Final Report: The implementation and extension of an anonymous, receipt-free, universally verifiable voting system

CPSC 410/411
Soham Sankaran
Sachith Gullapalli
Lincoln Swaine-Moore

1. Introduction

Elections are the most vital aspect of democracy. That citizens are able to cast votes freely and for the candidates and proposals of their choice is integral to the wellbeing of any democratic nation. Without them, or perhaps more to the point, without confidence in them, there is little to separate democracy from oligarchy. Unfortunately, elections, and voting in particular are a fraught enterprise, for a number of potential reasons. These include campaign finance, apathy among the electorate, and the political homogeneity of social circles.

In this project, we address a wholly separate concern that is orthogonal to the above, but equally if not more important: the collection and tallying of votes.

As it stands, it can be difficult for a skeptical voter to have any assurance that their vote has in fact been cast correctly. Walking out of their polling place, they can only know that they tapped a button on a touch screen or penned in a bubble on a scantron sheet, and the rest is faith. As election results come in, the narrowest results reported are usually at the precinct-level. [1] Were one's precinct totals completely contrary to expectations (based on demographic or polling data, for example), one might have cause for concern. But barring that sort of extreme circumstance, one can have little confidence that one's vote (and perhaps the votes of millions of others) has not been changed in a more sly and subtle way, or perhaps simply miscounted by accident.

There is no reason, however, to suppose that only a true skeptic need be concerned. Historical evidence suggests there may be real cause for worry, both of accidental miscounting of ballots and of intentional miscounting of ballots. For evidence of the former, we need only to look back 16 years to the U.S. Presidential Election of 2000, where the counting of votes--in this case, particularly in Florida--was contested. Due to some poorly constructed ballots, it was

[1] An example of this granularity available here http://storymaps.esri.com/stories/2012/precincts-2008/.

difficult to properly tally the vote, and thousands of people may have accidentally voted in ways they did not intend--perhaps enough to change the election outcome.[2] A faulty ballot is certainly to blame, but the wrongs that came out of the poor ballot design could have been righted had the system been designed so that voters could more easily check that their vote was cast properly. Technology advance have not been a panacea so far. More recently, security researchers have demonstrated that is fairly trivial to modify the DRE voting machines which "took off in 2002, in the wake of *Bush v. Gore*."[3]

And, crucially, it is now apparent that foreign powers have an active interest in American elections. Hacking of the Democratic National Convention raised alarms, and in October, the Department of Homeland Security and the Office of the Director of National Intelligence announced that,

> "The U.S. Intelligence Community (USIC) is confident that the Russian Government directed the recent compromises of e-mails from US persons and institutions, including from US political organizations. The recent disclosures of alleged hacked e-mails on sites like DCLeaks.com and WikiLeaks and by the Guccifer 2.0 online persona are consistent with the methods and motivations of Russian-directed efforts. These thefts and disclosures are intended to interfere with the US election process."[4]

What's more, the attacks now appear to have had a deliberate intention in mind. According to the Washington Post, "FBI Director James B. Comey and Director of National Intelligence James R. Clapper Jr. are in agreement with a CIA assessment that Russia intervened in the 2016 election in part to help Donald Trump win the White House."[5] Notably, there has not been

[2] http://www.cnn.com/2001/ALLPOLITICS/03/11/palmbeach.recount/.

[3] http://www.politico.com/magazine/story/2016/08/2016-elections-russia-hack-how-to-hack-an-election-in-seven-minutes-214144.

[4] https://www.dhs.gov/news/2016/10/07/joint-statement-department-homeland-security-and-office-director-national

[5] https://www.washingtonpost.com/politics/clinton-blames-putins-personal-grudge-against-her-for-election-interference/2016/12/16/12f36250-c3be-11e6-8422-eac61c0ef74d_story.html?utm_term=.05d617c29bc7

any substantiated claim that Russia's "interefere[nce]" touched the voting process itself, or involved any hacking of voting machines or processes. But the point remains: at least one foreign power is interested in steering the results of a U.S. election.

There are several potential reasons this known motivation for foreign cyber attack may be concerning in the context of voting. First, a foreign power may have literally altered votes or vote totals, which would be a distortion of democracy. In the instance of the 2016 U.S. Presidential Election, it seems likely that this did not occur.[6] However, it is not impossible that something untoward occurred. Second, a foreign power might alter votes or vote totals in a future election. In Ukraine, "Russian hacktivists" were responsible for executing a series of attacks that "rendering the vote-tallying system inoperable," creating "fake computer vote totals," and "delaying the finally [sic] tally until the early morning," though involvement of the Russian government is in question.[7] It is not out of the question that a foreign government, or perhaps other "hacktivists" would attempt something similar for a U.S. election. Third, it raises concerns among the public that one of the first two things has happened or will happen, and potentially undermines trust in elections. This is an issue because it could disincentivize voting in a country with already record turnout rates.[8] It could also put power in the hands of a politician willing to create panic through fearmongering in the wake of an unfavorable election result. Notably, a foreign government might be interested in accomplishing any of these things, and a cyber attack could do so.

The sorts of actions that would raise these kinds of concerns likely violate existing norms of nonintervention in the context of current international law. Indeed, they might even be

---

[6] http://www.nytimes.com/2016/11/25/us/politics/hacking-russia-election-fears-barack-obama-donald-trump.html?_r=0

[7] http://www.csmonitor.com/World/Passcode/2014/0617/Ukraine-election-narrowly-avoided-wanton-destruction-from-hackers-video

[8] http://www.cnn.com/2016/11/11/politics/popular-vote-turnout-2016/

sufficient to justify countermeasures on the the part of the country whose sovereignty has been infringed, though it would probably depend whether the attacks constituted a “use of force” under Article 2(4) of the United Nations Charter.[9] But escalation of this manner may not be the best solution, or at least certainly not the only solution. Instead purely domestic measure can be implemented to reduce concerns. Indeed, to combat these issues, it is desirable to implement a voting system with the following properties (among others):

1. Votes are verifiable. After casting their vote, voters should be able to verify at a later date that it is recorded properly, and that the tally has been done correctly.
2. Privacy. No individual should be able to identify another individual’s vote.
    a. Receipt-free. Verifying one’s vote should not involve evidence that enables any individual other than oneself to identify one’s vote.
3. Efficiency. The system should be practical, and not involve time complexities that could not be reasonably used in an election context.

Such a system would enable voters to verify that their votes were registered as they had cast them without introducing other harms into the democratic process (such as vote buying). This in turn would both deter direct foreign involvement in the voting process, and allow anyone concerned about their vote being intentionally or mistakenly miscounted to assuage those concerns. Under this system, were a country to launch a cyber attack, it would be possible to detect its occurrence. As such, this voting system would augment national cyber security, and help combat the potential for cyber warfare between two countries, such as the United States and Russia.

[9] http://www.un.org/en/sections/un-charter/chapter-i/

Moran and Naor have proposed systems with these properties in their papers "Receipt-Free Universally-Verifiable Voting with Everlasting Privacy"[10] and "Split-Ballot Voting: Everlasting Privacy With Distributed Trust."[11] We have spent the past semester implementing the cryptography and interface (including printing of receipts) that constitutes their system of vote casting, tallying, and verification from the former paper. In addition, we have introduced several new components in order to address potential attack vectors that remain unresolved in the paper, including signatures for receipts, and a receipt-verification web application. Section 2 describes the system from the paper. Section 3 discusses one major addition we made to the protocol from the paper. Section 4 discusses implementation details of our system, and other on the work of the paper. Section 5 offers some potential future directions for our work.

2. Description of System

Moran and Naor's 2006 paper (referred to henceforth as MN06) presents the first ever receipt-free verifiable voting system with information-theoretic privacy. Their protocol retains integrity (meaning that the final tally is verifiable) even in the event that all voting machines and talliers are compromised, although for obvious reasons the receipt-free property is lost here (if a voter is entering a vote in plaintext, a compromised machine can just illegally ascribe that vote to the voter). Importantly, while previous receipt-free verifiable voting systems required voters to perform mathematically intensive operations within the voting booth or rely on the integrity of a secure computing device, MN06 only requires the voter to confirm that the random challenges they selected are the same as on their receipt, and that the voting machine's commitment to the dummy challenges is printed on the receipt before the voter has entered the real challenge --

[10] Moran, T., & Naor, M. (2006, August). Receipt-free universally-verifiable voting with everlasting privacy. In *Annual International Cryptology Conference* (pp. 373-392). Springer Berlin Heidelberg.
[11] Moran, T., & Naor, M. (2010, October). Split-Ballot Voting: Everlasting Privacy With Distributed Trust. In *ACM Transactions on Information and System Security* (*13:16:1–16:43*).

given these invariants, the rest of the verification process can be performed externally and by third-party verifiers. This is done as a zero-knowledge proof that does not leak information about the vote. These verifiers simply confirm that the commitments printed on the receipt are valid, with probability ½ of detecting any individual fraudulent commitment -- if the voter-generated challenges have sufficiently high entropy, this provides high confidence that the receipt is a valid commitment to the intended vote. In the final tally phase, the voting authority generates a permutation of masked vote commitments, which is released publicly. Depending on the value of a random challenge bit, the voting authority then either releases the generating permutation and masks (demonstrating that it had "nothing up its sleeve" and that the masked vote permutations were generated legitimately from the public vote commitments) or opens the commitments and releases a vote tally that corresponds to the previously released permutation of masked commitments (and which is verifiable as an opening of these masked commitments via a special property of Pedersen commitment). Assuming that the voting authority cannot predict the value of these challenge bits (a verifiably random source is crucial here), it has again only a ½ chance of getting away with any individual fraudulent tally. If enough tallies are conducted, the legitimacy of the final result is then virtually assured.

