Introduction to Encrypted Voting Protocols

Eric Bornstein

May 27, 2018

1 Abstract

In this paper I discuss the security goals desired in an election protocol, namely adding verifiability to the current system. I present a general cryptographic approach to building such a system and two schemes that follow this approach. Lastly, I give a paper-based non-cryptographic scheme that achieves similar security.

2 Introduction

In 2000, many Floridians went to their local polling place and voted for the next president by punching a hole into a card by the name of the candidate they were choosing. Some people did not fully punch a hole in their ballot, creating instead what is called a hanging chad. These were not always counted as votes, potentially affecting the winner of that state and therefore the next president of the United States. This led many people to ask how could we improve our voting system.

The obvious first thought was to no longer use punch card voting so that everyone's vote counts. However, we had already wanted every vote to count, so saying we want everyone's vote to count is not enough. Really, what we want is a way to know that the ballot we are casting will actually be counted the way that we wanted. This is the notion of verifiability. We want a way so that each voter can verify that his ballot was cast and counted as intended. We can also hope that we can create a voting system where anyone can verify that the election was run properly. Although it seems obvious, it is important to specify exactly what we want from an election. We want to verify that our election has the following properties:

- Voter's choice: The ballot cast by any voter should be a vote for the chosen candidate.
- One vote: Each voter should get one and only one vote. No one else should get a vote.
- Correct Tally: The total results of the election should the sum of how many votes were cast for each candidate.

Lastly, ever since the late 1800s, we have wanted our elections to be private. No one should be able to tell how you voted. Further, we don't want any voter to be able to prove to anyone how he voted. It may seem as if we only wanted privacy and this has been a side effect of privacy, but that is not the case. We do not want any voter to be able to prove to anyone that he voted a certain way because we want to avoid vote coercion. We do not want a third party to threaten voters who do not provide proof of how they voted.

In the rest of this paper, I will describe the general approach that cryptographers have taken to create an election system that satisfies our goals. I will show a scheme by Neff and one by Chaum that nearly follow this general approach. Lastly, I give a paper-based non-cryptographic approach to solving the same problem. Before I introduce these protocols, I will discuss some reasons why our elections are hard to make secure to keep these protocols in perspective.

3 Inherent Vulnerabilities in Voting

We would like a verifiable voting system that keeps all the properties mentioned above so that it is better than our current system. However, in addition to problems like hanging chads, our current system has multiple flaws. Some of these flaws are fixed, worsened, or unaffected by using cryptographic protocols. Many of the ideas in this section come from [12]. In reality, these probably deserve more attention in changing our voting system than trying to add verifiability. For each flaw, I will describe how this flaw relates to cryptographic voting.

3.1 Absentee Voting and the Secure Platform Problem

For absentee voting, registered voters who will not be able to vote at their local polling place can receive a ballot to mark and send back by mail. This provides an easy method of vote coercion because a third party can see the ballot. Although this is a large vulnerability, it is accepted because it allows more people to vote. Here it is a trade off between ease of voting and ease of attacking the system.

If we implemented cryptographic voting, absentee voting would be even more of a threat. Assuming the system requires a computer, a voter could let a third party run software that would vote. This is the same problem as before, but now a third party can have a much bigger impact. By releasing software, a third party could influence many absentee ballots.

This is an important notion. In voting, we really only need to know that the number of compromised votes is less than the margin of winning. It may be useful to know the true counts of votes, but we just need to be sure we have the correct winner. Even though in this scenario, each ballot is no more compromised than in the current system, it is possible for a third party to have a big enough impact to swing the election.

Having third party software is actually much bigger of a problem than just absentee ballots. Almost all cryptographic protocols rely on each party having a secure platform to perform computations. However, we really do not want voters to be using their own computers because they could have third party software on their computer, knowingly or not. Because of this, we want a protocol where a voter might only need a computer after she has voted. Instead, many systems have voters interact with a DRE (Direct Recording Electronic) voting machine without the use of a personal computer.

Although naively, absentee voting is a big problem, its vulnerabilities can be mitigated. In the elections in the United States, we vote by precinct, so the real reason many people have to vote absentee is because they aren't in their own precinct, so they can't vote there. In cryptographic protocols, the ballots that voters cast are just bit strings, which don't reveal the voters' choices. A registered voter could create a bit string in any precinct and have it sent to be counted in her true district. This may reduce the number of absentee ballots, thereby improving the security of the election even more so than verifiability. However, this is logistically quite difficult. It would require different precincts to be using the same voting system. Further, it would require the DREs to know public keys for each precinct's election and be able to switch between the keys.

3.2 Accessibility and Simplicity

In our current system, there are many people who cannot vote by themselves due to physical limitations. This may lead to either accidentally voting for the wrong candidate or needing a third party to assist them in voting. Any scheme, cryptographic or not, should try to accommodate these individuals.

