It can then publish who has won and lost the election.

Note that a user might be concerned his vote was not counted by the CTF. We also address this in the following way. At the end of the voting period, the CTF will publish the votes and corresponding ID numbers for all voters to see. If a user sees that his vote is absent or miscounted, he can take the following steps. Before a user returns his voting token, he will print off the contents of his identity token, including a receipt given to him by the CTF during voting. He will then send this printoff to the CTF, which will prove that he had sent the following vote, since it was previously signed by the CTF.

3 System Description

The interaction between the CTF and the users will be implemented locally using socket programming. The users' details (RSA encryption keys, the identifier I, and voting and registration receipt) will be saved at the users' tokens.

3.1 Assumptions

In our project, we made several different assumptions to either clarify the scenario or to ease the implementation. We describe those here.

3.1.1 Physical Facilities

We assume that the CTF is some large central facility with the computational and networking capabilities to handle many simultaneous clients.

We also assume that clients will be connecting from a polling place, rather than from their homes. The polling place will be equipped with enough storage and computational power to maintain the list of available ID numbers, since it will be very large.

We also assume that there will be a staff member at the polling place who will be able to recognize if a voter is submitting multiple "Submit Vote" requests, rather than "Change Vote" requests, which could be used to execute a denial of service attack. This assumption prevents this type of attack from happening.

3.1.2 Network Security

For our scenario, we assume that there is a secure connection between the CTF and client. This could be implemented using either SSL or some other alternative. As such, we are not concerned with man in the middle attacks.

Since we are assuming the connection is secured, we omit any implementation to secure the connection itself with signatures, MACs, et cetera. Any cryptography in our project is therefore to ensure privacy and security in the voting protocol itself. This assumption helps to simplify the implementation and reduces development time needed.

3.1.3 Communication Robustness

In addition to assuming the connection between the CTF and client is secure, we also assume that the communication channel is robust. That is, any messages between the client and CTF will be well formed and not have erroneous information sent. By making this assumption, we are able to omit much of the code needed to verify the quality of messages and ensure that they are of the proper format. Instead, the only checking needs to be done to verify the values sent in the message, but not the message itself.

This simplifies the implementation and reduces development time. In a production environment, this assumption would need to be discarded, but it is acceptable in this situation.

3.2 The Voting protocol

1. **The CTF publishes a list of all eligible voters.** The CTF will receive the list of all eligible voters and store their details in its database. This list will contain the user's full name and their SSNs. Note that only the CTF will be able to view this list, since voter's should not be able look at other voter's SSNs.

2. **Before a deadline, each voter tells the CTF whether she intends to vote.** Each voter will check whether she is an eligible voter by inquiring the CTF and providing her SSN. If so, the voter tells the CTF whether she wants to vote. In addition, the server will send a signature on the user's SSN and send it back to the user. This will be used later by the user to prove that she has registered to vote and that the CTF acknowledged it. This signature must be time stamped, too.

3. **At this deadline, the CTF publishes a list of all eligible voters participating in this election.** The CTF publishes a list of all registered voters as a file listing the SSNs of the registered users. This list must be timestamped and signed by the CTF and stored at a safe location. In addition, the CTF creates a list of possible identifiers (Is) and encrypt them under its Mental Poker encryption key and makes the list available to all voters.

4. **Each voter receives a unique identification number I.** If a user is a registered voter, the voter's agent will receive the signed list of encrypted (Is) and randomly select one of them on behalf of the user (lets call it $E_{K_{CTF}}(I_i)$). The voter generates a Mental Poker key pair. After that, the voter agent will encrypt as the following $E_{K_C}(E_{K_{CTF}}(I_i))$ and sent it back to the CTF. The CTF will decrypt it and send it back to the voter. The voter can then decrypt it using her keys and retrieve the identifier I.

5. **Each voter send her encrypted vote anonymously to the CTF.** Each voter's agent will generate a public/private RSA key pair *k, d.* If her vote is v, she sends the message $I, E_k(I, v)$ to the CTF.

6. **If the CTF detects a conflicts in identifiers sent along with the vote, it resolves that immediately.** This is resolved by the CTF where it will ask the voter to select another ID and send her vote again. The user's agent should detect that and re-obtain a new ID from the CTF, lets call it I‘. After that, the voter's agent will reconstruct the vote as $I‘, E_k(I‘, v)$ and send it to the CTF.

7. **If there is not conflict in the identifiers, the CTF acknowledges receipt of the vote.** The CTF saves the pair $I, E_k(I, v)$ in the database and acknowledge that to the voter by sending a voting receipt. This receipt will be the CTF signature on the pair $I, E_k(I, v)$ that has been received from the voter. This can be used later by the voter to prove that she has voted and the CTF have successfully received her vote.

8. **If a voter wants to change his vote from v to v', she sends I,$E_k(I, v')$ to the CTF.** The CTF then updates the stored votes with the new vote

9. **After the deadline of the voting period, each voter sends the message I,d to the CTF.** This step is needed such that the CTF have the necessary information to decrypt the previously received votes and publish the results. In addition, the I's need to be sent with each decryption key to use the right key for the right vote.

10. **The CTF decrypts the votes and publishes the results of the voting.** For each different vote, the list of all $E_k(I, v)$ values which contained that vote will be published.

11. **If a voter sees that his vote is not properly counted, she protests.** The protest can be done by providing the voting receipt previously received from the CTF as an proof of a vote that has not been counted.

3.3 Detailed Protocol Steps

1. **The CTF publishes a list of all eligible voters.** This is done by referencing the lists of available population, such as census, tax records, etc.

2. **Before a deadline, each voter tells the CTF whether she intends to vote.** This is done by a voter mailing a slip in, coming to a polling place, or something similar.

3. **At this deadline, the CTF publishes a list of all eligible voters participating in this election.** This is done by the CTF listing the SSNs of all users who have previously registered.

4. **Each voter receives a unique identification number I.** This is done by each user coming to the polling place and acquiring a registration token. Then, each user executes the mental poker protocol to confidentially receive an ID number. This ID number, as well as the user's RSA and mental poker key pairs are stored on the identity token.

   Additionally, the CTF signs the SSN of the user, which confirms that he did in fact register previously.

5. **Each voter send her encrypted vote anonymously to the CTF.** This is done by sending $I, E_C(I, V)$ to the CTF. The CTF will responds by signing the vote and returning a receipt to the user. This can be used to show that the CTF did in fact receive the vote if it tries to deny it later.

6. **If the CTF detects a conflicts in identifiers sent along with the vote, it resolves that immediately.** The CTF will respond to the client to inform him of a conflict and assign him a new ID number for a special pool that the CTF previously had generated. The user notes this new ID number and uses it for all future communications.

7. **If there is not conflict in the identifiers, the CTF acknowledges receipt of the vote.** The CTF will responds by signing the vote and returning a receipt to the user. This can be used to show that the CTF did in fact receive the vote if it tries to deny it later.