We initially thought that dependence of MN06 on an incorruptible central voting authority to maintain voter privacy is a major drawback -- after all, the vulnerability of such authorities and systems is a large part of the motivation behind verified voting systems. In addition, the secret ballot is a hallmark of democracy for a reason: it is not hard to imagine the ends to which an authoritarian state might turn such a centralized store of voters and their votes. As it turns out, however, given that our eventual deployment model involves using a separate DRE for each voting machine, there is no overarching 'central authority' -- compromising one DRE would only deanonymize a trivial fraction of the votes in a given election. As a result, we now believe that the MN06 single-DRE model is reasonable in practice, and that split DRE systems such as the one described in Moran and Naor's 2010 paper referenced earlier are likely not worth using, on balance, due to massively increased protocol complexity.

3. Our Additions to the Protocol

As written, the protocol is not entirely bulletproof. Moran and Naor note in MN06 that, "...our model does not prevent false accusations against the DRE. For example, a corrupt voter that is able to fake a receipt can argue that the DRE failed to publish it on the bulletin board. Using special paper for the receipts may help, but preventing this and similar attacks remains an open problem." We have dedicated much thought to addressing this concern of voters crying wolf, so to speak, and here present our solution.

The problem of fake receipts is, at least superficially, easily remedied. The voting machine can make use of public-key cryptography to sign the receipts it prints for the voters, using a private key which is known only to the voting machines. Any "corrupt voter" (hereafter, CV) then, who wishes to falsify a receipt will be stymied by an inability to properly sign the receipt, and the receipt can be falsified by anyone who wishes to verify the authenticity of the CV's produced receipt.

Unfortunately, though, this fix introduces an interesting complication. Because the voting machine is responsible for signing the receipts, and because cryptographic signatures are not easily verifiable by hand or upon inspection, a corrupt DRE could print receipts that are intentionally signed incorrectly, and delete the votes associated with them. Presumably, an intelligently corrupt DRE would not do so consistently (it would obviously be suspicious if every single voter received a fake receipt), but could do so infrequently, perhaps even targeting voters who are unlikely to verify the validity of their votes. Thus, this initial “fix” allows for systematic disenfranchisement of voters.

Luckily, it is possible to mitigate this issue by providing a means for voters to verify the authenticity of their receipt at the polling station. This can be accomplished by making the signature (and the contents of the receipt) QR code-scannable. Then, after casting their vote, a voter may take their receipt to a designated receipt checking station, and use a receipt-checking app to scan the contents of the receipt and verify that the signature is correct. Voters could use apps developed by any third party they wish (for instance, the major political parties could choose to develop their own receipt verification applications), and in the event that a receipt does not verify as being signed by the voting authority, the voter can vote again and can submit a formal challenge at the polling station.

One more challenge arises in the wake of this solution. A CV could theoretically print a fake receipt (which would not be certifiable because it could not have a correct signature) ahead of time, and bring this to the polling station. Then at the receipt verification stage, this receipt would not verify, and the CV would potentially be able to delegitimize a fully functional voting system, again by crying wolf. To resolve this problem, in addition to the challenges, receipts should also contain the output of a public random beacon. Only receipts printed at the polling station could possibly have a correct value for the random beacon (because the values of the beacon could not be known ahead of time), and so a pre-printed receipt with a bad signature would not have a correct random beacon value, and would be easily discredited.

This cascading set of attack vectors, and the resulting additions to the protocol--receipts with random beacon values and QR code signatures and receipt checkers--are extremely valuable to consider, and we have included them in our implementation. Details follow below.

4. Implementation Details

When implementing a paper that was written theoretically, it is natural that certain decisions be made that are not specified within the context of the original paper. This was certainly the case for this project. While Moran and Naor did an excellent job of anticipating heading off many of these questions by providing details of their protocol, there were necessarily some vagaries that we filled in. In addition, there were many details that we needed to fill in with respect to our augmentation to the system (described in Section 3). This section is dedicated to addressing these sorts of details.

A. Architecture

One of the chief design choices we needed to make was how best to structure the various aspects of our system, which needs not only to collect votes, but also tally and release them in an auditable manner. We decided on the following architecture:

The DRE is implemented through a server which may reside on an embedded device. For our proof of concept, we placed this on a Raspberry Pi, though of course if things were to be done at scale, it would be more reasonable to use a more capable machine or set of machines. Each voting machine essentially acts as its own DRE. In our proof of concept, we placed the DRE on the same machine that was acting as a voting machine. The voting machines themselves are responsible for tallying and forwarding vote totals and proofs, along with receipts, to the bulletin board.

Notably, we have also managed to hook up the voter interface to a receipt printer, which allows us to produce physical receipts. This both makes our proof of concept more realistic, and allowed us to test our receipt verification application more successfully.

The receipt verification app (discussed in Section 3) is responsible for reading the QR codes on the receipt and verifying that the signature is correct. In our proof of concept, the app is a standalone React Native application. However, it is important to stress that, unlike the DRE, the receipt verification application may well be written by a third party. This is important because, unlike the DRE, the receipt verification application requires trust from the voter. That is, a corrupt receipt verification system could mislead a user into believing that their receipt is correct (no analogous property is true of the DRE). So users should be able to choose one or multiple receipt verifications applications from an assortment--perhaps they wish to choose one sponsored by the party they align with, or perhaps they wish to write their own to establish the utmost trust. A side effect of this fact is that receipt verification applications may have as many features as their user desires. For instance, a receipt verification application could include an option to forward the receipt contents to a trusted third party who can check at a later date that the receipt has been published to the bulletin board properly. The version we implemented simply allows the user to verify that their receipt made it to the bulletin board. This feature, of course, is to be used after the receipt verification stage--likely later in the day, when the receipts have been published.

As noted above, voting machines forward their tally, proofs, and receipts to a bulletin board. As currently implemented, this is accomplished by posting this information on transfer.sh, a file hosting site, where they can be downloaded and added to the bulletin board. The link to the information is printed by the voting machine, and also emailed to election officials. Our bulletin board is a server consisting of two endpoints: a tally verifier endpoint, and a receipt locating endpoint. The tally verifier returns the tally if the proofs adequately corroborate the tally. The receipt locating endpoint receives POST requests, and replies acknowledging whether the requested receipt is or isn't on the bulletin board that is being tallied. In practice, this bulletin board should also be available from as many third party sources as possible, for the same reasons of decentralizing trust as described above. Because the verification information is posted online publicly (provided the links to the data on transfer.sh are shared), any party interested can download all the files, and use them to host their own bulletin board. Moreover, third parties could choose to provide only one of the two services (tally verification or receipt location), since each can be accomplished independently.

All our code, except for the receipt verification app, is in Python.

B. Cryptography

There were two areas where we identified potential vulnerabilities in the MN06 protocol where a dishonest DRE could attempt to cheat.

The first of these is in the Pedersen commitment itself. Pedersen commitment relies on the knowledge of two public parameters, g, and h, where both are generators of the same elliptic curve. The committer is responsible for determining these parameters. However, it's very important that the committer doesn't know log_g (h) -- if so, the binding property of the commitment is lost and the committer can pretend that they committed to any value of their choice. This issue was seen a few years ago with the random number generator DUAL_EC_DRBG, where the NSA published curve parameters in an NIST manual without any explanation of their derivation, and researchers realized soon after that the parameters could have been generated in such a way that the NSA had a backdoor into the output of the generator. The only way to address this issue is to generate parameters in such a way that they could not have been generated maliciously -- these are called "nothing-up-my-sleeve" numbers. We determined that g and h should be chosen by taking input strings that are very unlikely to have been chosen maliciously via exhaustive search (for example, we chose “nothing_up_my_sleeve”) and hashing them deterministically onto the elliptic curve via Icart’s method. It’s impossible to pick two English language strings such that one hashes into a point that you happen to know the discrete log of with regard to the other, so this should enable trust in the chosen system parameters.

Secondly, the zero-knowledge verification protocol in MN06 is specified interactively: first the election authority announces a masked permutation of commitments, then depending on the bit output by a trusted verifier either releases the masks and the permutation (proving that the tally is of the receipts and commitments on the bulletin board) or unmasks the permutation completely, revealing a permuted tally of the votes. The independence of the verifier is critical -- if the election authority knows the sequence of bits ahead of time, they can announce real masked permutations whenever they know they’ll be asked to release the masks and permutations, and announce a fake tally of votes whenever they know they’ll only be asked for the tally. Of course, this total dependence on a trusted verifier is problematic, as it creates a single point of failure for the entire protocol. We used the Fiat-Shamir heuristic to mitigate this issue.