As in the hanging chad case, there may also be voters physically capable of performing a task, but do not realize what they are expected to do. In the hanging chad example, they might not know that a hanging chad wouldn't be counted as a vote. This is an extreme problem with many cryptographic protocols. Security generally comes only if a party follows a protocol, so we would only want a cryptographic protocol if it were easy to use. In this case, we really only require voting to be done easily. It is not as big of a problem if it requires a strong understanding of cryptography to verify that the election was run properly.

4 A General Cryptographic Approach

In this section, I will describe a general approach that people have taken in designing encrypted voting protocols. I will describe the entities that are usually involved and the protocol that they follow. I will then discuss possible attacks and the security of the scheme.

4.1 Entities

The following are entities that are generally involved in these protocols.

- Trustees: a few agencies are selected as election trustees. These include competing political parties who are unlikely to collude to rig the election. Together, they will compute the results of the election.
- Voters: a set of people who are allowed to vote.
- DRE: the machine with which a voter interacts. These have some way of getting input from a voter. Also, they generally have a printer or some way of outputting information that they cannot change once outputted.

- Public Bulletin Board (PBB): a entity that maintains a publicly viewable list of all ballots cast.
- Poll Workers: helpers who facilitate the election and should be notified when there is a problem.

4.2 Protocol

In general, a cryptographic voting protocol can be broken down in to four parts: election initialization, voting, ballot tallying, and verification[8]. There are two general strategies for these voting protocols, either using homomorphic tallying or mix nets[3].

In election initialization, the trustees, using a multi-party functional, each generate a share of a secret key or keys. These shares will allow the trustees, in full collaboration or with a certain threshold of the trustees collaborating, to decrypt ballots.

In the voting stage, in a private area, voters will interact with a DRE. Each voter will tell the DRE which candidate he is selecting. The DRE will then encrypt this choice into a ballot using the keys generated in the election initialization phase. The DRE will then perform a zero-knowledge proof to the voter that the ballot reflects the voters choices. Then, the DRE will send the voter's ballot to the PBB. Also, the PBB can record who the voter was and from which DRE he voted. In the homomorphic tallying protocols, the DRE will also create a non-interactive zero-knowledge proof that the ballot is of the proper form. Lastly, the poll workers will keep track of who voted.

The ballot tallying stage is where the two types of protocols differ widely. First, I will cover the homomorphic tallying case. In this case, the ballots had been encrypted using a homomorphic scheme. A tallier, who can be a trustee or not, will look through the ballots and their proofs. For each valid proof, the tallier will combine the ballot with all the previous ballots. The ballots are combined using homomorphic encryption, such that the end result is an encryption of the total counts of the election. Once this is done, the election trustees will perform a multi-party functional to decrypt the results using the key shares from the election initialization.

The mix net schemes are different. Once the voting has ended, each trustee (or any mixer) will take a turn mixing the votes. The first trustee will take all of the ballots $x_1, \dots, x_n$ and turn them into $y_1, \dots, y_n$ such that there is a permutation σ and where $y_{\sigma(i)}$ is related to x_i. There are multiple choices for this relation. One possibility is that $y_{\sigma(i)}$ and x_i decrypt to the same value, possibly using different keys. Also, $y_{\sigma(i)}$ could be the decryption of x_i. Each trustee will also produce a non-interactive zero-knowledge proof that it mixed the ballots appropriately. After the last trustee performs its mixing, the result will be $z_1, \dots, z_n$ which are the votes of each voter in some order. These can then be counted to produce the total counts of the election.

The last step is election verifiability. In this stage, each voter can check if his ballot is correct on the PBB. Anyone can check that the PBB has only one ballot for only the voter who checked in on election day. For the homomorphic case, anyone can check the tallying. For the mix net case, anyone can check the proofs that each trustee mixed the ballots correctly.

There are many cryptographic protocols that have been designed, many following of this form or a similar form. The way they differ is in how they encrypt the votes and how the zero-knowledge proofs are performed. The computational costs of the zero-knowledge proofs are one of the most important factors in picking one protocol over another.

4.3 El Gamal Encryption

El Gamal encryption is one of the most famous encryption and is used in many of the voting protocols, so I will review it here. This is a public key encryption protocol [5]. We will have Alice be creating the keys so that people can send her encrypted messages.

To generate the keys, Alice will pick a $g \in \mathbb{Z}_p$ with order q for prime p, q. She will then pick $s \in_R \mathbb{Z}_q$. She will release $g, h = g^s$ as the public keys.

To encrypt a message m, which we can take to be in $< g >$ by some representation, Bob can pick $r \in_R \mathbb{Z}_q$ and compute g^r, mh^r.

To decrypt a message c_1, c_2, Alice will compute $c_2/(c_1^s)$. This will be $mh^r/(g^r)^s = m$, which will allow Alice to read the message.

This scheme is secure if the Diffie Hellman Decisional problem is hard. This would mean that g^r, g^s, g^{rs} is indistinguishable from g^r, g^s, g^t for random t.