8. **If a voter wants to change his vote from v to v', she sends I,E_k(I,v'),d to the CTF.** The CTF will update the database with the corresponding new vote. This could be a potential vulnerability, which we address later in the security analysis.
9. **After the deadline of the voting period, each voter sends the message I,d to the CTF.** Send the decryption keys is necessary for the CTF to tally the votes. For our scenario, each user physically returns, either by mailing to the CTF or in person at the polling place, his voting token. As such, he can only decrypt one vote.
10. **The CTF decrypts the votes and publishes the results of the voting.** This shows who the winner of the election is. It also shows which ID numbers voted for whom.
11. **If a voter sees that his vote is not properly counted, she protests.** Since the user will have a receipt of the vote she sent in, it will be possible for her to prove that the CTF miscounted her vote.

3.4 The Networking Specification

3.4.1 The Lexical Blocks

- QUESTION := "?"
- AMP := "&"
- ID := "ID="
- EXPONENT := "EXPONENT="
- MODULUS := "MODULUS="
- NAME := "NAME="
- SSN := "SSN="
- RCPT :="RECEIPT="
- CIPHERTEXT := "CIPHERTEXT="
- IV := "IV="
- SYMKEY := "SYMKEY="
- CIPHERBLOCK := CIPHERTEXT NUMBER AMP SYMKEY NUMBER AMP IV NUMBER
- STRING := STRING [a-zA-Z0-9]
- NUMBER := NUMBER [0-9]
- MESSAGE := "ACCEPT:" STRING || "FAIL:" STRING
- STATUSBLOCK := "STATUS=" MESSAGE ""

3.4.2 Voter to CTF Messages

- Reveal := ”REVEALVOTE” QUESTION ID NUMBER AMP EXPONENT NUMBER AMP MODULUS NUMBER
- IdEnroll :=”IDENROLL” QUESTION ID NUMBER
- CheckValid := ”CHECKVALID” QUESTION NAME STRING AMP SSN STRING
- RegisterReq := ”REGISTERREQ” QUESTION NAME STRING AMP SSN STRING
- Submit := ”SUBMITVOTE” QUESTION ID NUMBER AMP CIPHERBLOCK
- Protest := ”PROTEST” QUESTION ID NUMBER AMP EXPONENT NUMBER AMP MODULUS NUMBER AMP CIPHERBLOCK
- ChangeReq := ”CHANGEREQ” QUESTION ID NUMBER AMP CIPHERBLOCK

3.4.3 CTF to Voter Messages

- ProtestResponse := ”PROTESTRESPONSE” QUESTION STATUSBLOCK
- CheckResponse := ”CHECKRESPONSE” QUESTION STATUSBLOCK
- RegResponse := ”REGRESPONSE” QUESTION STATUSBLOCK AMP RCPT
- RevealResponse := ”REVEALRESPONSE” QUESTION STATUSBLOCK
- IdResponse := ”IDRESPONSE” QUESTION STATUSBLOCK AMP ID NUMBER
- SubmitResponse :=”SUBMITRESPONSE” QUESTION STATUSBLOCK AMP RCPT NUMBER || ”SUBMITRESPONSE” QUESTION STATUSBLOCK AMP ID NUMBER
- ChangeResponse := ”CHANGERESPONSE” QUESTION STATUSBLOCK AMP RCPT NUMBER

4 Security Analysis of the Protocol

Our protocol is based on the protocol specified in [1] and [2] which generally consists of three major phases; namely the preliminary phase, the voting phase and the results phase. In this section we will analyze the security of these three phases pointing out to what security mechanisms we have used.

4.1 The Preliminary Phase

In this phase the CTF publishes the list of all eligible voters and store them in the database. In addition, the CTF will create two lists of voting IDs; one of these list available for users to choose from, and the other one will be saved internally at the CTF side to resolve the potential conflict in the users IDs. The public list will save the IDs in an encrypted form under the CTF's Pohlig-Hellman encryption key (which has a modulus length of 4096-bit).

When an eligible user wants to register for voting she will send a command to the CTF asking for registration where she will receive a receipt from the CTF proving that she has successfully registered to vote and the file of the encrypted IDs to choose from. This receipt will be a CTF signature on the voter's SSN plus a time-stamp. After that, the voter has to randomly select and ID from the list and encrypt it under his Pohlig-Hellman encryption key and send that double encrypted ID to the CTF to decrypt it under its Pohlig-Hellman decryption key and send it back to the voter where it will be now only encrypted under the voter's Pohlig-Hellman encryption key. Finally, the voter can decrypt that ID and then she will be ready to vote when the voting phase start.

At the end of this phase the CTF will publish a signed time-stamped file of the list of registered voters that can be save at an external place. In addition, all the communications between the CTF and the voters can be save on an append-only log file.

The main purpose of having this preliminary phase is to limit the CTF ability to add fraudulent votes during the voting period [2]. However, this does not prevent the CTF from adding a fraudulent votes for voters whom registered to vote but then they did not vote during the voting period. In addition, in this phase each voter will receive a receipt that prove that she has registered at a given point of time which can be used to limit the ability of the CTF denying some legitimately registered voters from voting during the voting period.

In this stage we are the Pohlig-Hellman encryption algorithm to encrypt/decrypt the IDs where we exchange these IDs using the Mental Poker protocol. We utilize the commutative feature of the Pohlig-Hellman protocol to decrypt the double encrypted ID by applying the CTF decryption process in any order. By doing that we preserve the anonymity feature of in our protocol where the CTF will not be able to link the voter's identity with her ID despite that it can link the ID decryption process with the registration process as this linkage is needed to prevent a user from obtaining an ID without registering or obtaining and ID twice. This de-linking is achieved because when the CTF decrypts the double encrypted ID it will have the ID encrypt under the voter's key and the CTF will not be able to decrypt it in a way better than brute-forcing the voter's keys which is computationally infeasible.

It worth noting that the number of possible IDs should be larger enough to minimize the potential of having collisions between different users. This is one of the area that we discuss further in the future work of our project. On the other hand, the number of secondary can be at maximum equal to the number of possible users in the system where we can resolve collisions for every users in the worst case. However, on average having IDs that are half of the number of users is an average reasonable selection.

4.2 The Voting Phase

This is the main stage of the voting protocol where voters will submit the encryption version of their votes to the CTF. In addition, during this phase voters can request to change their vote to a new one and provide the encryption form of the new vote.

When a voter wants to vote she will send her ID and the encryption of the ID concatenated into the vote to the CTF. The CTF will check that and ID is valid, if so the encrypted vote will be added to the database along with the corresponding ID. If the vote has been added successfully to the database the CTF will generate and sign receipt as a proof of voting to the voter.