Instead of using an interactive verification system, we modeled the cryptographic hash function SHA-512 as a random oracle. Then, we require the election authority to generate in advance the 512 masked permutations it wants to be challenged on, feed them into SHA-512, and use the resulting bitstring to determine which verification it’ll perform for each masked permutation. As it should be impossible to generate a series of masked permutations such that all the real ones correspond to 1 bits in the hash and all the fake ones correspond to 0 bits in the hash, this should provide excellent security that relies on the hardness assumptions underlying cryptographic hash functions rather than requiring trust in human organizations.

All our cryptographic functions were implemented using the elliptic curve functionality in the Python petlib[12] library.

C. Interface

The voter interface (that is, the interface which the voter is presented at the voting machine) is implemented as a Flask web application server. Each voting machine may then connect to this Flask server. While the voter is casting a vote, information about them (including their ID, their challenges, who they've voted for, etc.) is stored in a session variable. When they have cast their vote, their receipts and all the relevant information used to construct the tally proofs is stored also stored in a session variable, and the individual voter information is overwritten by the next voter. The session is managed by a Redis database on the backend.

When a new machine connects to the Flask server, its first step is to initialize the cryptographic primitives that will be used to create commitments and proofs, and then the machine initializes the candidates and a new list of vote data. Then, the machine is ready for users to input votes. As voters cast their votes, they are guided through a series of endpoints to the Flask server corresponding to the steps of voting described in MN06. Three word challenges are generated automatically from a dictionary culled from a list posted on github[13] (with some stop words removed), but users may also modify them. Voters may cancel an ongoing vote at any point, and restart the process. When a voter has finished casting their vote (and certified that their receipt printed), the vote (and all accompanying information) is appended to the list of vote data. Then the machine is ready for the next voter, or to be shut down (currently via a keystroke, though in practice, shutdown should require an authenticated election official's approval). When all voters have finished voting, and the machine is shutdown, it uses the list of vote data to construct the tally and its proof of the tally, then forwards this information (along

[12] https://github.com/gdanezis/petlib
[13] https://github.com/first20hours/google-10000-english

with receipts) to transfer.sh, where it can be downloaded and used for the bulletin board. Information about actual content of the votes (that is, who exactly voters voted for) is not forwarded anywhere.

A full walkthrough of the interface is provided in the Appendix.

D. Receipts

Our ‘receipts’ -- not technically receipts in the voting sense, since they cannot be used to prove who someone voted for, so not in violation of the protocol’s receipt freedom -- are printed using an Adafruit Mini Thermal Receipt Printer[14] interfaced with the Raspberry Pi over TTL serial. We had to modify the driver for this printer to be able to print QR codes of the correct size.

The receipts themselves contain the following data: Voter id, the commitment to the vote, the challenge strings for each candidate, a timestamp, the latest random beacon value (at the time of printing) from the NIST website, and two QR codes, one ‘plaintext’ QR code containing a plaintext encoding of all the previous field, and one signed QR code encoding a signature of the data in the plaintext code using the DRE’s private key, such that it can be decoded and verified using the DRE’s known public key. This is necessary to prevent the new attack vectors discussed previously.

A sample receipt is provided in the Appendix.

E. Receipt Verification App

The receipt verification app, implemented in Javascript-based React Native for cross-platform compatibility, serves two purposes. First, within the polling station, the user can use the app to verify that the receipt was generated after the voter entered the polling booth using the NIST beacon, and that the DRE has signed the receipt with the private key

[14] https://www.adafruit.com/product/597

counterpart to its known public key. These steps ensure that if the voter is not able to verify that their vote has been included in the tally, their challenge will be seen as legitimate given that a signed receipt generated after the voter entered the booth must be genuine. To do signature verification, the user is prompted to capture the two QR codes referenced above in order, first the ‘plaintext’ code and then the signed code. The app then prints out the contents of the plaintext code on the screen so that the voter can verify that the data in the human-readable text of the receipt is reflected in the QR code -- in the future, we hope to do this automatically using Optical Character Recognition (OCR). If the voter confirms that the text on the receipt matches the text generated from the QR code, the app proceeds to verify that the second QR code encodes a signed (with the DRE’s key) version of the data in the first QR code. If any of these equality checks fail, then the receipt is faulty, and the voter is able to challenge immediately. Crucially, the voter is only allowed to do this in the small interval of time between voting and leaving the polling booth, otherwise they would be able to make a fake challenge using a fabricated receipt with a purposely faulty code. As such, it is imperative that the app be able to help the user perform signature verification quickly and smoothly.

The second purpose of the app is checking that an individual vote is included in the tally on the online bulletin board after the election is over. This is accomplished by sending the data on a receipt to an API endpoint of the bulletin board. A user can select the “Verify Bulletin Board” option on the app, scan in the plaintext QR code, and immediately be apprised of whether their vote was counted or not.

Images of the app are provided in the Appendix.

F. Bulletin Board

Our bulletin board is written as a Flask app, which upon starting, performs the tally verification (using the library in section G), and then has two primary endpoints. The first endpoint (“/tally”) displays the tally resulting from the check that was performed upon starting. The second endpoint (“/verify”) accepts POST requests with a JSON dicts identifying the voter’s ID, their challenges, and the commitment from the voting machine. It then seeks the appropriate receipt associated with the voter ID, and returns JSON identifying whether the challenges and commitment on the bulletin board match those from the POST request.

G. Verifying Election Totals

We implemented a verifier library in Python using petlib based on the protocol in MN06, modified as described in section B. This library takes in a JSON file generated by a DRE, verifies the votes of that DRE and compares its calculated total to the DRE’s provided total, returning True iff both those conditions are satisfied.

H. Independent Implementations

The Bulletin Board, Verifier, and Receipt Verification app can and should all be implemented independently by different groups of people. Our vision is that political parties, nonpartisan vote watchdog organizations, and the government itself will each have separately implemented versions of all 3, ensuring that users can choose a provider they trust, or even independently verify using multiple versions. Hopefully, this will shield the system from both actual compromise and criticisms of compromise, tamping down on people ‘crying wolf’.

5. Future Directions

A. Testing with real people

It would be ideal to give our system a test run in an election with real voters (and potentially even real stakes). This would allow us to get feedback on potentially unintuitive portions of our user interface, and identify if there were any particular aspects that people less familiar with the system struggle with.

B. Open sourcing, community feedback, independent verifier implementations

Currently, the code for our project is maintained in two private repositories on Github. Ultimately, the code for any voting system of this nature should be released in an open source manner. We have not done this yet primarily because there are some aspects of the code which require further requirement. For example, keys used for signing receipts (along with keys used for other purposes elsewhere) have not been abstracted in such a way that anyone using our code could properly operate a secure system.

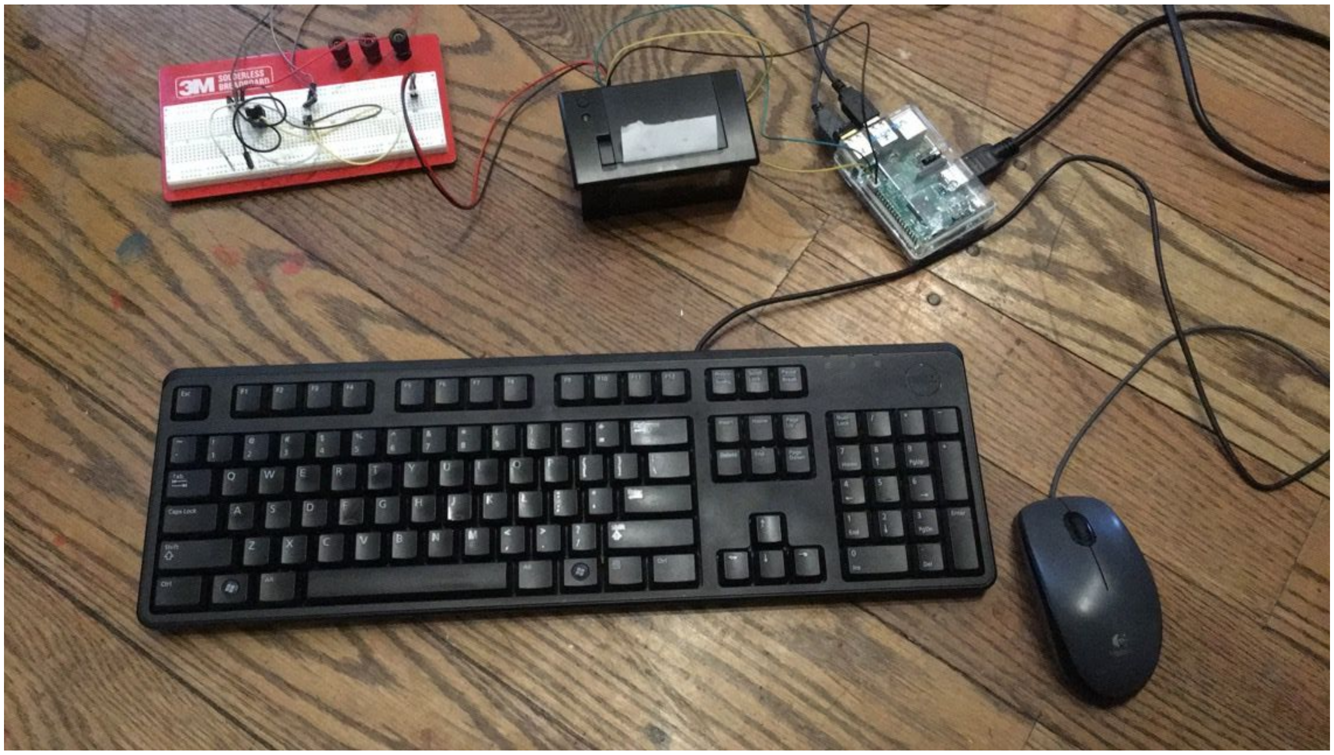
Open sourcing our code would be valuable in several ways. First, it would permit others to download the code and run their own elections. The benefits of this are numerous. Second, it would provide opportunity for community feedback--it is certainly possible that fresh sets of eyes could uncover a bug in our implementation--or perhaps just an inefficiency. Releasing the code to the public will thus help us make these improvements. Third, open sourcing our code will allow people to begin to write their own third party implementations of the receipt verifier and bulletin board (the advantage of which are discussed in the Architecture portion of section 4).