El Gamal is also a useful homomorphic encryption because $(g^r, mh^r)(g^{r'}, m'h^{r'}) = g^{r+r'}, mm'h^{r+r'}$ and $mm'h^{r+r'}/(g^{r+r'})^s$ is mm', making this scheme multiplicatively homomorphic. For voting protocols, we only want a tally of how many votes for each candidate, so we want an additive homomorphism. There is a variant of El Gamal encryption called the exponential El Gamal encryption [3]. The difference is that instead of m, we will use g^m where m is the representation of the message in $\mathbb{Z}_q$. Now, multiplying encryptions of m, m', will get an encryption of $m + m'$. However, decryption is now harder because Alice has to find m from g^m. This is solving the discrete log problem, which is hard. However, if we know that the messages will be small, Alice can find m by trying all small possibilities. In many homomorphic voting protocol, if a voter wants to pick a candidate, the DRE will encrypt 1 otherwise 0. This means that the final message to decrypt will be between 0 and the number of voters, which is small enough to do an exhaustive search.

4.4 Security of the General Approach

I will present guarantees of security in cases where entities follow the protocol and will present some attacks if the parties do not. Some of the sections include what happens if a party not in the heading also colludes.

4.4.1 Security when DRE, PBB, and Trustees Follow the Protocol

If all parties follow the protocol, it is clear that each voter will get only one vote, that each ballot will represent the chosen candidate, and that the results will be the correct total. Now, I want to show that a third party cannot coerce a voter. The DRE only gives the voter a zero-knowledge proof of how he voted.

There are zero-knowledge proof systems where it is possible to simulate even a false statement. Using this, the voter could have simulated the proof that the DRE gave. Because the voter does not get a computer, it could be worthwhile to have the DRE give the voter a simulated proof that she voted each possible way. However, if the DRE is colluding, it could embed information into the proof, so seeing the proofs that the DRE created might be enough for a third party to know how one voted. Specifically, a third party can require a voter to provide a proof that she voted one way. In this proof, a DRE can hide information about who the voter picked such that the voter can simulate a proof, but not know if it has this needed hidden information.

This attack can be done on any system with a prototypical zero-knowledge proof. By prototypical, I mean a proof system that relies on the prover picking some randomness. This can be solved with tools such as unique zero-knowledge proofs, where the prover doesn't have any choices [9]. This is called a fair zero-knowledge proof, where the verifier is certain that any third will not learn anything from the proof transcript that it could have already learned.

The voter has no other information. The third party could check the ballot, but this is an encryption, so the third party cannot know who the voter picked. Lastly, after homomorphic tallying, the third party will only see the aggregated results. The third party can learn information from this, but we are okay having everyone learn the results.

If instead, there was a mix net, the third party cannot know which of the mixed ballots belongs to any voter. I need to be more specific on the election to make sure this secure. This will be secure if there are not very many possibilities for filling out a ballot. This assumption is the Short Ballot Assumption [13]. For example, if there are only two choices for candidates, a third party will not be able to know how any specific person voted. However, if the ballot had several offices, it is likely that there will be combinations of choices that no one will make. A third party could coerce someone to vote that way and will then know whether the voter listened. This coercion is even easier if there are write ins and multiple questions because it is very easy to write in a name that no one else will. It is important to note that this problem would exist in our current system, but in our current system, most third parties don't get to see any of the ballots.

4.4.2 Security when a Threshold of Trustees Collude

Most systems have the threshold be all the trustees. This would include competing political parties who are quite unlikely to collude.

But, If a threshold of trustees do collude, they cannot rig the election. This is because everything they do requires a non-interactive proof. This is still true no matter what entities are colluding with the trustees. However, with unbounded computation, they could potentially prove something false, letting them rig the election. But, they can learn how everyone voted. This seems like something unavoidable, but Neff showed a scheme where this is not the case [10].

In his scheme, each registered voter has a private key. Also, the voters are anonymous, so no one sees them voting. Before voting, the voter performs a mix net on a set of public keys. Specifically, a public key is of the form g^i, g^{is} for secret key s and any i. She then removes the public key related to her private key from the list and votes. Because the voter mixes the public keys, no one can know who is related to which key. Because the voters are doing this anonymously, no one can tell who is voting.

This construction does not seem to apply well to an election because it requires voters to have a private key and understand this system. Further, a third party could buy the private keys from the voters.

4.4.3 Security when some Trustees Collude

This will be no worse than when a threshold collude. Thus, they cannot rig the election. They also cannot read the ballots at the beginning because they can only be decrypted with a threshold of trustees. In the homomorphic case, they only learn the final result. In the mix net case, there is some trustee who performs the mix properly, so the trustees cannot learn which ballots are the same as the originals.

However, if enough trustees collude so that there is not enough honest trustees to reach the threshold, they can stop the results from being decrypted. This is bad, but is required given that if not, a smaller number of trustees than the threshold is able to decrypt. What we can hope for is that we could tell which trustees where breaking protocol. To do the mixes, the trustees have to provide proof that they mixed, so a trustee cannot fake this. Thus, the only chance they have of sabotaging the results without getting caught is in the decryption step. We can require them to release a hash of their key share at the beginning, and have all trustees prove that they can use their key share that has that hash to decrypt. Then, when it comes time to decrypt for real, each trustee must give a proof they are using a key share that has that hash. They cannot fake this unless they can find a collision in the hash. Thus, by using a collision resistant hash function, we can know which trustees are breaking the protocol.