Each voter will have an RSA key pair which will be used to encrypt the payload of the vote (ID concatenated into the vote) using hybrid encryption. That is, the RSA encryption key will be used to encrypt a randomly generated symmetric key, and then this key will be used to encrypt the payload using AES in CTR mode where the IV will be randomly generated for each different encryption. After that, the voter will triple of the RSA encrypted symmetric key, the AES encrypted content and the IV as a whole cipher block to the CTF.

At the CTF side, it will check that the ID is valid and store this whole cipher block in the database. After that, it will generate a signature on the voter ID concatenated into her cipher block and send that signature to the voter.

It worth noting that there is a probability that two IDs will collude, which means that there are two voters who selected the same ID from the CTF list of valid IDs, and when that happens that will be resolved at the CTF side in the following way. We have mentioned earlier that the CTF has another list of ID that are not offered to the voters and it is kept at the CTF side to resolve collision. When a voter submit her vote the CTF first check if there is a collision and if so the CTF will select an ID from the secondary list and send back the voter asking her to resend her encrypted vote by including this new ID. On the voter side, she will detect that and redo the submission process using this new ID provided by the CTF. It worth noting that this will also not link the vote to the voter's ID because the collided ID is also anonymous and we only can link the new ID to the collided one.

If the voter want to change her vote, it will perform the exact same process of submitting vote and send the ID and the cipher block to the CTF. However, the CTF will update the database with the voter's new vote deleting the previously stored vote and will generate a new signature on the new vote.

4.3 The Results Phase

After the voting process ends, each voter will send her ID and her RSA private key to the CTF where the votes can be decrypted. At the CTF side, it whenever it receives a new pair it will extract the cipher block from the database, decrypt the vote and store the decryption key and the vote to the database. When all the votes has been decrypted the CTF will publish a file that contains the following triple for each ID: the vote, the encrypted vote and the decryption key.

After the results in the file has been published the voter can check if her voter has been counted successfully. If not, she can protest by sending her ID, encrypted vote, the decryption key and the receipt to the CTF. At the CTF side, it will check that the receipt is valid be verifying the signature, if so it checks whether the it has miscounted the vote or didn't even receive the vote and add to the database. In both cases it will resolve them by updating its database and republishing the results.

4.4 Summary of the Security Analysis

We believe that we have managed to build and implement the core for a secure voting system that addresses the major problems in the electronic voting system as discussed in [2] and [1]. However, there are some other security features that can only be addressed by logistics and managerial activities. For example, storing the voters' IDs and keys and protesting that a receipt is not valid. Therefore, it is not enough to specify the protocol and the security requirements of the e-voting system to ensure its security, we need to address all the infrastructure assumptions and managerial activities that need to be implemented in order to have a secure e-voting system.

5 Implementation

For language, we are using Java. There was some concern that using Java and the built in BigInteger classes would not be sufficiently fast, as compared to another solution such as GMP, but this did not appear to be a major problem. More details are in the timing section of the paper.

We did not want to invest a lot of time reimplementing many common algorithms such as RSA, AES, etc. As such, we used a library called BouncyCastle, which contained implementations of these algorithms. There is more information about this library at www.bouncycastle.org. A particularly interesting feature about this library is that various parts of it are FIPS certified, meaning that the implementations have been verified to be correct. We used the BouncyCastle library to implement wrappers to help make our coding easier.

For the storage layer, we used the MySQL database. It is straightforward to use and high quality tools are readily available, so this was a good choice. We did not run into any major problems with it either.

We implemented our networking layer using Java Sockets and ServerSockets. For the CTF, the networking layer would spin off a new thread as each client connected and would handle it seperately from all other threads. Clients would run one thread to connect to and manage communication with the CTF. Additionally, our networking layer provided a blocking send()/receive() abstration to ease programming.

The GUIs were made using Java Swing. Swing is fairly simple to use and is a native solution for Java, so it was an easy choice.

We also used the Apache Common Lang library as helper functions at various stages in our project.

Finally, the CTF is required to post lists in public places for all voters to see, such as the voting results and the list of mental poker encrypted ID numbers. In our scenario, each polling place would hve a copy of the ID list and the voting results would be posted on a website. For our implementation of this, we created a file that both the CTF and the voter classes can access.

Below, we provide a more in depth summary of the various parts of our implementation.

5.1 Cryptographic Classes

To use cryptography, we wrote two classes, one an RSA helper class and one a mental poker wrapper class. It was much easier writing a wrapper around the BouncyCastle code, rather than duplicating the complex ways of calling BouncyCastle.

The RSA class encapsulated functionality such as key pair generation, encryption, decryption, signing, and signature verification.

The mental poker wrapper encapsulated a simple encrypt()/decrypt() functionality. Note that since mental poker operations commute, encrypt/decrypt could be used in either order.

5.2 Communication Class

The networking layer provides a communication interface between the CTF and client. Two classes are provided, one for clients that connects to a server and communicates with it and one for the CTF that listens on a given port, accepts clients, and spins them off into their own thread.

Both classes provide a send()/receive() abstraction. Receive() is blocking in both cases and will not return until a message has been received. The send() method for client's simply takes a String and sends it to the CTF it connected to. The send() method for the CTF takes a String as well as an int that represents a specific client. It then sends the message to the designated client. Note that send() is not blocking in both cases.

5.3 The Database Class

We used MySQL for our database. The abundance of examples and tools available made it a very easy platform to work with. We created three different tables, one for ID numbers, one for voters, and one for votes. The ID table simply stores the plaintext ID numbers available for use. The voters table stores the list of eligible voters for the election and marks if they have registered or not. The third table, votes, stores an ID number, encrypted vote, as well as a decryption key and the plaintext vote.

Our storage layer is encapsulated inside of our DB class. This class provides an API to the CTF so that it can simply execute requests that will automatically be handled appropriately at the database level. This allowed us to focus on writing an abstraction layer above the database code once and then using our abstraction layer whenever we needed to interact with the database.

5.4 The CTF

The CTF has a very large share of responsibility in the voting scenario. It is responsible for maintaining votes, user databases, ID numbers, and more. As such, it has quite a large amount of local data and processes.

5.4.1 CTF Local Data

The CTF is responsible for storing multiple different pieces of data, both in memory and using more permanent structures such as a database.

In memory, the CTF stores an RSA and mental poker keypair, the list of candidates, and session IDs of connected clients.

The CTF also uses a MySQL database to store the list of plaintext ID numbers, eligible users, registered users, and votes, which consists of an encrypted vote, ID number, decryption key, and plaintext vote.

5.4.2 CTF Processes

The CTF provides both a GUI process and command line diagnostics to administrators. Internally, it is also running many other processes.

The CTF GUI allows the administrator to change the time periods of the election, from registration, to voting, to the reveal stage. The GUI also allows an administrator to tally the votes and publish results.