6. Conclusion

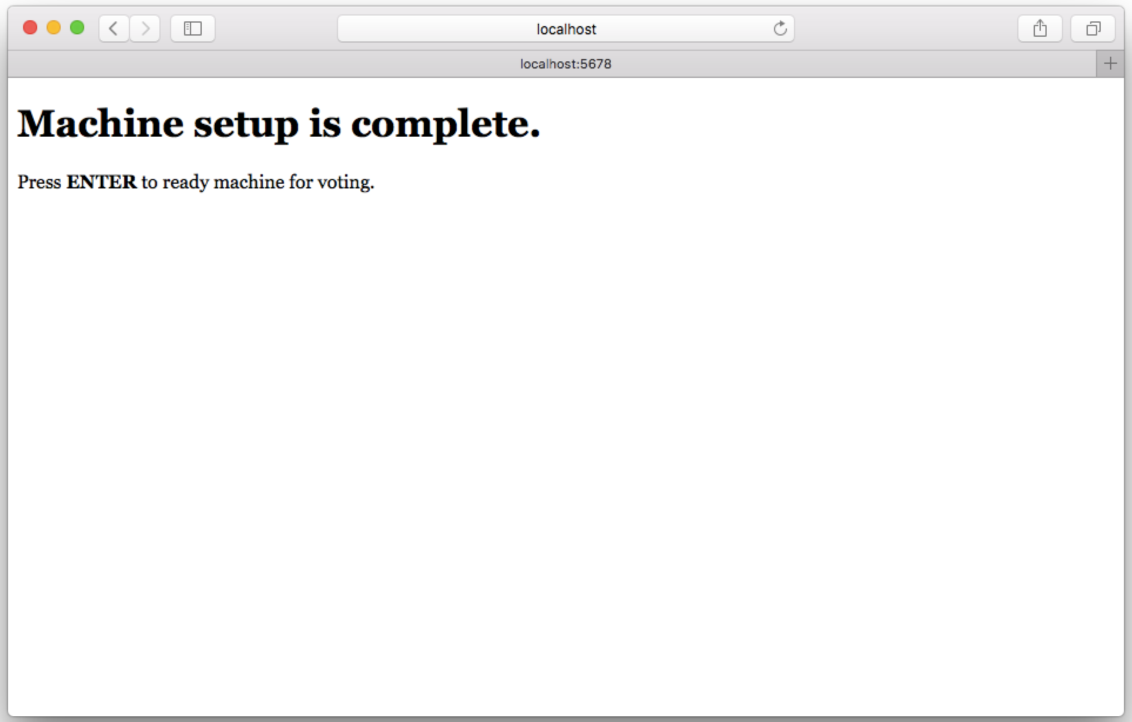
Over the course of the semester, we have not only implemented the protocol described in MN06--which allow for a receipt-free verifiable voting system--but expanded it to address concerns of corrupt voters attempting to delegitimize the election. Moreover, we have installed our implementation on a Raspberry Pi (simulating an embedded device), and successfully made use of a receipt printer to complete the demonstration of a physical voting machine. Our proof-of-concept demonstrates that a verified voting system is not only theoretically possible, but also feasible. Using machines like ours in countries around the world would reduce the possibility of errors in the voting process--for example, those that occurred in 2000. But perhaps more importantly, it would reduce the capacity for malicious cyber attacks on voting perpetrated either by nation-states or by determined organization that otherwise threaten either to change election results or undermine faith in the electoral process.

Appendix

A. Example Machine Setup



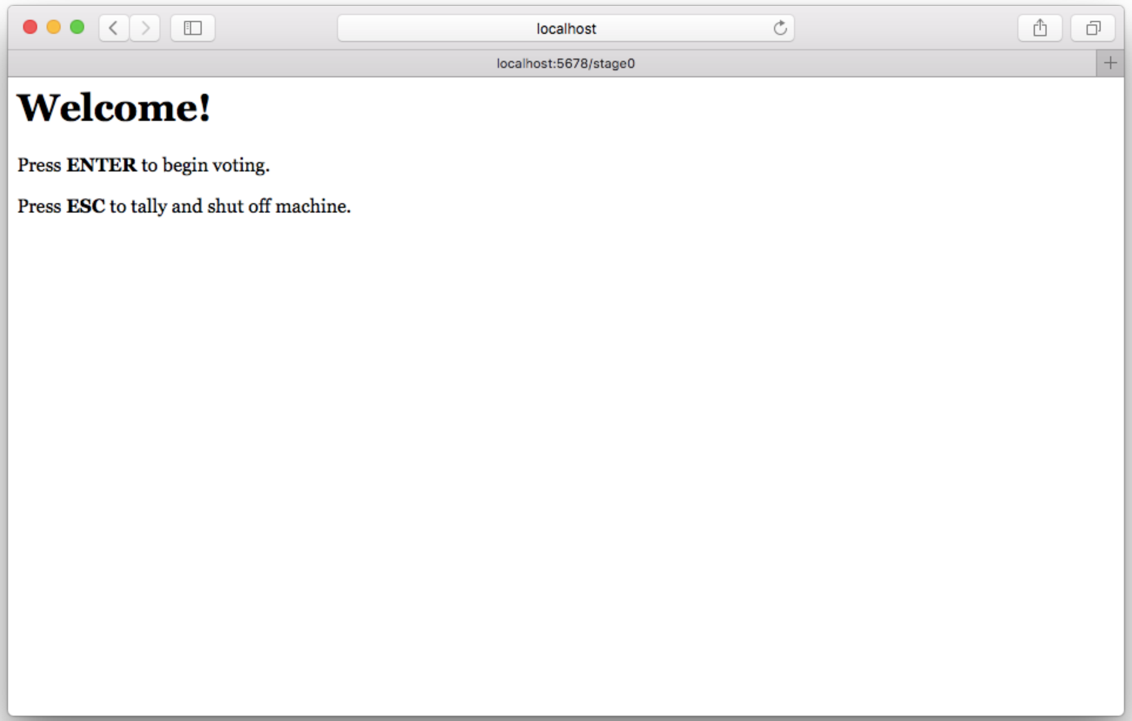
B. Voter’s perspective at the ballot box



Machine setup is complete.

Press **ENTER** to ready machine for voting.

This is the first thing seen upon connecting to the Flask web application. This means that the machine has set up the proper cryptographic premiums, and initialized the candidates, along with the list of data which will be accumulated as voters cast ballots.

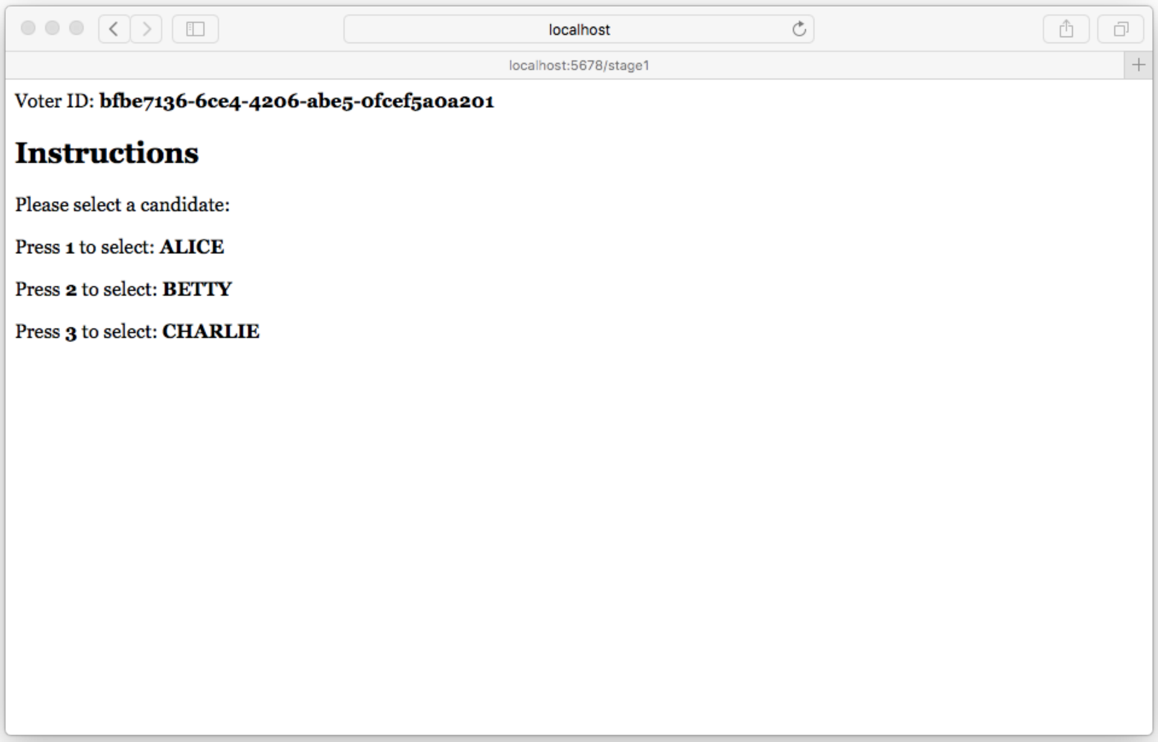


Welcome!