4.4.4 Security when the PBB Break Protocol

Depending on the design of the PBB, there can be many attacks. It could add, change or delete ballots. Deleting or posting wrong ballots can be caught by the voters. If the PBB changes a ballot after posting it, this can be caught be verifiers. To stop it from adding ballots, we could require each ballot to be signed by a DRE. Now, the PBB would need to collude with a DRE. To stop this, we could keep track of who voted and cross check the names on the PBB with the names of voters. Thus, to cheat, they would need a poll worker to say someone voted who didn't. This trio of collusion can't be stopped, but this is basically equivalent to a poll worker committing identity fraud and voting under the other person's name.

Although these actions are caught, they would ruin the election. To mitigate this, we can have multiple PBBs. The main mitigation factor is a paper-based audit trail (paper ballots) that can be counted if a PBB or other party ruins the election.

A PBB could try to trick people by showing a different set of ballots to different people. Then, a voter might see her ballot, but the trustees wouldn't. This would be very dangerous because it wouldn't be caught. The design of the PBB can remove this risk. If all the trustees use a distributed system of data storage (such as in [2]), this risk can be removed. Doing this, one would need a threshold of the trustees to collude to have vulnerabilities.

Blockchains also are a natural candidate for the PBB. This way it would be difficult for the PBB to show people different sets of ballots except for showing people a subsequence of the blockchain. This use of blockchains differs in important ways from that of Bitcoin. In Bitcoin, users trust the longest blockchain whereas here, we use the chain from the PBB. Firstly, we do not want parties being able to rig the election just because they have a lot of computing power. Secondly, we are using the blockchain solely to be more sure that the PBB is not trying to show different people different ballot lists.

4.4.5 Security when the Poll Workers Break Protocol

Poll workers have two real jobs, marking who votes and seeing if machines are broken. If they say someone voted who didn't, this could be noticed by the person who didn't vote by looking at the PBB. However, this type of person likely will not look at the PBB. This could also be noticed by an outside group trying to check the election is run correctly. If this voter ended up coming to vote after the worker said she already did, the voter would know. We could also allow the voter to change her ballot. If the poll worker says someone didn't vote who actually did, this voter can check the PBB for his ballot. We can make the PBB note which people are counted as having voted. Then, the voter will notice.

The poll workers role with a broken machine does not affect the election too much. Because this would be catching a cheating machine, there would need to be rules to do in this case. This is related to what should happen if entities break the protocol. The important thing is that a voter will have noticed, so it will be known that there was a cheating machine.

4.4.6 Security when the DRE Breaks Protocol

The DRE cannot prove something false to a voter. If it tries, the voter should notice and show a poll worker. If the DRE does not send the correct ballot to the PBB, this will be caught. If it sends any extra ballot, this will be caught in the same way as if the PBB created it. So, the DRE cannot rig the election.

However, a voter may not be able to tell if the DRE actually cheated. This is because most systems use mathematical operations that a human could not do. To check if the DRE cheated, there could be third party groups helping check the proofs that the DRE gave. This will allow a voter to know if the DRE cheated, but the third party will not be able to tell how the voter voted because the zero-knowledge proof could have been simulated. Given that voters will almost certainly not simulate their own zero-knowledge proof, especially without a computer, it would be worthwhile for the DRE to simulate proofs that the voter chose all other candidates.

The real risk of a DRE being hacked is in the creation of a real ballot. A DRE can insert a random subliminal channel into the ballot, meaning that it hides information in the encryption of the vote [8]. For example, a DRE could keep trying randomness until the encryption ends with the voter's choice. This would reveal the voter's choice, making this not private.

Karloff, Sastry, and Wagner have some suggestions for fixing this problem[8]. One suggestion is giving the DRE a tape of randomness to use for each ballot. Each voter would also receive a hash of this randomness at the start. The DRE would then use a zero-knowledge proof that it used this randomness. This runs into the same problem as in section 4.4.1 and can be fixed accordingly. Another suggestion they have is finding voting protocols that do not rely on randomness, but they point out that more research needs to be done.

5 Neff's Voting Scheme

As I said earlier, most schemes follow similar lines, but diverge in the zero-knowledge proofs. Here I will present a mix net proposed by Neff, which lends itself to a voting scheme [11].

5.1 Neff's Mix Net

For lists $(X_1, Y_1), \ldots, (X_k, Y_k)$ and $(\overline{X}_1, \overline{Y}_1), \ldots, (\overline{X}_k, \overline{Y}_k)$ where each is an element of $< g >$ as of above, Neff showed a way to prove that there were $(\beta_1, \ldots, \beta_k)$ in $\mathbb{Z}_q$ where q is the prime order of g and a permutation π for which

$$(\overline{X}_i, \overline{Y}_i) = (g^{\beta_i} X_{\pi(i)}, h^{\beta_i} Y_{\pi(i)})$$

without revealing any information about β or π.