Internally, a large part of the CTF is focused on maintaining network connections. The previously described networking layer maintains a list of connected clients and identifies them with session IDs. The CTF also internally processes data and retrieves or stores it in the database. The CTF also leverages various cryptographic tools at various stages of the election, to generate receipts, decrypt votes, or verify signatures.

5.5 The Voter

The voter classes contain everything that is used to communicate and interact with the CTF. It provides a GUI interface which clients can use, prints diagnostic information on the command line, and internally handles all cryptographic and other processes.

5.5.1 Voter's Local Data

The voter application will be responsible for maintaining several different pieces of information.

The networking code will maintain session IDs, so that a user can properly maintain a communication state with the CTF.

The client also stores an RSA key pair and a mental poker keypair, both of which are used at different points. Additionally, RSA parameters are stored, so that the CTF can configure RSA in the same way that the client was using it.

Signed receipts from the CTF are also stored, so that a client can prove that he has registered or voted.

5.5.2 Voter Processes

There are several actions that the voter interface provides to end users, via the GUI interface. Using the GUI, a voter is able to register with a name and SSN, submit votes, change votes, reveal votes, and protest the vote counts. These are the main pieces of functionality that end users are concerned with. Each of these was tested, with results in the testing section.

Internally, the voter interface is responsible for several other tasks. Specific details can be found in the Javadoc submitted.

Probably the most straightforward internal process is key generation for the RSA and mental poker interfaces.

The voter interface is also responsible for transparently selecting an ID number for a user. This is done by accessing a file of encrypted ID numbers and randomly choosing one with mental poker. If a new ID number is needed later because of a collision, the voter interface transparently handles that as well.

The last internal processes is the networking. This is important as it ensures that clients and CTF are able to communicate, but this happens transparently to the user. He simply starts the application and is connected.

6 Performance Analysis

In the appendix, please find pictures of the profiling data for both CTF and client.

6.1 Client Analysis

For our client profiling, we had a client connect to the CTF, register, vote, change his vote, reveal his vote, protest his vote, and then disconnect.

Our results showed that generating an RSA keypair took 2189 ms and a mental poker keypair took 35.4 ms. Encryption took 1316 ms and 20.2 ms with RSA and mental poker, respectively. Decryption took 353 ms with mental poker. Key generation is somewhat slow, but due to the nature of the GUI, key generation is being done in the background and will finish before a user needs to perform a task.

The networking code accounted for the majority of the client's functioning. It uses a similar networking stack as the CTF, which means that it has two separate threads to handle network sending and receiving. As can be seen from our results, the listening task accounted for 43.7% our total run time. In the future, we could work on having this system wait more efficiently, but busy waiting did not cause any slow down in our program, so we are comfortable with the client performance.

Overall, there is a noticeable pause when registering and voting, but it is a brief few seconds and would not inconvenience users too much.

6.2 CTF Analysis

The CTF is a multi-threaded application. That is, for every client that connects to it, new threads are created. A large part of its job is to do the network communication, in addition to the cryptographic lifting. We ran a test of the CTF with 1 client connected who registered, voted, and revealed his vote. The results of our test showed that the majority of the CTFs time was spent in busy waiting for messages from the client.

Generating a keypair took approximately 3 seconds, while decrypting the client messages took approximately 686 ms with RSA and 409 ms with Mental Poker, respectively. These are all fairly fast operations, and we feel comfortable with their runtime.

7 Testing

Note that in some cases, we omit long strings of numbers, such as receipts and signatures from test cases in the interest of brevity and readability.

7.1 Networking Interface

The networking code uses a Java Socket and ServerSocket at its core. As such, it is using a TCP link. To test this link, we can monitor the traffic from the application using a program such as WireShark to monitor the messages being sent to and from the client application.

The client program executes network transactions when certain buttons in the GUI are pressed. The table below lists the task, expected output, and actual output from our program, as well as a conclusion of whether the expected and actual output match.

Network Test #1
Connect to the server and receive a session ID.
Expected:
Upon connecting to the server, the client should receive the following set of messages.
0:CONNECTED
{ID#}:CONFIG
Actual:
The client received this set of messages once he connected to the CTF
0:CONNECTED
552685592:CONFIG

Test Passed.

Network Test #2
Send a basic message from client to server.
Expected:
{ID#}:HELLO WORLD
Actual:
947533209:HELLOWORLD

Test Passed.

Network Test #3
Send a basic message from server to client. Note that this was done by echoing back the previous message sent from the client.
Expected:
{ID#}:HELLO WORLD
Actual:
947533209:HELLO WORLD

Test Passed.

Network Test #4
Send two messages from client to server. The response from the server should use the same session ID number each time. Note that the server is sent to simply echo back what it receives to the client for this test, which is how we monitor which session ID is being used.
Expected:
{ID#}:HELLO WORLD
{ID#}:HELLO WORLD
Actual:
652695574:HELLO WORLD
652695574:HELLO WORLD

Test Passed.

Network Test #5
Connect two clients to the same server and send either "1234" or "4567" to the server. The server is configured to echo back the messages, along with the session ID number. Each client should receive back his session ID number as well as the message he originally sent to the server. This test is designed to test simultaneous connections to the CTF.
Expected:
{ID_1#}:1234
{ID_2#}:4567
Actual:
1419656789:1234
-1223864479:4567

Test Passed.

The previous tests illustrate that at a basic level, our network layer connects a client and server so they can communicate with each other . As such, we feel confident in our network implementation.

7.2 Client Testing

Client testing is considerably easier than testing the CTF. If the client can successfully generate network messages for the various parts of the protocol, we can be sure that it is properly functioning, since it does not interact with a database or other components.

We present test cases for the registration stage, the voting stage, and the vote revealing stage. These can all three be tested independently.

Registration Stage

This stage involves testing the registration protocol. At this point, we are comfortable with our networking layer, so we will monitor the traffic between client and server to ensure that registration is proceeding as expected.

Registration Test #1
Registration request for valid user "John Smith" and "123456789". A REGISTERREQ message should be generated and sent.
Expected:
ID#:REGISTERREQ?NAME=John Smith&SSN=123456789
Actual:
651287367:REGISTERREQ?NAME=John Smith&SSN=123456789

Test Passed.

Registration Test #2
Registration request for valid user "John Smith" and "123456789". A REGISTERREQ message should be generated and sent. The client should then received a REGRESPONSE message, handle it properly, reset the session, and then execute the IDENROLL stage.
Expected:
ID#:REGISTERREQ?NAME=John Smith&SSN=123456789
ID#:REGRESPONSE?STATUS={ACCEPT:Congratulations, you have been registered.}&RECEIPT=...
ID#:IDENROLL?ID=...
ID#:IDRESPONSE?STATUS={ACCEPT:The ID is decrypted and attached}&ID=...
Using ID number:...
Connected to:...
This connection is:...
0:CONNECTED
ID#$_2$:CONFIG
Actual:
1369725753:REGISTERREQ?NAME=John Smith&SSN=123456789
1369725753:REGRESPONSE?STATUS={ACCEPT:Congratulations, you have been registered.}&RECEIPT=...
1369725753:IDENROLL?ID=...
1369725753:IDRESPONSE?STATUS={ACCEPT:The ID is decrypted and attached}&ID=...
Using ID number:...
Connected to: /127.0.0.1:4444,49507
This connection is: /127.0.0.1:49507
Server: 0:CONNECTED
Server: -1267839115:CONFIG

Test Passed.