Press **ENTER** to begin voting.

Press **ESC** to tally and shut off machine.

This is the first thing a new voter sees. They then press ENTER, taking them to the first step of the voting process.



Voter ID: **bfbe7136-6ce4-4206-abe5-0fcef5a0a201**

Instructions

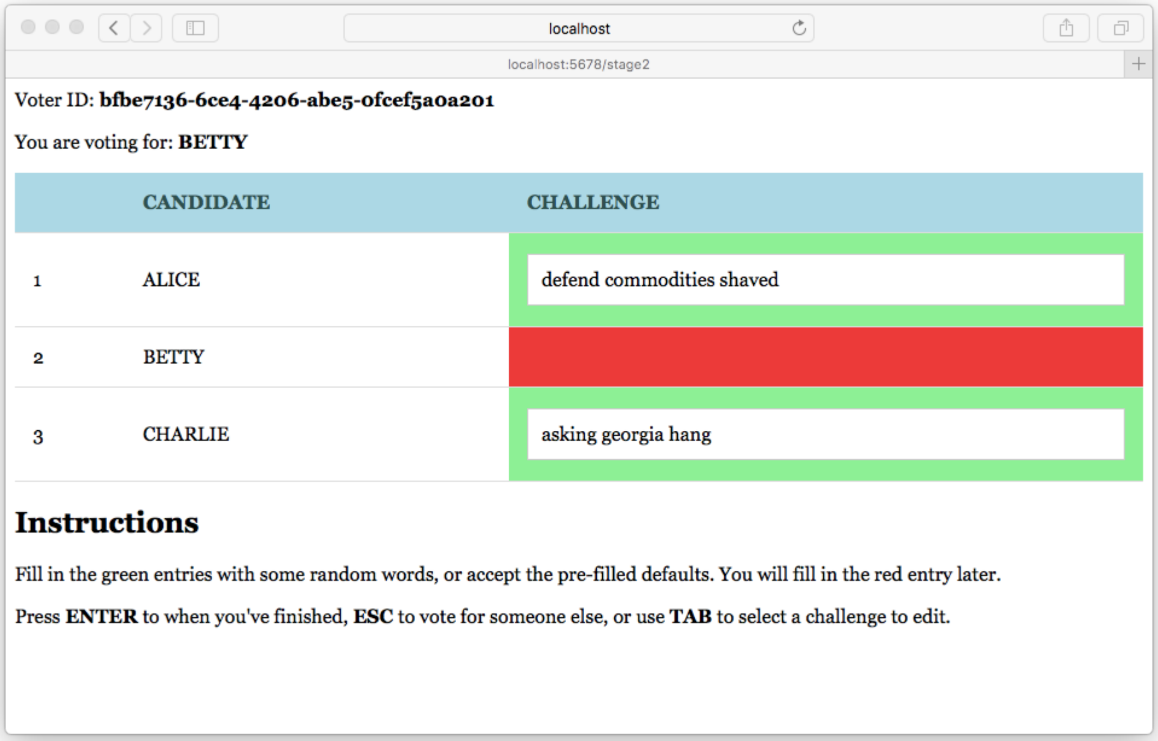
Please select a candidate:

Press **1** to select: **ALICE**

Press **2** to select: **BETTY**

Press **3** to select: **CHARLIE**

The first thing a voter does is pick a candidate. This is accomplished using the keyboard (no mouse or touchscreen interface necessary). From any point here on, the voter may return to the starting screen and terminate their vote by pressing ESC.



Voter ID: **bfbe7136-6ce4-4206-abe5-0fcef5a0a201**

You are voting for: **BETTY**

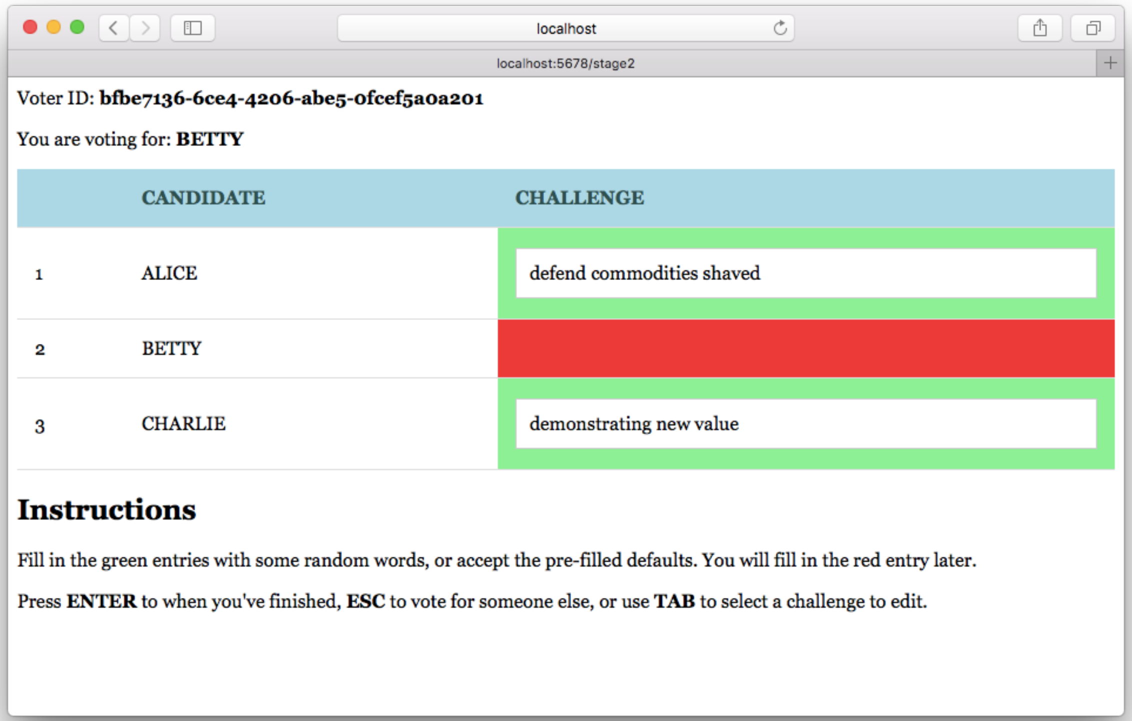
	CANDIDATE	CHALLENGE
1	ALICE	defend commodities shaved
2	BETTY	🟥
3	CHARLIE	asking georgia hang

Instructions

Fill in the green entries with some random words, or accept the pre-filled defaults. You will fill in the red entry later.

Press **ENTER** to when you've finished, **ESC** to vote for someone else, or use **TAB** to select a challenge to edit.

Next, the voter must submit challenges for the two candidates they did not vote for. Challenges are randomly populated from a dictionary, but can be modified using the keyboard.



