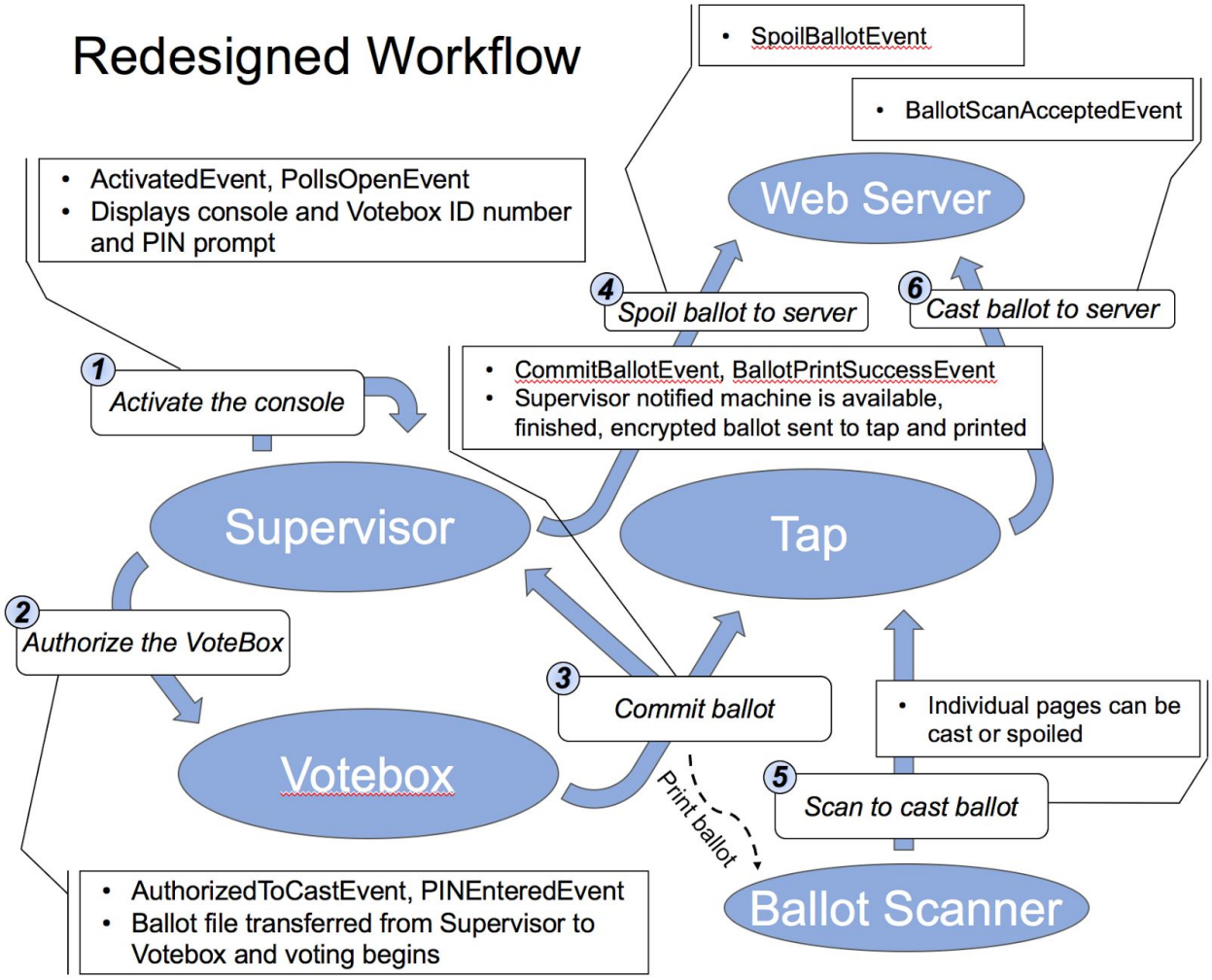
Resurrecting STAR-vote Recap and Further Instructions