7.2.1 Voting Stage

For the voting stage, the important things to test for the client are that vote submission works, vote changing works, and that the client can accept a new ID when issued on. One test cases thus illustrate these three cases.

Voting Test #1
Assuming the user is registered, submit a vote. It will send the vote encrypted using RSA hybrid encryption. The packet will contain the necessary values to load the RSA machine on the server side.
Expected:
ID#:SUBMITVOTE?ID=...&CIPHERTEXT=-...&SYMKEY=...&IV=...
Actual:
241440999:SUBMITVOTE?ID=...&CIPHERTEXT=-...&SYMKEY=...&IV=...

Test Passed.

Voting Test #2
Assuming the user is registered and another user has voted with the same ID number, submit a vote. This should indicate a collision, which the client should process and reset his ID number.
Expected:
ID#:SUBMITVOTE?ID=...&CIPHERTEXT=...&SYMKEY=...&IV=...
ID#:SUBMITRESPONSE?STATUS={FAIL:Sorry, but the ID you're using has been used by another user.}&OTP=ID_{OTP}
ID#:SUBMITVOTE?ID=ID_{OTP}&CIPHERTEXT=...&SYMKEY=...&IV=...
Actual:
241440999:SUBMITVOTE?ID=...&CIPHERTEXT=...&SYMKEY=...&IV...
241440999:SUBMITRESPONSE?STATUS={FAIL:Sorry, but the ID you're using has been used by another user.}&OTP=..
241440999:SUBMITVOTE?ID=...&CIPHERTEXT=...&SYMKEY=...&IV...

Test Passed.

Voting Test #3
After the user has successfully voted, he attempts to change his vote by sending a CHANGEVOTE request.
Expected:
ID#:CHANGEREQ?ID=...&CIPHERTEXT=...&SYMKEY=...&IV=...
Actual:
241440999:CHANGEREQ?ID=...&CIPHERTEXT=...&SYMKEY=...&IV=...

Test Passed.

7.2.2 Vote Revealing Stage

During the reveal stage, the only actions that a user can take are revealing his vote and protesting the way his vote was counted. As such, we show two test cases, one for each of the previously mentioned items.

Vote Revealing Test#1
After clicking the "Reveal Vote" button, the user's client should generate a REVEALVOTE command, containing his ID number and the pair of his RSA decryption exponent and modulus, and send it.
Expected:
ID#:REVEALVOTE?ID=...&EXPONENT=...&MODULUS=...
Actual:
241440999:REVEALVOTE?ID=...&EXPONENT=...&MODULUS=...

Test Passed.

Vote Revealing Test#2
After clicking the "Protest Count" button, the user's client should generate a PROTEST command, containing his ID number and the pair of his RSA decryption exponent and modulus, RSA encryption parameters, ciphertext, and signed vote receipt, and send it to the CTF.
Expected:
ID#:PROTEST?ID=...&EXPONENT=...&MODULUS=...&CIPHERTEXT=...
&SYMKEY=...&IV=..&VOTERCPT=...
Actual:
61614213:PROTEST?ID=...&EXPONENT=...&MODULUS=...&CIPHERTEXT=...
&SYMKEY=...&IV=..&VOTERCPT=...

Test Passed.

7.3 CTF Testing

Testing the CTF is a little more involved than testing the client. Whereas the client is functionig properly if it correctly sends out various network requests, the CTF must verify users against its databases, who has connected before, et cetera. As such, our CTF testing is more lengthy.

7.3.1 Registration Stage

For the CTF, during the registration phase, the main things that are needed to be handled are accepting client registrations and making sure that they are valid registrations and that they have not previously registered.

Registration Test #1
Registration request for valid user "John Smith" and "123456789". The CTF should respond and acknowledge the registration.
Expected:
ID#:REGRESPONSE?STATUS={ACCEPT:Congratulations, you have been registered.}&RECEIPT=...
Actual:
5027100:REGRESPONSE?STATUS={ACCEPT:Congratulations, you have been registered.}&RECEIPT=...

Test Passed.

Registration Test #2
Registration request for "Bilbo Baggins" and "123123123". The CTF should respond, but not acknowledge the registration, since this is not a valid user.
Expected:
ID#:REGRESPONSE?STATUS={FAIL:status}&RECEIPT=0
Actual:
5027100:REGRESPONSE?STATUS={FAIL:Sorry, but you are not eligible to register.}&RECEIPT=0

Test Passed.

Registration Test #3
Register "John Smith" with SSN "123456789", which is a valid registration. The CTF should respond with a receipt and the client should automatically obtain an ID number by executing the IDENROLL command.
Expected:
ID#:REGRESPONSE?STATUS={ACCEPT:Congratulations, you have been registered.}&RECEIPT=...
ID#:IDRESPONSE?STATUS={ACCEPT:The ID is decrypted and attached}&ID=..
Actual:
5027100:REGRESPONSE?STATUS={ACCEPT:Congratulations, you have been registered.}&RECEIPT=...
5027100:IDRESPONSE?STATUS={ACCEPT:The ID is decrypted and attached}&ID=...

Test Passed.

Registration Test #4
Register "John Smith" with SSN "123456789", which is a valid registration, after he has already registered once. The CTF should respond that this is not a valid registration.
Expected:
ID#:REGRESPONSE?STATUS={FAIL:Sorry, but you are not eligible to register.}&RECEIPT=0
Actual:
651287367:REGRESPONSE?STATUS={FAIL:Sorry, but you are not eligible to register.}&RECEIPT=0

Test Passed.

7.3.2 Voting Stage

The main cases for the CTF voting stage that need to be tested are properly handling submitted votes when there is no ID number collision, handling votes when there is an ID number collision, and handling change vote requests. Since the CTF must access the database and validate ID numbers, these tests are more involved than the client tests.

Voting Test #1
Assuming the user is registered, submit a vote. It will send the vote encrypted using RSA hybrid encryption. The packet will contain the necessary values to load the RSA machine on the server side. The CTF should acknowledge the vote with a SUBMITRESPONSE message coupled with a signature of the vote.
Expected:
ID#:SUBMITVOTE?ID=...&CIPHERTEXT=-...&SYMKEY=...&IV=...
ID#:SUBMITRESPONSE?STATUS={ACCEPT:Your vote has been saved}&RECEIPT=...
Actual:
241440999:SUBMITVOTE?ID=...&CIPHERTEXT=-...&SYMKEY=...&IV=...
241440999:SUBMITRESPONSE?STATUS={ACCEPT:Your vote has been saved}&RECEIPT=...