The system has seven rounds and is a public coin protocol. This means that the verifier picks randomness and sends it publicly to the verifier. Using the Fiat-Shamir heuristic, this can be turned into a non-interactive proof [6]. Allowing restarts to the proof, Neff's system is complete. A proof can be forged by an exponential-time adversary with less than $(2k + 3)/q$ probability. This is computationally zero-knowledge if the Decisional Diffie-Hellman problem is hard. Lastly, this takes linear time to prove and verify. Specifically, the number of exponentiations is $8k + 4$ and $12k + 4$ for proving and verifying.

If X_i, Y_i is a El Gamal encryption pair, now $\overline{X}_{\pi(i)}, \overline{Y}_{\pi(i)}$ is an El Gamal encryption for the same message.

This is not yet a mix net. To make it a mix net, a mixer will receive El Gamal encryptions $(X_1, Y_1), \ldots (X_k, Y_k)$ and will pick $\beta_1, \ldots, \beta_k$ and π randomly. Then, the mixer will compute $(\overline{X}_1, \overline{Y}_1), \ldots, (\overline{X}_k, \overline{Y}_k)$ and will then create a non-interactive proof of knowledge that this is correct.

5.2 Voting Scheme

Karloff, Sastry, and Wagner give a voting scheme designed by Neff using his mix net [8]. I will explain the main ideas of the voting step of the protocol.

The DRE will create a ballot with n rows for n candidates. In each row, there will be l pairs of El Gamal encryptions of 0 or 1. For the row corresponding to the chosen candidate, all pairs will be encryptions of the same bit. For all other rows, the encryptions in each pair should be of different bits.

The interaction between the voter and the DRE proceeds as follows. The voter will tell the DRE his candidate. The DRE will create a ballot of the form above. The DRE will then tell the voter what bits are encrypted for the row corresponding to the chosen candidate. The voter will then pick an encryption from each pair for the row corresponding to the chosen candidate. The DRE will reveal the randomness used to make those encryptions. I will call this opening the encryption. The DRE will open one encryption from each pair in the whole ballot and post it the ballot with the openings on the PBB.

Karloff, Sastry, and Wagner do not give the way the mix net is used, but I will present a way it could be used. After all votes have been cast, there can be a mix net for all the El Gamal encryptions in row i where the paired encryption was opened and was b. After mixing the encryptions, the trustees can open the encryptions and each b will be $1/l$ of a vote for candidate i and each $\bar{b}$ will not be a vote for the candidate.

5.3 Usability Features

Neff added in another piece to the voting process to make it easier for people to vote. Before the DRE sends the unopened ballot, it sends a hash of the ballot. Then it asks the voter if he wants to see proof that the ballot reflects what he wants. The point of this is that some voters will not understand this concept, so they won't want to see a proof. Because the DRE already had to commit to a ballot because it sent a hash, it will not be able to create a fake ballot. This way, because some people will want to check their ballots, the people who don't will also have security.

5.4 Difference from General Approach

This differs slightly from the general approach that I outlined. Specifically, the DRE did not perform a zero-knowledge proof to the voter. This technique was not a mistake because this method is similar to a zero-knowledge proof in the important ways. Notice, it is not a zero-knowledge proof in the sense that the voter would not have been able to determine which bits had been encrypted. The ways that it is similar is that the DRE proves to the voter that the ballot is a ballot for the voter's chosen candidate. Further, if the voter chooses which encryptions to open randomly, he will not be able to convince anyone who he voted for. This is because the other encryption in each pair could be the opposite bit of what the voter claims.

5.5 Buying Votes in Neff's Scheme

This difference between Neff's scheme and my general approach makes it much easier to prove that the ballot is correct in Neff's scheme. It solves the problem of having a zero-knowledge proof, but not the problem of subliminal channels. Also, it introduces a vulnerability. One vulnerability that I found in Neff's scheme, not mentioned by Karloff, Sastry, and Wagner, is a way for a third party to buy from the voter a proof that they voted a certain way. I explained why this was secure for an honest-voter. However, the voter need not use his own randomness. A third party could give a voter certain rules for picking which encryptions to open. Then, if the voter follows this rule, then the voter will be acting as a middle man while the DRE performs a proof for the third party. This will easily allow third parties to coerce voters.

Karloff, Sastry, and Wagner are not clear about which of the pairs in the other rows the DRE will open. For example, say the DRE just opens the same encryptions for each row, meaning if the voter picked to open the left encryption for row i, than all the left encryptions will be opened. A voter could pick the right encryption when the bit is 1 and the left when it is 0. The row corresponding to the voter's choice will likely (depending on l) be the only one where all the pairs satisfy this, showing a third party who the voter picked. Also, a voter would only need to interact with a third party after he already voted to prove how he voted.