Voter ID: **bfbe7136-6ce4-4206-abe5-0fcef5a0a201**

You are voting for: **BETTY**

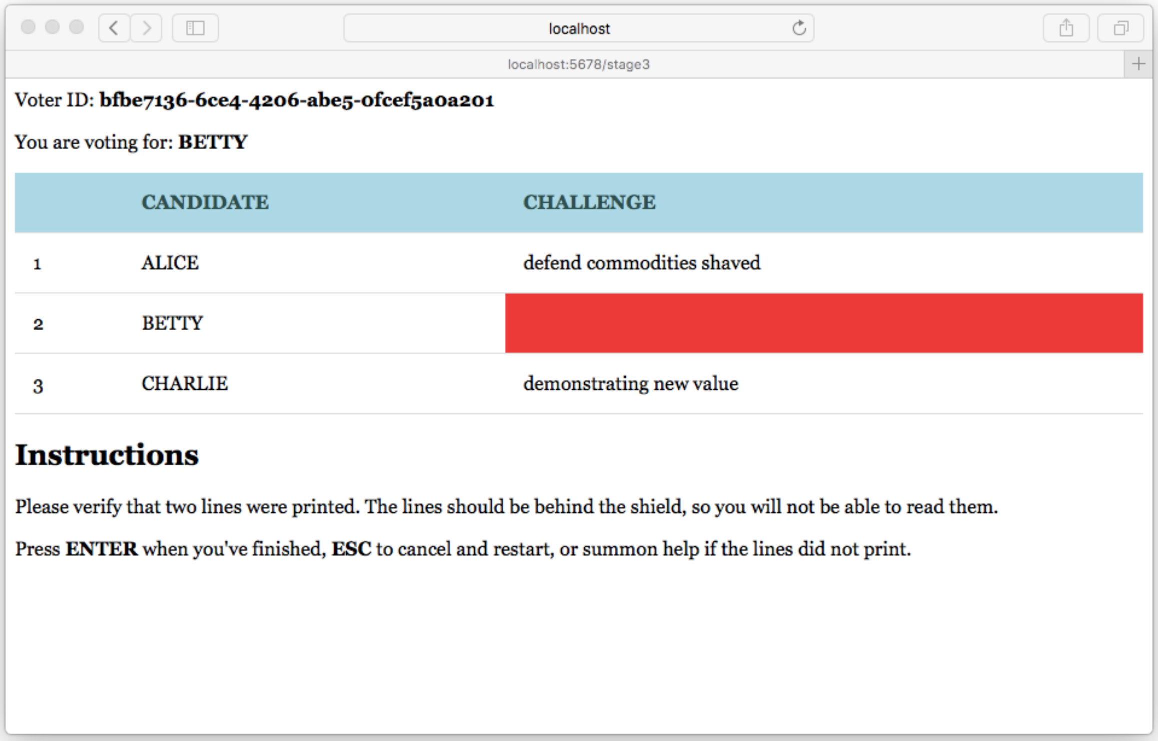
	CANDIDATE	CHALLENGE
1	ALICE	defend commodities shaved
2	BETTY	🟥
3	CHARLIE	demonstrating new value

Instructions

Fill in the green entries with some random words, or accept the pre-filled defaults. You will fill in the red entry later.

Press **ENTER** to when you've finished, **ESC** to vote for someone else, or use **TAB** to select a challenge to edit.

This is the same step as above, but demonstrates that a challenge (here, the challenge for Charlie) can be modified. This change will be populated throughout the remainder of the ballot casting.



Voter ID: **bfbe7136-6ce4-4206-abe5-0fcef5a0a201**

You are voting for: **BETTY**

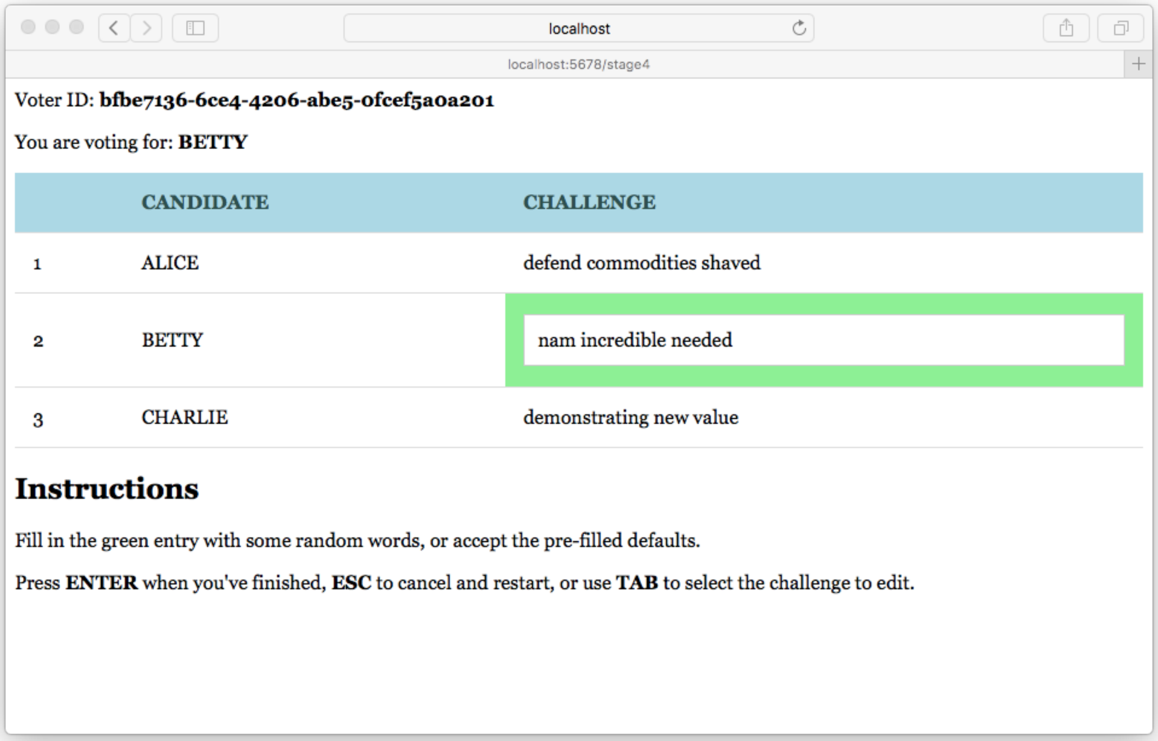
	CANDIDATE	CHALLENGE
1	ALICE	defend commodities shaved
2	BETTY	🟥
3	CHARLIE	demonstrating new value

Instructions

Please verify that two lines were printed. The lines should be behind the shield, so you will not be able to read them.

Press **ENTER** when you've finished, **ESC** to cancel and restart, or summon help if the lines did not print.

Now, the voter verifies that a portion of the receipt was printed. This portion is the machine's commitment to the challenges provided. These lines, however, should be behind a shield, so that the voter cannot enter this commitment into the third challenge (demonstrating to a third party who the voter cast a ballot for). Note that our third challenge ("demonstrating new value") has been persisted.



Voter ID: **bfbe7136-6ce4-4206-abe5-0fcef5a0a201**

You are voting for: **BETTY**

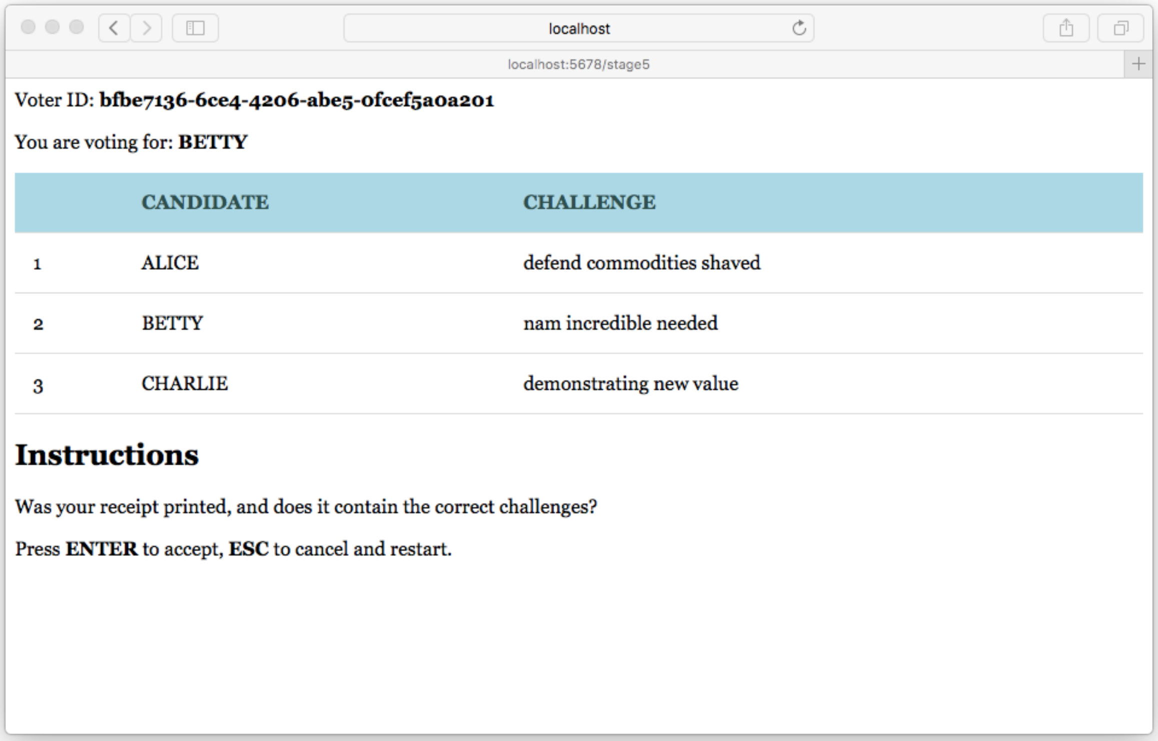
	CANDIDATE	CHALLENGE
1	ALICE	defend commodities shaved
2	BETTY	nam incredible needed
3	CHARLIE	demonstrating new value

Instructions

Fill in the green entry with some random words, or accept the pre-filled defaults.

Press **ENTER** when you've finished, **ESC** to cancel and restart, or use **TAB** to select the challenge to edit.