Test Passed.

Voting Test #2
Assuming the user is registered and another user has voted with the same ID number, submit a vote. This should indicate a collision, which the client should process and reset his ID number.
Expected:
ID#:SUBMITVOTE?ID=...&CIPHERTEXT=...&SYMKEY=...&IV=...
ID#:SUBMITRESPONSE?STATUS={FAIL:Sorry, but the ID you're using has been used by another user.}&OTP=ID_{OTP}
ID#:SUBMITVOTE?ID=ID_{OTP}&CIPHERTEXT=...&SYMKEY=...&IV=...
Actual:
241440999:SUBMITVOTE?ID=...&CIPHERTEXT=...&SYMKEY=...&IV...
241440999:SUBMITRESPONSE?STATUS={FAIL:Sorry, but the ID you're using has been used by another user.}&OTP=ID_{OTP}
241440999:SUBMITVOTE?ID=ID_{OTP}&CIPHERTEXT=...&SYMKEY=...&IV...

Test Passed.

Voting Test #3
After the user has successfully voted, he attempts to change his vote by sending a CHANGEVOTE request. The CTF should respond with a CHANGERESPONSE command and a signed receipt of the new vote.
Expected:
ID#:CHANGEREQ?ID=...&CIPHERTEXT=...&SYMKEY=...&IV=...
ID#:CHANGERESPONSE?STATUS={ACCEPT:Your vote has been changed successfully}&RECEIPT= **Actual:**
241440999:CHANGEREQ?ID=...&CIPHERTEXT=...&SYMKEY=...&IV=...
241440999:CHANGERESPONSE?STATUS={ACCEPT:Your vote has been changed successfully}&RECEIPT=...

Test Passed.

7.3.3 Vote Revealing Stage

For the vote reveal stage, the CTF has a number of possible actions. First, it must accept decryption keys from different voters. It must also accept protests from users about how their votes were counted. Finally, it must be able to calculate and publish the results of the election. We show test cases for these three scenarios.

Vote Revealing Test#1
After a user has submitted a REVEALVOTE command, the server should generate a REVEALRESPONSE message to the client. Additionally, the server should update the database records with the user's decryption key and his plaintext vote. In this test, the client has voted for the candidate "Jackie Burkhart".
Expected:
ID#:REVEALVOTE?ID=...&EXPONENT=...&MODULUS=...
ID#:REVEALRESPONSE?STATUS=ACCEPT:Your decryption key has been saved Got vote: ID#:Jackie Burkhart **Actual:**
241440999:REVEALVOTE?ID=...&EXPONENT=...&MODULUS=...
241440999:REVEALRESPONSE?STATUS={ACCEPT:Your decryption key has been saved}
Got vote: ID#:Jackie Burkhart

Test Passed.

Vote Revealing Test#2
After the CTF has published the voting results, a user protests that his vote was not counted correctly. The CTF responds with a PROTESTRESPONSE command informing the user of his protest status. If his protest vote is not different from the stored vote, the signature is not valid, or there is some other problem, the protest is denied. If the signature is valid and the protest vote differs from what is stored, the protest is accepted. For this test, we connected a client and had him reveal a vote. We then modified the database while the client was connected and corrupted his stored vote. The client then protested the vote, which should thus be accepted. **Expected:**
ID#:PROTEST?ID=...&EXPONENT=...&MODULUS=...&CIPHERTEXT=....
&SYMKEY=...&IV=..&VOTERCPT=...
ID#:PROTESTRESPONSE?STATUS={ACCEPT:your vote has been updated successfully.}
Actual:
61614213:PROTEST?ID=...&EXPONENT=...&MODULUS=...&CIPHERTEXT=....
&SYMKEY=...&IV=..&VOTERCPT=...
61614213:PROTESTRESPONSE?STATUS={ACCEPT:your vote has been updated successfully.}

Test Passed.

Vote Revealing Test#3
Our final CTF test is that of counting and reporting the election results, which is arguably the most important function it does. For this test, we connected three client simultaneously. One client voted for 'John Smith', another voted for 'Gordon Freeman', and the final voted for 'Jackie Burkhart'. The client voting for 'Gordon Freeman' then changed his vote to 'John Smith'. All clients then revealed their votes and the results were taken. This test is good since it tests simultaneous connections, vote changing, and vote tallying. It is somewhat difficult to describe exactly here, but it is presented in the demonstration.
Expected:
John Smith: 2 votes
Gordon Freeman: 0 votes
Jackie Burkhart: 1 votes
Michael Bluth: 0 votes
Angela Martin: 0 votes
Actual:
John Smith: 2 votes
Gordon Freeman: 0 votes
Jackie Burkhart: 1 votes
Michael Bluth: 0 votes
Angela Martin: 0 votes

Test Passed.

8 Limitations

While we put our best work into this project, the project still has some limitations that need to be mentioned.

Firstly, our project only works over a secure channel, such as SSL. If an eavesdropper is able to intercept the messages flowing between the client and CTF, it will be very easy to execute various attacks, since messages are being sent in the clear.

Secondly, due to the nature of how mental poker is used over a large list of ID numbers, the polling place requires a large amount of storage for the ID list. This prevents users from voting at home, since it will take a large amount of space and bandwidth to store and transfer this list of ID numbers.

9 Future Work

In the future, we would like to expand our project in several ways.

One of the most useful extensions of the work would be to enable the system to work over an insecure channel. This would make it cheaper for a polling place to be set up, since dedicated secure lines would not be necessary. This would most likely not be difficult, since we would encrypt all traffic using the hybrid RSA scheme, but we did not have enough time to implement this.

Additionally, we would like to be able to extend our protocol so that voters could vote at home, rather than at a polling place. The problem with this is how to distribute the ID numbers evenly over the group. If you send a group of 50 users a small set of ID numbers, say 500 out of 200 million, you won't be able to learn anything about an individual voter, but you might be able to learn something about that group of 50 voters. We have some ideas on how to address this that would make for interesting future work.

It would also be interesting to provide a web interface to our project, so that voters could vote using a browser instead of downloading our custom Java client.

One problem we had was deciding how many ID numbers to generate for a given number of users. Due to the birthday paradox, a large number of ID numbers are needed, but is unrealistic to store enough ID number for 6 billion, or even 250 million people. With a 4096 bit mental poker key modulus, 250 million $\approx 2^{2}8$ people would require 4096 bits * $2^{5}6 = 3.7$ * 10^{7} TB of data, which is clearly not realistic. As such, we leave this problem as future work.