This is a big flaw in the Neff system, but this type of attack can affect most interactive proof protocols that are not zero-knowledge. A classic non-zero-knowledge interactive proof where the verifier sends public coins can be turned into a non-interactive proof if the verifier uses a hash of the prover's message by the Fiat-Shamir heuristic [6]. This is very dangerous for voting because the verifier/voter would only need to interact with a third party after voting to show how he voted. I call this attack the Derandomization Attack because the voter is no longer using his own randomness as is expected in the protocol.

5.5.1 Protecting against the Derandomization Attack

It is pretty simple to fix this problem. We can do this by effectively making the voter choose his randomness at the beginning. Specifically, the voter will send a commitment of his randomness to the DRE at the beginning. Then, when the DRE expects randomness in the protocol, the voter will open the commitments.

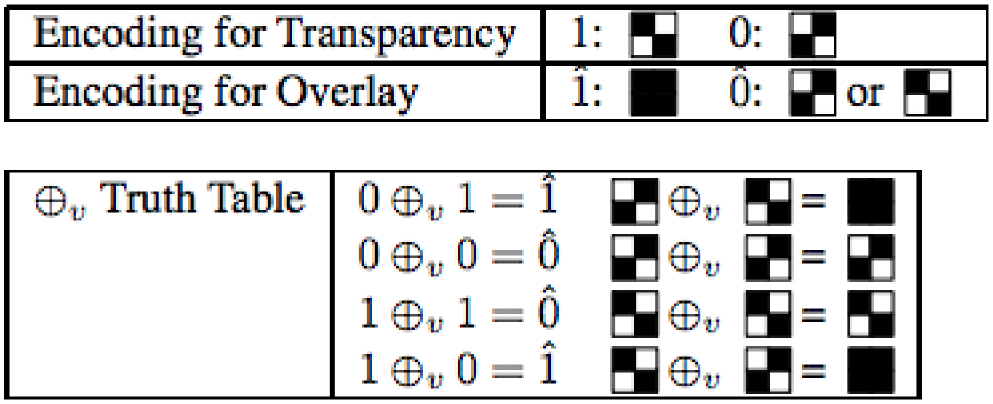
This seems like it would be a problem for a voting scheme because we don't want the voter to have any computer. However, we don't need to use a cryptographic commitment scheme. Instead, we can use a physical commitment scheme because the voter is only interacting with a machine.

This method would make a series of parallel zero-knowledge proofs also be zero-knowledge.

For the Neff protocol, this would let a third party at most choose which encryptions to open in the ballot. This is not enough to know how the voter voted. This would make the scheme secure assuming the DRE was following the protocol. If the DRE knew what choices the voter was going to make by colluding with the third party, the DRE could cheat in making the ballot to pick a candidate not chosen by the voter. However, the DRE gets no information about the voter besides his candidate until the randomness is revealed. So, the DRE cannot know if the voter is being coerced by the third party. Thus, the DRE cannot cheat in this way without risking an honest voter knowing that it cheated. Further, if a voter had a way of checking the encryptions real-time, he could check real-time if the DRE cheated. Assuming the DRE's output was printed and is connected to the DRE, a voter could show this to a poll worker to prove the DRE cheated. Also, if the voter could do this without sharing who he was trying to vote for by just showing the specific encryption that the DRE cheated on.

6 Chaum's Visual Cryptography Scheme

Now I will present another voting scheme. Chaum suggests a voting scheme that is based on visual cryptography so that it is easier for voters to follow the protocol [4].



Encoding for Transparency	1: ▚ 0: ▞
Encoding for Overlay	$\hat{1}$: ■ $\hat{0}$: ▞ or ▚

$\oplus_v$ Truth Table		
	$0 \oplus_v 1 = \hat{1}$	▞ $\oplus_v$ ▚ = ■
	$0 \oplus_v 0 = \hat{0}$	▞ $\oplus_v$ ▞ = ▞
	$1 \oplus_v 1 = \hat{0}$	▚ $\oplus_v$ ▚ = ▚
	$1 \oplus_v 0 = \hat{1}$	▚ $\oplus_v$ ▞ = ■

Figure 1: Chaum's visual cryptography [8]

6.1 Visual Cryptography

Visual cryptography is a way of visualizing bits. In Chaum's scheme, visual cryptography is used for humans to compute XOR. As input, the voter will receive two arrays that can be overlayed with the visual encoding of bits. Then, the voter will overlay the arrays, which will show him the XOR of the two layers. This is done as shown in figure 1.

6.2 Voting Protocol

When the voter starts to interact with the DRE, the DRE will show the voter three screens, a ballot screen visually representing all 0 and two screens visually representing pseudorandomness where the left and right screen will always XOR to the ballot screen. The DRE will also commit a doll for each screen. I will explain later what this is. Then, the voter gets to draw on the ballot screen and where he draws, the ballot will change to visually represent 1s. The left and right screens will change so that they always XOR to the ballot screen. Importantly, the ballot screen can be thought of as a checkerboard of black and white such that if the voter draws on a black square, it will change the left screen and if the voter draws on a white square, it will change the right screen. This checkerboard is unrelated to the actual bits on the screen and is just a way of understanding how the screens act.