Alex Addy
Zach Bodek
Susan Wen
Alex Shultz

STAR-Vote Code Structure



The STARVote project is an amalgam of different projects, student developers, and milestones, some of which contradict each other, so its workflow and structure are, in places, convoluted. This diagram is our attempt to abstract the project's mechanics. Each of STARVote's modules are organized around the Auditorium, which serves as the communication link between the computers running on the network. The modules that use Auditorium include Supervisor, Votebox, Tap, and BallotScanner. The Supervisor starts by "activating the console", which

initializes a UI to control the Votebox instances, start a voting session/select a ballot, generate PINs for voters, and stop a voting session. The Supervisor is to be used by the election staff. The Votebox is the interface through which the voter makes their selections and cast their ballots. The voter enters their PIN, the ballot ZIP file is transferred from the Supervisor, and the voter UI is initialized. The voter finishes the session, and then chooses to print the ballot. The ballot is rendered to a PDF file, and the Votebox prints the ballot to the VVPAT printer. Each ballot page has an individual ID so that pages can be cast and spoiled individually. The voter then chooses whether or not to spoil each page. To spoil a ballot, the voter takes the page to the Supervisor and is scanned. The completed ballot is sent to the Tap, which forwards it to the web server. At this point, the voter can use the instructions on the ballot page to check the validity of their ballot. If the voter chooses to cast their ballot, they take the ballot to the BallotScanner, which scans the ballot and casts the ID. When the ID is cast, the Tap uploads the encrypted

Communications among these “actors” are through messages. The machines send messages to auditorium to report any events, and auditorium broadcasts messages to all other machines. Messages extend ABallotEvent. Because there are many different types of the messages, and not all of them are relevant to the specific role of the machines, each “actor” installs the matcher for a specific event while starting up so that it reacts to these events.

Instructions on setting up simulations

- Get your own router (can’t use Rice Owls or Rice Visitor)
    - Rice Owls, and Rice Visitors have internal load balancing in their routers, so other machines might not be visible to you even when you are on the same network. Having your own router will guarantee that you discover each other during the auditorium broadcasting events. If you have a dysfunctional router, you might see ConnectionTimeOut exception during the discovery period of the program. This can often be solved by connecting to the router with ethernet cables.
- Each computer will play a different role in the system (Supervisor, Votebox, Tap, and BallotScanner)
- You will need a minimum of two machines to do anything (Supervisor and Votebox)
    - Need the following in run configurations:
    - VM argument: “-Djava.net.preferIPv4Stack=true”
    - Need a unique number (0, 1, 2, etc) under Program Arguments.
        - Each machine needs a unique number
        - For running the Tap three arguments in your runtime configuration. A serial number for your machine, a report address (not actually used), and a port for connection. When we are testing on localhost, we are actually using the hardcoded “localhost:9000” rather than the address for checkmyvote.com, the report address and port are not used. However, for the main function to run successfully, you need to have placeholder

arguments, (so just do a serial number, another number, then a 4-digit port number) ex: (4 12 4000)

- Run Supervisor and Votebox on two different machines. Console should indicated that they “Connected” and then “Joined!”
- Once connected, the Supervisor GUI will let you “Activate this Console”
- Once activated, Supervisor must load the test ballot
    - Located here:
    - src/main/java/test-ballots/testballot.zip
- Supervisor, then click “Open Polls Now”. Any connected Voteboxes will then have a field for a 5 digit PIN
- Supervisor, click generate PIN (from precinct “lot”).
- Once this PIN is entered into a Votebox, the Votebox GUI will take over.
- Vote and click print ballot (as of now, this writes a PDF to disk). Your main desktop should return after “printing.”
- The Ballot ID will be displayed somewhere in the Votebox console. It is a 15 character string with dashes every three characters. (ex: **HV1-235-Z75-68R-K84**)
- To spoil the ballot, Supervisor should click “Spoil Ballot” and then enter the ballot id exactly as it appears (notice the difference between O and 0).
    - Remember that capitalization and dashes matter
    - If the Tap is running, it will upload the spoiled ballot’s ID to a localhost
    - If the webserver is running on localhost, it will register the ballot’s ID as spoiled
- To cast the ballot, BallotScanner must also be running. Enter the ballot ID exactly as it appears into the BallotScanner console to cast the ballot
    - Remember that capitalization and dashes matter
    - If the Tap is running, it will upload the cast ballot’s ID to a localhost
    - If the webserver is running on localhost, it will register the ballot’s ID as cast.
- To stop, Supervisor should click “Close Polls Now.” This will prompt a batch upload of all the cast and spoiled ballot IDs to the locally run webserver, in case the real time uploads were disabled.
- We hope your simulations go smoothly! Good luck!