Next, the user inputs the third challenge, which is again randomly pre-filled, but changeable.


localhost

localhost:5678/stage5

Voter ID: **bfbe7136-6ce4-4206-abe5-0fcef5a0a201**

You are voting for: **BETTY**

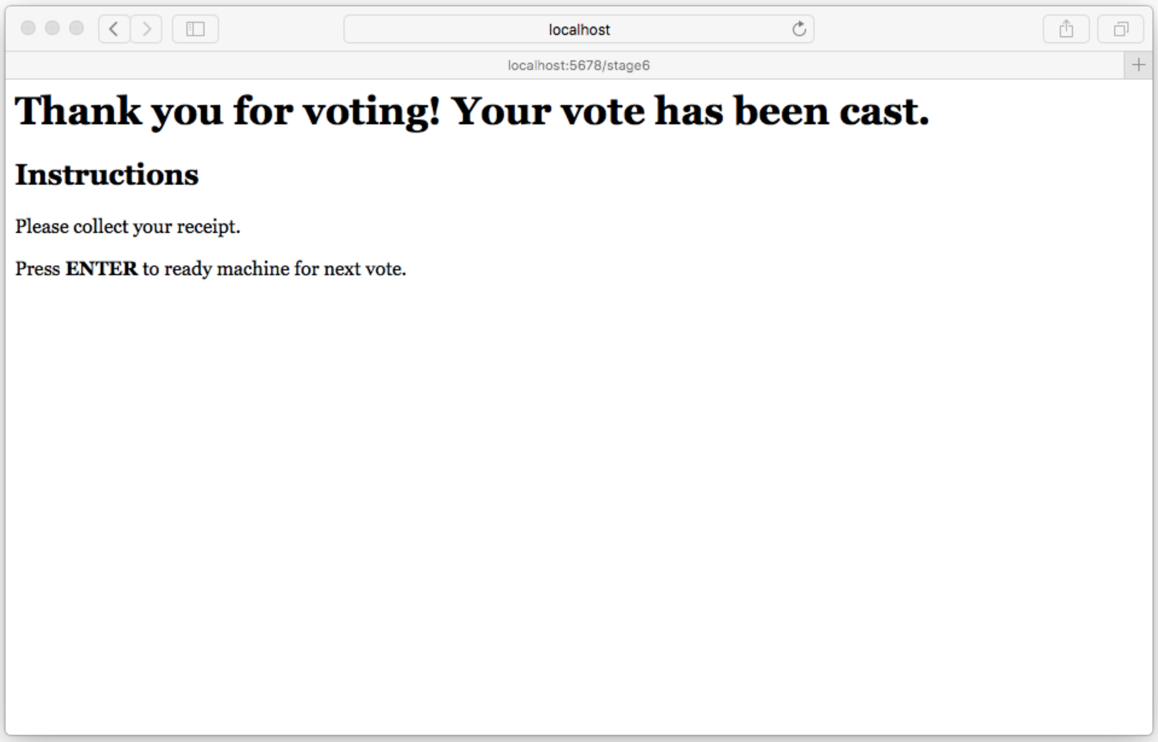
	CANDIDATE	CHALLENGE
1	ALICE	defend commodities shaved
2	BETTY	nam incredible needed
3	CHARLIE	demonstrating new value

Instructions

Was your receipt printed, and does it contain the correct challenges?

Press **ENTER** to accept, **ESC** to cancel and restart.

Here, the voter certifies that the receipt was printed, and looks to be correct (formal receipt verification occurs later, using a receipt verification app).



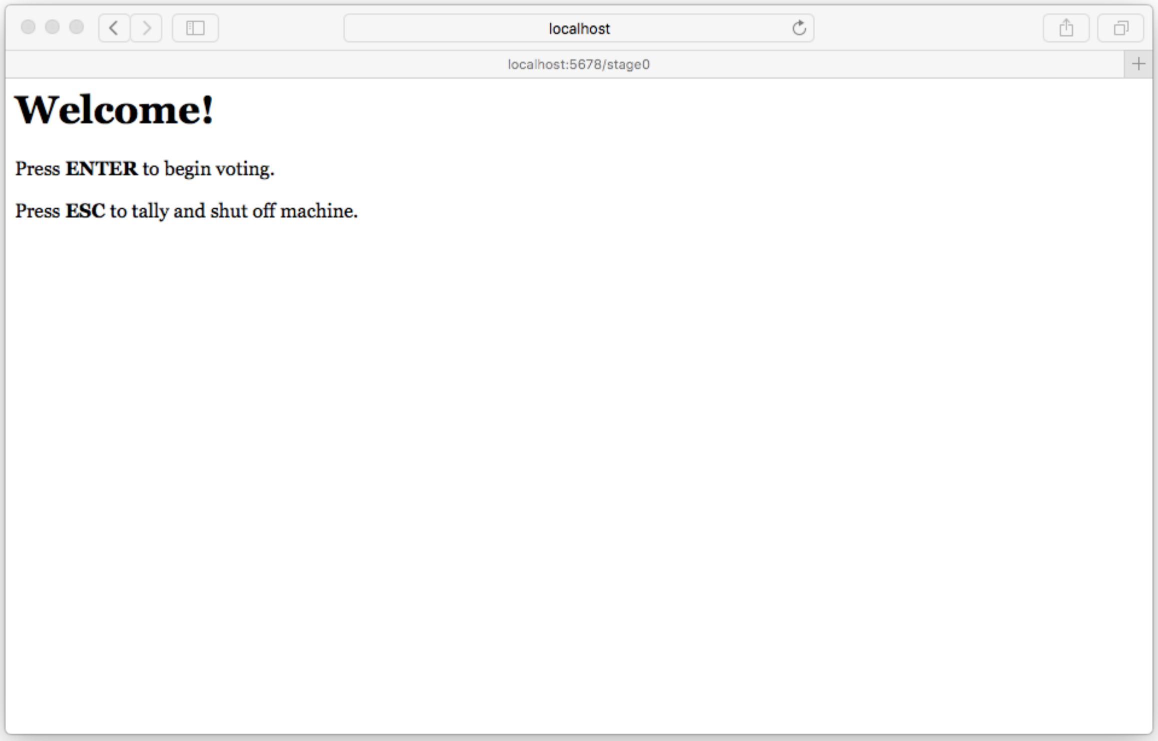
Thank you for voting! Your vote has been cast.

Instructions

Please collect your receipt.

Press **ENTER** to ready machine for next vote.

If the user is satisfied with their receipt, they have finished casting their ballot, and the machine may be prepared for a new vote.

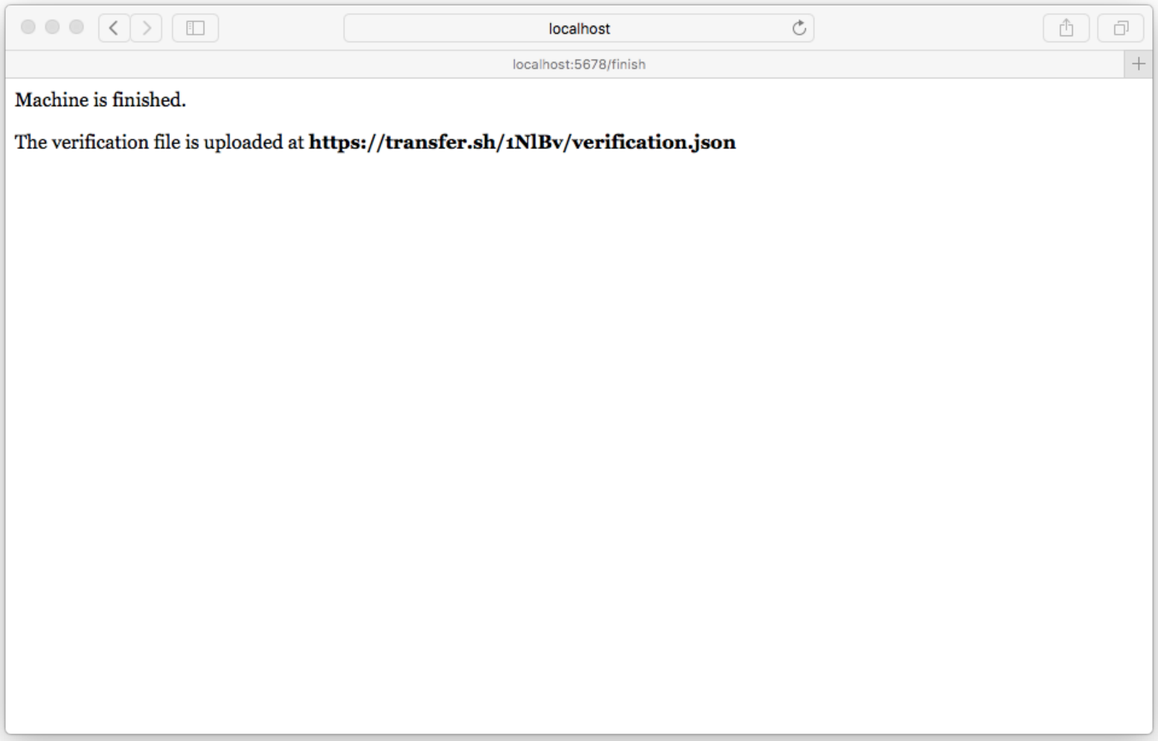


Welcome!