Then, the DRE will print either the left or right screen as a transparency as chosen by the voter. The voter can check that this XORs with the screen he picked to get an all 0 array. Also, he should check that the transparency XORed with the other screen gives the ballot. When the voter checks this, the DRE will send the chosen transparency to the PBB. The DRE will also send the doll for the other screen, which will allow the trustees to remove the pseudorandomness. Lastly, the DRE will prove it generated the pseudorandomness in this transparency correctly. I will explain this in the next section.

Then, the trustees will perform a mix net while also removing the pseudorandomness. At the end, there will be a visual representation of each ballot, coming from the pixels in the transparency that were changed by the voter. Then, the results can easily be tabulated.

6.2.1 Pseudorandomness in the Voting Protocol

Here I will describe how the pseudorandomness is created based on [8]. As explained in the previous section, half of the pixels on each screen do not change. These are the pseudorandomness the DRE creates. It creates the pseudorandomness separately for each screen. I will explain how it does the pseudorandomness for the left screen, which is the same for right (by replacing l with r). Each DRE has a unique private key k_l. It creates pseudorandomness for each trustee i given by $sl_i = h(sign_{k_l}(ID), i)$ where ID is the ID of the voter or ballot, $sign$ is a public signature function, and h is a hash function. The pseudorandomness it uses is then $\bigoplus_i h'(sl_i)$ for a new hash function.

The doll is named after the Russian nesting dolls because it has the sl_i values nested in layers of encryption. This is also known as onion encryption. Specifically, the doll DL has

$$DL = E_{k_1}(sl_1, E_{k_2}(sl_2, \ldots (E_{k_t}(sl_t))))$$

where E is a public key encryption scheme and k_i is the public key for trustee i. This doll allows the trustees to remove the pseudorandomness. Also, they can check that the pseudorandomness was appropriately chosen because the can see if sl_i was correctly formed. This is done in the mix net, which is described in the next section.

At the end of the interaction, the DRE reveals the $sign_{k_l}(ID)$ if the voter picks the left screen. Now, the voter can check that the pseudorandomness in his transparency was picked correctly, but the voter needs a computer to do so.

A DRE that is trying to trick the voter must change the pseudorandomness on one screen or else the DRE couldn't do anything without lying about the ballot screen. If it changes one screen's pseudorandomness, it has at least a 1/2 chance of being caught.

As in the general scheme, creating the doll is an encryption using randomness that is susceptible to subliminal channels. Another vulnerability is if the DRE hides information by slightly changing the ballot screen, which would change the final ballot slightly [8]. This can be solved by limiting the possibilities for what drawings can be encoded.

It is important to note that the security of this scheme is ruined if anyone besides the DRE knows k_l or k_r.

6.3 Randomized Partial Check Mix Nets

The mix net used in this protocol is different from the previous one because it does not guarantee security, but guarantees it with high probability [7]. The way it works is that each trustee performs a mix on the ballots, removing the randomness and opening the doll. Once all trustees have mixed the ballots and the final ballots have been seen, the trustees will need to reveal how it mixed half of the ballots and how it removed the pseudorandomness. The way these ballots are chosen is a hash of what had happened previously. So, if not all the trustees collude, no trustee will know which ballots it will have to reveal. This way, they cannot cheat without getting caught with high probability. The ballots are also chosen so that each ballot has some mixer who does not reveal that ballot. If some mixers collude, they could know how someone voted. To fix this, we could have each trustee mix twice in a row and in the second mix only reveal the ballots not revealed in the first mix.

6.4 Security

As I have already shown, the DRE cannot lie about which candidate a ballot is picking and the mixing is valid. Now, I want to make it clear why a third party cannot coerce voters. First, if we assume that there are a small number of drawings accepted, the short ballot assumption holds and the final ballots will not help the third party know how a voter voted. Second, the transparency that the voter gets has half pseudorandom bits, which are not helpful. The rest of the bits are the drawing XORed with other pseudorandom bits, but the third party cannot get those bits without knowing $sign_{k_r}(ID)$ if the voter picked the left screen. Thus, a third party cannot know how someone voted.

7 Paper Based Voting

Although it is easy to quickly jump to trying to vote by using encryption, there are many ways to improve our current system while sticking with paper-based voting. Not only will these be simpler to understand, the machines need will likely be cheaper, making these protocols better suited to be real election schemes. I will describe the Vote/Anti-Vote/Vote (VAV), seen in [13], as I think it is one of the most understandable systems.

In this scheme, voters are given three unmarked ballots, two that say V and one that says A. The V ballots represent real votes. The A ballot is an anti-vote that cancels out a vote ballot of the same form. To fill out the ballots, a voter must have the A ballot exactly match one of the V ballots. From this, it is clear that if the election is run properly that the results will be the same as our current system.