Webserver Setup

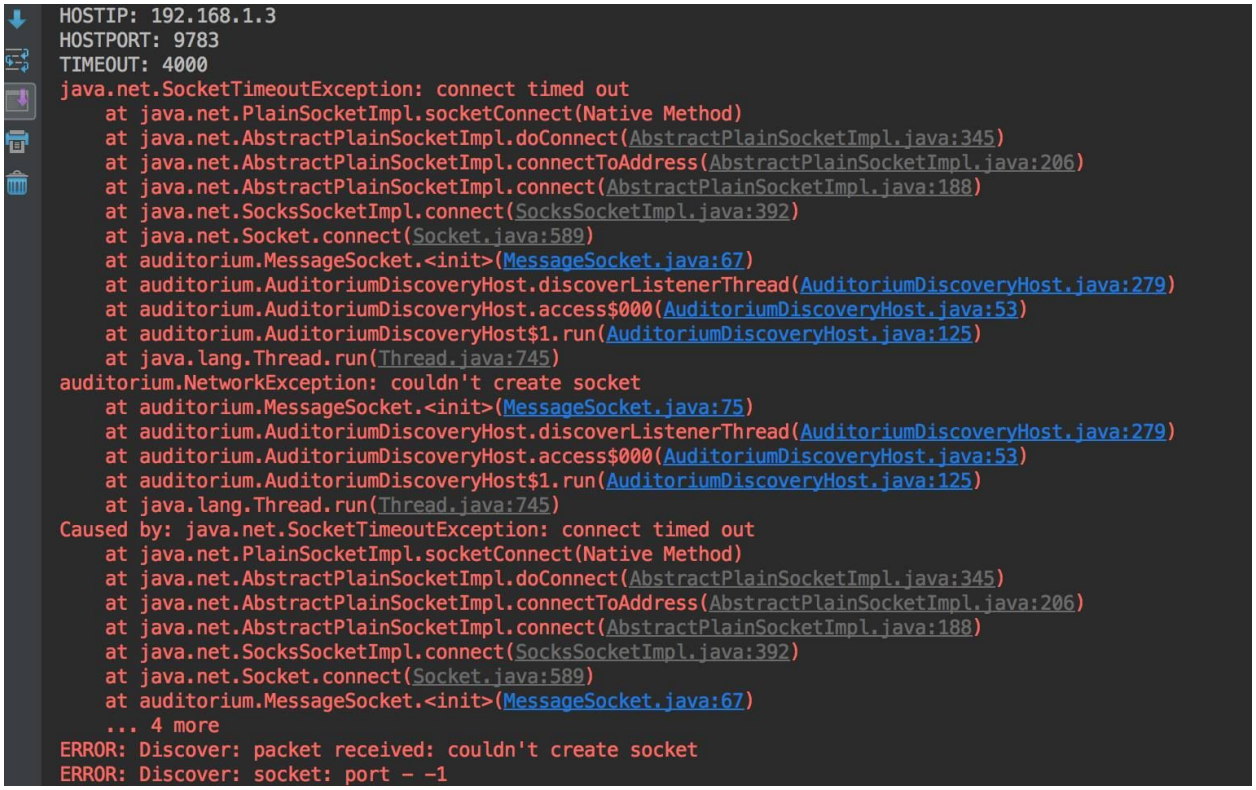
- How to setup activator on your local host
- (For the record, we aren’t sure what the relationship between the Play library, the Play application, and the Activator application is, and there isn’t much documentation left behind from whoever wrote the webserver to give us an idea of how to run it. This is the product of our trial-and-error.)
    - Download the Activator binary from https://www.lightbend.com/activator/download.
    - We found the webserver to behave better (compile at all) when run from the command line version of Activator, rather than the web UI version.
    - In the root of the starvote-webserver repository, run the activator binary.

    - Activator will start up. If it gives you an error, you're probably missing a library that it wants. Install it and then restart Activator.
    - Activator will present you with another command line prompt. Type "run" and then hit enter. After it finishes compiling, you can access the web server at localhost:9000.
    - Everytime you refresh the page, the activator will try to rerun the code. Since IntelliJ auto saves your changes, you can test compilation errors and runtime errors on the fly.
    - However, there are cases where you cannot see your immediate changes, it is most likely that your browser cached previous changes, so you clear your cache or open the website on Incognito version.