Finally, we would like to add features that allows the system to handle bugs when communicating over the network such as dropped packets or the like. One of our assumptions was that our network communication was robust and would not do this, so we could add this in the future.

10 Conclusion

We have presented a system for electronic voting. Our system envisions a central tabulating facility (CTF) to count votes and has voters go to their designated polling places to make votes. Voters are given a hardware voting token to store their personal digitial information. In contrast to a normal voting system, our system allows users to change their vote, even several days after voting, as well as dispute how their votes were counted.

Our system has been tested and shown to work in multiple different test cases. We have also leveraged existing tools and libraries, such as MySQL and BouncyCastle, as these have been verified and known to function properly.

Our system does have some limitations, such as that it requires a secure link between CTF and client and the networks must be robust. For the future, we would like to improve on these limitiations, relax some of the assumptions that we made, and add some additional features to make the system even more compelling.

This project was a wonderful learning experience. We were able to see how cryptography can be applied to solve a real problem; electronic voting. In addition to cryptography, we were exposed to networking, databases, algorithm, and protocol design. Working in a team also helped us learn many lessons, such as how to work together, but also how to use more practical techniques, such as version control.

We looked for ways to improve the existing protocol specification[2][1] and found that we could prevent a malicious CTF from miscounting votes. We added the step of having a CTF issue a receipt upon a user registration and voting, which allows users to prove their actions later. The original protocols lacked this and we feel this is one of our more important contributions to the problem.

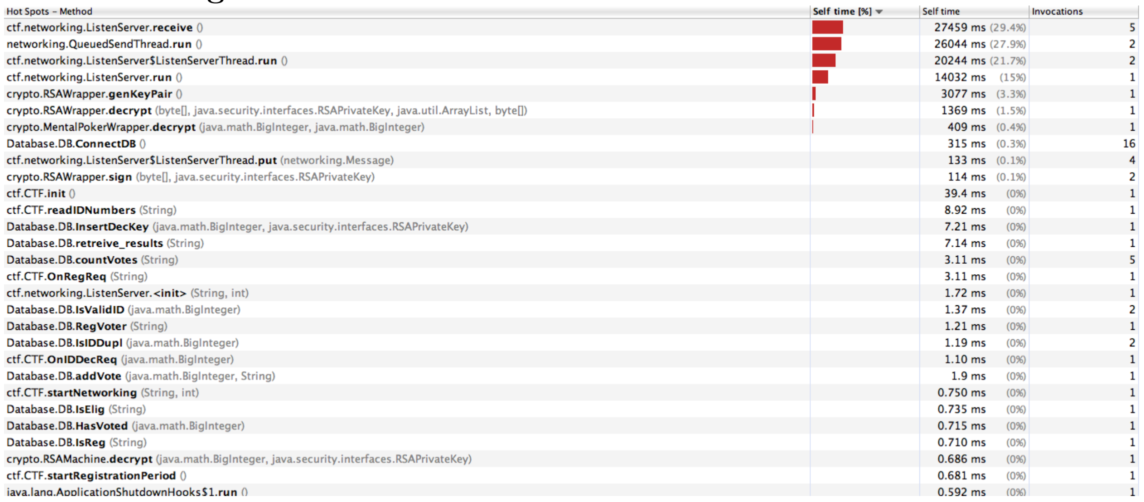
With the increasingly digital world, voting on paper seems to be a thing of the past. Electronic voting will become more important in the future and this project helped us to learn more about some of the difficulties with this problem.

11 Appendices

Any Code or explanation of some methods. e.g. the one in Bouncy Castle.