Press **ENTER** to begin voting.

Press **ESC** to tally and shut off machine.

After a vote is cast, the next voter may vote, and so on. When the machine is to be shutdown and the tally constructed, an election official may hit ESC to begin the shutdown process.

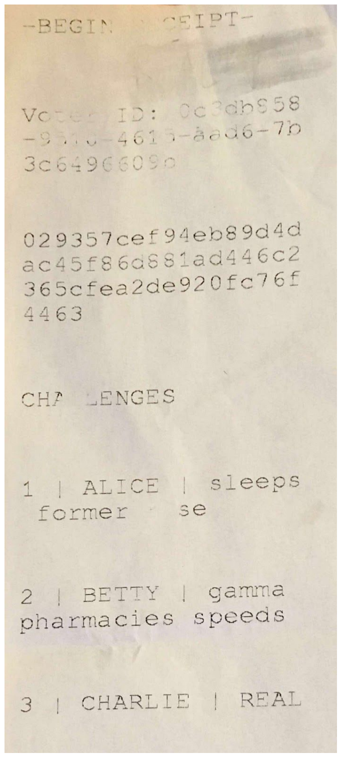


Machine is finished.

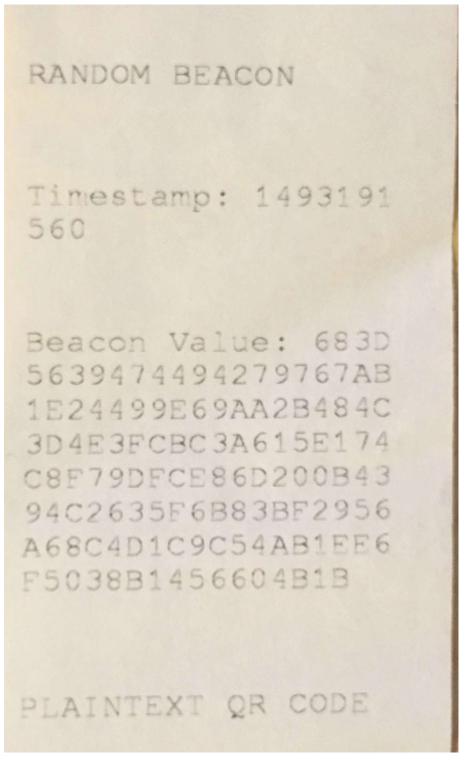
The verification file is uploaded at **https://transfer.sh/1NlBv/verification.json**

When the machine is shutdown, it constructs the proofs for the tally, and posts the tally, proofs, and receipts (along with some meta-information about the cryptography) online, emailing the link. This voting machine is now finished.

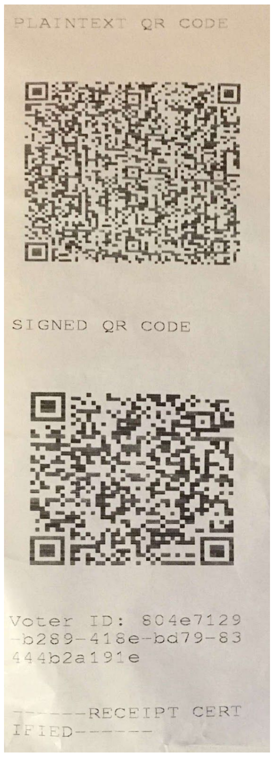
C. Correct Receipt



```
-BEGIN [illegible]CEIPT-

Vo[illegible] ID: 0c3db858
-9[illegible]-4615-aad6-7b
3c6496609b

029357cef94eb89d4d
ac45f86d881ad446c2
365cfea2de920fc76f
4463

CHA[illegible]LENGES

1 | ALICE | sleeps
 former [illegible]se

2 | BETTY | gamma
pharmacies speeds

3 | CHARLIE | REAL
```

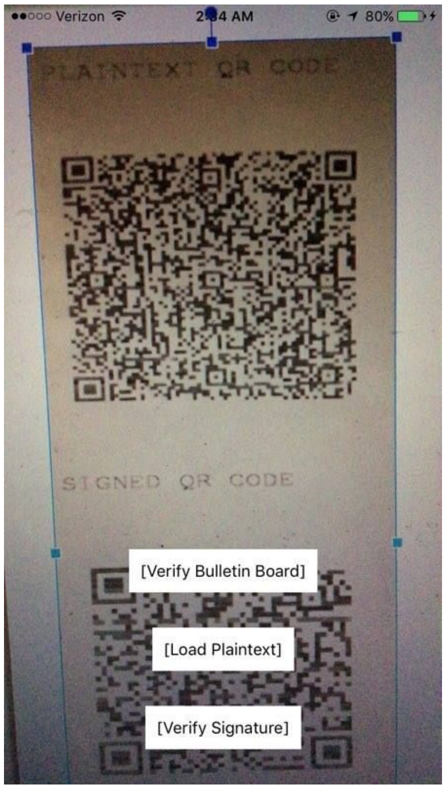


```
RANDOM BEACON

Timestamp: 1493191
560

Beacon Value: 683D
563947449427976 7AB
1E24499E69AA2B484C
3D4E3FCBC3A615E174
C8F79DFCE86D200B43
94C2635F6B83BF2956
A68C4D1C9C54AB1EE6
F5038B1456604B1B

PLAINTEXT QR CODE
```

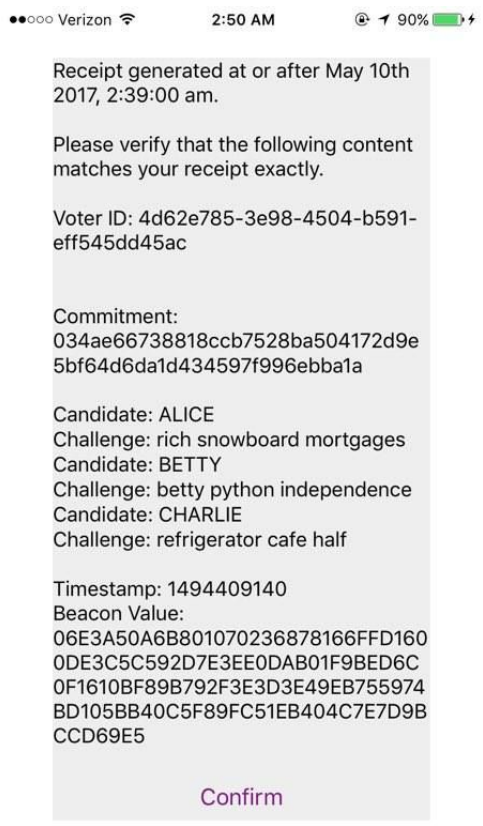


```
PLAINTEXT QR CODE

SIGNED QR CODE

Voter ID: 804e7129
-b289-418e-bd79-83
444b2a191e

------RECEIPT CERT
IFIED------
```

D. Receipt Verification Application



View of app homescreen



Receipt generated at or after May 10th 2017, 2:39:00 am.

Please verify that the following content matches your receipt exactly.

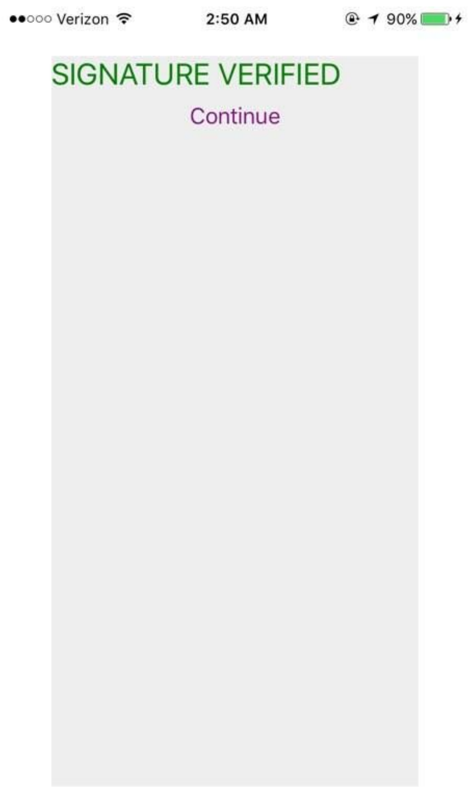
Voter ID: 4d62e785-3e98-4504-b591-eff545dd45ac

Commitment:
034ae66738818ccb7528ba504172d9e5bf64d6da1d434597f996ebba1a

Candidate: ALICE
Challenge: rich snowboard mortgages
Candidate: BETTY
Challenge: betty python independence
Candidate: CHARLIE
Challenge: refrigerator cafe half

Timestamp: 1494409140
Beacon Value:
06E3A50A6B801070236878166FFD1600DE3C5C592D7E3EE0DAB01F9BED6C0F1610BF89B792F3E3D3E49EB755974BD105BB40C5F89FC51EB404C7E7D9BCCD69E5

View of app after a plaintext QR code has been scanned -- the user must check that the text on the receipt is exactly the same.



View of app after the signature QR code has been scanned and verified to contain a signature of the plaintext data. The app would present a SIGNATURE FAILED TO VERIFY screen if verification fails.