Troubleshooting

- NullPointerException in xml ballot schema on Votebox
    - In Gradle, the *getResource()* method is determined by the IntelliJ build environment. If you're resource path is incorrect, you'll get this exception because it cannot load the static resource. Set your resource path correctly, and this should go away. Also be wary of checking out changes from Github relating to IDEA files. If another member of the group commits an incorrect file to the repository, you'll continue to have the error because your fix was overwritten.
- PDFBox not being recognized as a valid library
    - If IntelliJ does not recognize PDFBox as a library, and you get a compile time error, try updating Gradle and then rebuilding the project
- Exception trying to establish a socket (AddressInUseException)
    - Unable to connect to Auditorium: Could not bind the discovery socket: Address already in use
    - Only one instance of an Auditorium-using program (Votebox, Supervisor, Tap, BallotScanner) can be run on any machine at one time. Run the program on another machine connected to the same network to solve this error. Also note that the webserver does not use Auditorium, and can be run on the same machine as Tap for simplicity.
- Invalid hostname exception
    - This happens when Java tries to connect using IPv6. Make sure to set your VM arguments correctly. Note that IntelliJ has different VM arguments for different main methods, so if you switch to a new role, you'll have to add in the VM arguments again.
- ConnectionTimeOutException when connecting with other machines
    - We were unable to determine a definitive source for timeout exceptions that we encountered. The most robust way we avoided these issues was to connect all of the computers via ethernet rather than WiFi. Sometimes the timeouts were flukes and restarting everyone fixed the issue. There is also a timeout setting in the

Supervisor configuration file which can be configured. We have this set to a large value, but you could increase it if it fixes the timeout exceptions.



What we achieved this semester:

Dynamically generated ballot IDs

When we inherited STAR-Vote, ballot IDs were simply random numbers, typically 9 or 10 digits. The human factors testing around the website found success in a 15 character string mix between numbers and characters, with 5 three character sections, separated by a dash. We wrote a method to generate this type of ballot ID to accommodate the human factors testing.

Timed spoiling of ballots

At Claudia's request, we also implemented a feature that spoils ballots based on a variable timer. After a ballot is printed, the Supervisor holds onto that ballot id. If that ballot is cast and put into the ballot box, Supervisor forgets it. If 5 minutes pass and that ballot has yet to be cast or spoiled, then Supervisor spoils it.

Redesign of the ballot render class to use PDFBox

The initial ballot renderer included in STARVote constructed the ballot as an HTML file and then ran a PDF application, separate from STARVote, that “prints” the HTML file to a PDF. The process struck us as convoluted, and we were unable to ever get the process to work, probably because we were running on MacOS and the last round of developers were running on Windows.

We decided to rewrite the ballot renderer, the result of which is in the BoxPrinter class. Rather than have the intermediate HTML step, we use the Apache PDFBox library to render the ballot directly to a PDF file. When Votebox calls the printCommittedBallot method in the class, it begins by creating a PDDocument object. We render the ballot ID barcode using the PrintImageUtils class. We then loop through all of the races provided to the printing method and render them on the page. Because we don’t have access to the plaintext of the ballots, we have to render the PNG files included with the ballot. This presents more problems because the images are too small to be legible on the letter-sized paper. To remedy this, we crop the whitespace out of the image. We then add all of the elements to the page, column wise, and return the PDDocument object to the Votebox instance. We chose to separate the saving of the PDDocument from the rendering to allow Tap to send the ballot document to the webserver to be rendered when the voter checks their spoiled ballot. Once the ballot is rendered and returned, Votebox saves the ballot to file.

Reversal of the ballot ID generation workflow to occur after ballots are cast

Previously, ballot IDs were randomly generated at the beginning of each Votebox voting session. When the user finished voting and committed their ballot, the ID was used in Auditorium events. In order to accommodate the casting of ballots by page, we had to restructure the way that STARVote generates the IDs. Now, when the user decides that they want to commit their ballot, Votebox determines how many physical pages the ballot will fit on to, and generates ballot IDs for each of the pages. Then, the CommitBallotEvent is done with the new generated ballot IDs, and the workflow returns to normal.

Each of the individual ballot pages has its own ID number, each of which can be committed or spoiled independently of the other pages.