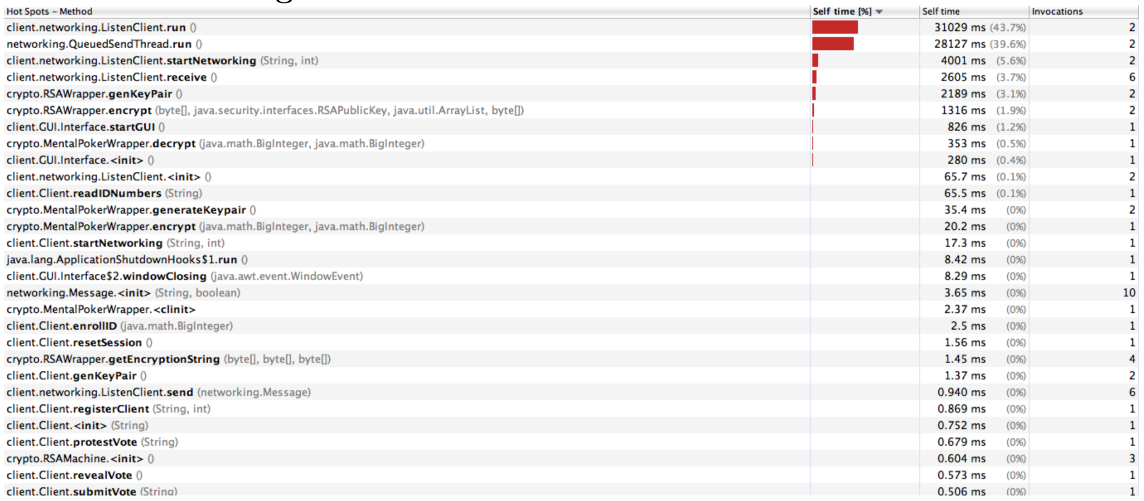
11.1 Profiling Images

CTF Profiling



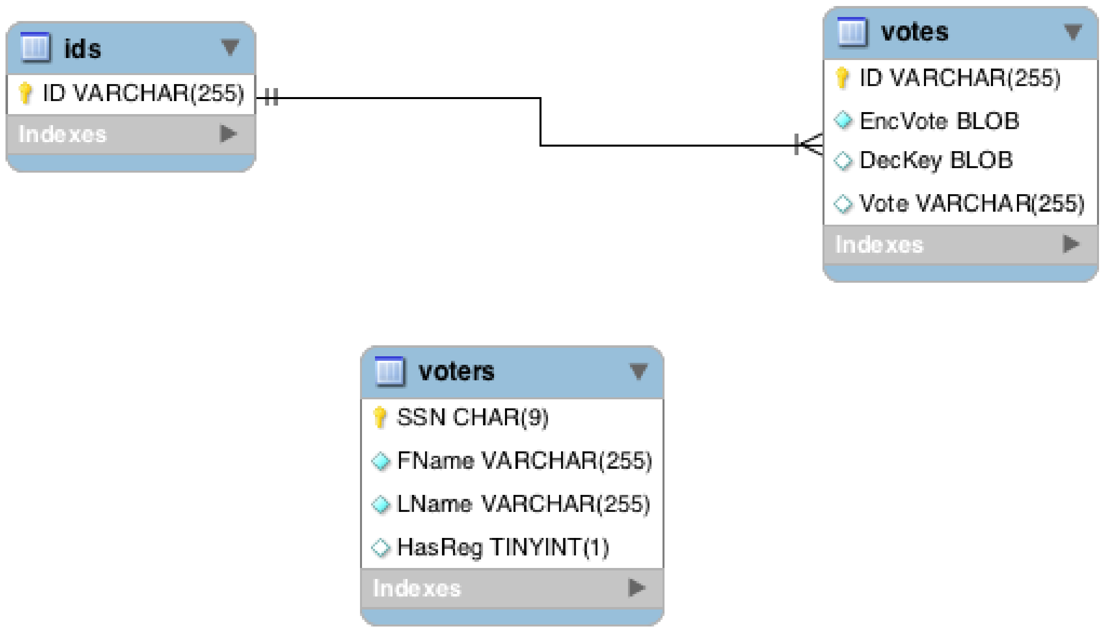
Hot Spots – Method	Self time [%]	Self time	Invocations
ctf.networking.ListenServer.**receive** ()		27459 ms (29.4%)	5
networking.QueuedSendThread.**run** ()		26044 ms (27.9%)	2
ctf.networking.ListenServer$ListenServerThread.**run** ()		20244 ms (21.7%)	2
ctf.networking.ListenServer.**run** ()		14032 ms (15%)	1
crypto.RSAWrapper.**genKeyPair** ()		3077 ms (3.3%)	1
crypto.RSAWrapper.**decrypt** (byte[], java.security.interfaces.RSAPrivateKey, java.util.ArrayList, byte[])		1369 ms (1.5%)	1
crypto.MentalPokerWrapper.**decrypt** (java.math.BigInteger, java.math.BigInteger)		409 ms (0.4%)	1
Database.DB.**ConnectDB** ()		315 ms (0.3%)	16
ctf.networking.ListenServer$ListenServerThread.**put** (networking.Message)		133 ms (0.1%)	4
crypto.RSAWrapper.**sign** (byte[], java.security.interfaces.RSAPrivateKey)		114 ms (0.1%)	2
ctf.CTF.**init** ()		39.4 ms (0%)	1
ctf.CTF.**readIDNumbers** (String)		8.92 ms (0%)	1
Database.DB.**InsertDecKey** (java.math.BigInteger, java.security.interfaces.RSAPrivateKey)		7.21 ms (0%)	1
Database.DB.**retreive_results** (String)		7.14 ms (0%)	1
Database.DB.**countVotes** (String)		3.11 ms (0%)	5
ctf.CTF.**OnRegReq** (String)		3.11 ms (0%)	1
ctf.networking.ListenServer.**<init>** (String, int)		1.72 ms (0%)	1
Database.DB.**IsValidID** (java.math.BigInteger)		1.37 ms (0%)	2
Database.DB.**RegVoter** (String)		1.21 ms (0%)	1
Database.DB.**IsIDDupl** (java.math.BigInteger)		1.19 ms (0%)	2
ctf.CTF.**OnIDDecReq** (java.math.BigInteger)		1.10 ms (0%)	1
Database.DB.**addVote** (java.math.BigInteger, String)		1.9 ms (0%)	1
ctf.CTF.**startNetworking** (String, int)		0.750 ms (0%)	1
Database.DB.**IsElig** (String)		0.735 ms (0%)	1
Database.DB.**HasVoted** (java.math.BigInteger)		0.715 ms (0%)	1
Database.DB.**IsReg** (String)		0.710 ms (0%)	1
crypto.RSAMachine.**decrypt** (java.math.BigInteger, java.security.interfaces.RSAPrivateKey)		0.686 ms (0%)	1
ctf.CTF.**startRegistrationPeriod** ()		0.681 ms (0%)	1
java.lang.ApplicationShutdownHooks$1.**run** ()		0.592 ms (0%)	1

Client Profiling



Hot Spots – Method	Self time [%]	Self time	Invocations
client.networking.ListenClient.**run** ()		31029 ms (43.7%)	2
networking.QueuedSendThread.**run** ()		28127 ms (39.6%)	2
client.networking.ListenClient.**startNetworking** (String, int)		4001 ms (5.6%)	2
client.networking.ListenClient.**receive** ()		2605 ms (3.7%)	6
crypto.RSAWrapper.**genKeyPair** ()		2189 ms (3.1%)	2
crypto.RSAWrapper.**encrypt** (byte[], java.security.interfaces.RSAPublicKey, java.util.ArrayList, byte[])		1316 ms (1.9%)	2
client.GUI.Interface.**startGUI** ()		826 ms (1.2%)	1
crypto.MentalPokerWrapper.**decrypt** (java.math.BigInteger, java.math.BigInteger)		353 ms (0.5%)	1
client.GUI.Interface.**<init>** ()		280 ms (0.4%)	1
client.networking.ListenClient.**<init>** ()		65.7 ms (0.1%)	2
client.Client.**readIDNumbers** (String)		65.5 ms (0.1%)	1
crypto.MentalPokerWrapper.**generateKeypair** ()		35.4 ms (0%)	2
crypto.MentalPokerWrapper.**encrypt** (java.math.BigInteger, java.math.BigInteger)		20.2 ms (0%)	1
client.Client.**startNetworking** (String, int)		17.3 ms (0%)	1
java.lang.ApplicationShutdownHooks$1.**run** ()		8.42 ms (0%)	1
client.GUI.Interface$2.**windowClosing** (java.awt.event.WindowEvent)		8.29 ms (0%)	1
networking.Message.**<init>** (String, boolean)		3.65 ms (0%)	10
crypto.MentalPokerWrapper.**<clinit>**		2.37 ms (0%)	1
client.Client.**enrollID** (java.math.BigInteger)		2.5 ms (0%)	1
client.Client.**resetSession** ()		1.56 ms (0%)	1
crypto.RSAWrapper.**getEncryptionString** (byte[], byte[], byte[])		1.45 ms (0%)	4
client.Client.**genKeyPair** ()		1.37 ms (0%)	2
client.networking.ListenClient.**send** (networking.Message)		0.940 ms (0%)	6
client.Client.**registerClient** (String, int)		0.869 ms (0%)	1
client.Client.**<init>** (String)		0.752 ms (0%)	1
client.Client.**protestVote** (String)		0.679 ms (0%)	1
crypto.RSAMachine.**<init>** ()		0.604 ms (0%)	3
client.Client.**revealVote** ()		0.573 ms (0%)	1
client.Client.**submitVote** (String)		0.506 ms (0%)	1

11.2 SQL Schema



References

[1] A. S. Hannu Nurmi and L. Santean, “Secret ballot elections in computer networks,” *Computers & Security*, vol. 10, no. 6, pp. 553 – 560, 1991.

[2] B. Schneier, *Applied Cryptography.* John Wiley & Sons, second ed., 1996.