Migration of STARVote main project to Gradle and STARVote-Webserver to an SBT repository

When we inherited it, the STARVote repository did not have a build system, and the webserver source code was not separated. We decided that, in order to facilitate development in the future, we would migrate the main STARVote repository to the Gradle build system and would migrate the webserver to its own repository, in order to better make use of its existing SBT build system. We accomplished the latter this by moving the webserver to a new Github repository (starvote-webserver). For the main

repository, Brian and Jerry superficially implemented the Gradle build structure, but we were tasked with fixing the build process, which was broken by the migration. Gradle changed the way that static resources were loaded by Java, so we had to troubleshoot each instance of static resource loading.

Uploading balloting casting/ committing/ spoiling to web server in real time

For the human factor testing, we don't want the users to wait for the poll to be closed to check their vote, so we decided to transmit the ballot information in real time to the web server. To make it happen, we add Matchers for SpoilBallotEvent and CastCommitedBallotEvent in Tap so that Tap responds to these Events while listening to all the messages.

When SpoilBallotEvent or CastCommitedBallotEvent is broadcast in the auditorium, Tap finds relevant information for each event, ballot ID for CastCommittedBallotEvent, ballot ID, precinct ID, and (supposedly) decrypted ballot selection for SpoilBallotEvent. It then serializes the information into fields of BasicNameValuePairs for the HTTP POST request that goes over to the webserver. For testing purpose, Tap sends the information to localhost:9000. Next step will be enabling the Tap to send to actual CheckMyVote website. Because we are sending messages over the internet, we need to serialize all the information to Sexpressions. Details on how to use the sexpression packages is here.

Necessity for decrypting ballots

As per Claudia's request, we were attempting to display a PDF of a spoiled ballot on the website. By showing a voter the exact choices on their spoiled ballot, we can instill more confidence in the voter that their incorrect choices were indeed not counted. To do this, we would need to decrypt the ballot upon being spoiled, and send the decrypted information to the web server, so it could reconstruct the PDF of the ballot. We unfortunately made the assumption that system decryption had been implemented already. However, decryption was only implemented such that a single Votebox could decrypt the ballots that it had originally encrypted, whereas StarVote requires the Supervisor to decrypt a ballot after it has been spoiled. This discovery came late in the semester, and set back some of our planned achievements. It will make a good project for a future semester, and we have already laid out a gameplan for fixing the issue.

Storing relevant ballot information in web server for verification

The web server uses the spark play framework. Under the framework, the model implements a EBean database. Both CastedBallot and ChallengedBallot are instances of the database. The database can hold different fields of the ballots, including ballot ID, hashed ballot information, precinct information, and ballot race selections. These fields

are helpful in generating the PDFs of the spoiled ballots. But since we have not figured out how to decrypt the encrypted ballot on STAR-vote, these fields now only hold placeholder strings.

For human factors real time testing, whenever the web server receives an HTTP POST request on uploading the casted ballot or challenged ballot (spoiled ballot), the Audit Server will check if the ballot ID already exists in the EBean database of Casted or Challenged ballot using static method. If not, it will create an instance of the CastedBallot or ChallengedBallot and insert them into the Casted/ Challenged ballot database using static method (CastedBallot.create(new CastedBallot(“ballotID”, “BallotHash”)).
When the poll is closed, Tap will receive all the voting events as supervisor record. It will upload all the record to the web server. Audit server then extract the ballot information and create the ballot information accordingly and insert them into the database. If there are ballots with the same ballot ID, the Audit Server replaces the ballot information with the one in the last batch upload.

When any user look up their ballot on the verification website www.checkmyvote.com (right now it only works on local host), the audit server looks up in the EBean database for the ballot with certain ballot ID and return the information to the user.

Adding QR code and instruction page

As requested by Dr. Wallach, voters are now given a QR code that they can scan to visit checkyourvote.com and instantly see their ballot status, without having to type their ballot ID. Currently, this takes the user to a localhost:9000 address, as the website has not gone live yet. To change this after the website goes live, one line of code in the BoxPrinter class has to be uncommented, and one has to be commented out. The QR code generation relies on two libraries, ZXING and QRGen.

What’s left to do:

Some features could not be completed by our group within the allotted time of the semester. The following is a rough explanation of the what tasks remain and how to complete them.

Decrypt spoiled ballot

The STARVote system does not as yet decrypt ballots. There are two ways to decrypt the ballot, as shown in the slide below.

Elgamal decryption

***Two* ways to decrypt:**

$$E(g^a, r, M) = \langle g^r, (g^a)^r M \rangle$$

$$D(g^r, g^{ar} M, a) = \frac{g^{ar} M}{(g^r)^a}$$

$$D(g^r, g^{ar} M, r) = \frac{g^{ar} M}{(g^a)^r}$$

g	group generator
M	plaintext (message)
r	random (chosen at encryption time)
a	(private) decryption key
g^a	(public) encryption key

(Slide 44 of STAR-vote lecture by Dr. Dan Wallach)

All the encrypt and decrypt methods are implemented in DHExponentialElGamalCryptoType.

The current STARVote implementation has a decrypt method that decrypts the ballots using a list private keys. It is only called in the BallotCrypterTest file. However, all ballots share the private key in one voting machine, but they have different random numbers associated with each page of ballot. When you spoil individual ballot pages, it is best to publish its associated random number rather than the general private key. So you need to implement the second decryption method that uses the random number.

The DHExponentialElGamalCryptoType has an AdderPublicKey PEK, which it uses to encrypt ballots. In line 88 of the class AdderPublicKey in package crypto.adder, the random number is generated with the code: `AdderInteger r = AdderInteger.random(q);`. It is this random number that needs to held onto in the case that a ballot will be spoiled and then decrypted.

Our recommended game plan for implementing this other decryption type involves scaling from a test, to one machine system, to the StarVote network. First, you should add tests to the BallotCrypter test file, starting with a static number instead of a random one and decrypting individual bytes up to decrypting an entire byte array. Next, turn the static number into a randomly generated one that you can hold onto inside the test, and update your decryption code to handle it. Then, figure out a way to hold onto those random numbers outside of the test file,

but still accessible on that machine. Finally, ensure that the random numbers can be held onto on one machine but can be retrieved by another. Remember, Votebox does the encryption, but Supervisor does the decryption if that ballot is later spoiled. The last step should be ensuring that it can be achieved across two different machines. (this will probably involve writing a new event and event handler to be announced in the auditorium).

Generate PDF of spoiled ballot in web server

To render spoiled ballots for the user, we anticipate that future developers could proceed one of two ways, depending on whether or not the way ballots are created is changed, as discussed above.

If ballots continue to not contain the plaintext list of candidates, then the rendering can be done with the BoxPrinter class that is included with the STARVote library made available to the web server. By passing in the spoiled ballot data in the same way that VoteBox produces ballot printouts, the web server can produce the same ballot printer to the user in the form of a PDF embedded in the verification page.

If ballots are updated to include plaintext, the renderer should probably be rewritten to generate an HTML verification page. This would be faster, more lightweight, more user-friendly (probably, but we aren't human factors people), and more widely compatible.

In either case, the Tap will need to be updated to include ballot data with spoiled ballot events, which it currently does not. We attempted to do this using serialized s-expressions, but were not able to complete the task due to time constraints. We think that the vision behind the Tap was to use s-expressions to represent all of the ballot data, whether encrypted or not, but the implementation as it stands, prior to our arrival, is a mixture of s-expressions and Java class serialization. Dr. Wallach discussed during the semester that it would be good to rip out all of the communication logic and replace it with Protobufs, which would be a cool project to tackle.

Ballot tallying

The infrastructure for ballot tallying is already in place in the webserver. The system is able to generate trustee keys as defined by its configuration file and runs without error when polls are closed by the Supervisor targeted at it. There are two structural problems with STARVote that will need to be addressed before tallying works in the way that the specification intended and with any sort of user-friendliness. The first problem, described previously, is that decryption has not yet been implemented on the webserver, which, for obvious reasons, is required to tally the tally the encrypted ballots and decrypt the sum.

The second is that Preptool generates ballots that do not contain the plaintext of the candidates. Rather, the candidates' names are only contained within the image files that it generates. This was done to be a security feature, so that a third party could not increase the vote count of a

particular candidate (because there was no way to tell which candidate is who). Unfortunately, this prevents the tallier from being able to report a tally that maps names to votes. The ballot structure will have to be rewritten to include the plaintext names of the people on the ballot. We anticipate that this could be accomplished easily by including an additional file in the ballot archive, which could be read by the supervisor and sent to the webserver when the polls close. Eventually, it would be beneficial to rewrite the entire ballot structure to be based primarily on plaintext, rather than on images, but this would require a significant rewrite of Preptool and of Votebox.