The three ballots are cast together, so a voting machine needs to check if they are a valid trio. Attacking this machine could allow a voter to get three votes. Then, each ballot will get a random ID number, but the voter cannot see what they are. The voter is then allowed to take home a copy of one of the ballots with the ID. At the end of the election, all of the ballots will be posted with their IDs. This may require special design as in the PBB above.

If we do not want to trust a machine to check that the triple of ballots is valid, we can just have the voter show the Vote/Anti-Vote pair to a poll worker before putting it in the bin of ballots. We will need to make sure that both the ballots make it into the bin. If we only allow the voter to get one of these two ballots copied, a voter will clearly not be able to convince anyone how he voted. Further, given the short ballot assumption, a third party will not be able to tell if a V ballot was part of a Vote/Anti-Vote pair, so it will not know if it can change the ballot.

Rivest also suggests that it may be possible to have the IDs on the ballots at the start, but have them be complicated enough that a voter cannot remember them. However, if a voter can remember a small portion of the ID, it may be possible for the voter to tell a third party this small piece and be able to identify the other ballots. Instead, if the IDs were actually bar codes, a human would be unlikely to remember anything about the ID. Then, when they receive their copy, they can scan the barcode.

Because all of the ballots are posted, the results can be easily verified. The only risk is people changing, adding, or deleting the ballots. If we take a list of who voted as in the general approach, no one can add ballots without being caught; however, it will be impossible to know which were the added ballots, potentially ruining the election. Any one who changes or deletes a ballot has a 1/3 chance of affecting a copied ballot. This can be easily noticed.

A third party cannot coerce any voters because the voters can bring a copy of any ballot. Further, the other ballots are not connected to the copied ballot, so with the short ballot assumption, a third party cannot tell how a voter voted.

This scheme also has the benefit that it can be combined with our current system. Specifically, for people who do not understand the system, they can be given just one V ballot. This will be the same as voting in the current system. Their votes will be indistinguishable from V ballots from a VAV triple, so the security of these ballots comes from no one wanting to get caught changing a copied ballot of the VAV triple.

8 Conclusion

In this paper, I explained what security assurances we would like from a country-wide election. I discussed difficulties that our current system and other systems face. Then, I outlined a general approach that cryptographers have used to design cryptographic election protocols. Although this was secure in the ways that we could hope for, it is not useful on its own because this protocol would need more applied details before it could be implemented.

Because of the importance of the implementation I described two protocols, one from Neff and one from Chaum. These both followed the general approach very closely. They both differed in the same way. Namely, in the general approach, I said that the DRE would perform a zero-knowledge proof that the ballot represented the voter's choice. Instead, in both of these protocols, the voter is given a choice of what the ballot will say. With this choice, the DRE convinces the voter that the ballot represents the chosen candidate in a way that the voter cannot convince anyone else. This change from the general approach keeps the important notions of secrecy and verifiability of the ballot. Although the idea behind what these protocols are doing is more complicated than the general approach, they do so to lessen the computation needed to verify the election. It is important that the protocol does not have too high a demand for computation.

In all of these protocols, voters have the opportunity to check if the DRE has lied to them. However, all of the protocols have required a computer to do this check. It may be worthwhile to find protocols that are designed to work with and be checked by the limited human computational ability (as in [1]). Further in these protocols, the voter has a very limited task, but he may be easily tricked by a DRE not following the protocol [8].

Lastly, I acknowledged that we don't necessarily need to use cryptography to achieve our goals of a verifiable election. Specifically, I showed a paper-based protocol that gives us the same security. As is the usual engineering principle, this paper-based scheme seems to be a better protocol for the United States to adopt because it is far simpler than the others.

References

[1] Adida, Ben. *Advances in Cryptographic Voting Systems.* PhD thesis, MIT Department of EECS, August 2006.`http://assets.adida.net/research/phd-thesis.pdf`

[2] Adya, Atul; Bolosky, William J.; Castro, Miguel; Cermak, Gerald; Chaiken, Ronnie; Douceur, John R.; Howell, Jon; Lorch, Jacob R.; Theimer, Marvin; and Wattenhofer, Roger P. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *5th Symposium on Operating System Design and Implementation (OSDI)*, pages 1-14, December 2002.`https://www.usenix.org/legacy/event/osdi02/tech/full_papers/adya/adya.pdf`

[3] Bernhard, David and Warinschi, Bogdan. Cryptographic Voting–A Gentle Introduction `https://eprint.iacr.org/2016/765.pdf`

[4] Chaum, David. Secret-ballot receipts: True voterverifiable elections. *IEEE Security & Privacy Magazine*, 2(1):38-47, Jan.-Feb. 2004. `https://chaum.com/publications/Secret_Ballot_Receipts.pdf`

[5] El Gamal, Taher. A public key cryptosystem and a signature scheme based on discrete logarithms. In: *IEEE transactions on information theory*, Pages 469- 472, Volume 31, 1985. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.476.4791&rep=rep1&type=pdf`

[6] Fiat, Amos and Shamir, Adi. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: *Advances in Cryptology - CRYPTO '86*, pages 186-194, 1986. `https://www.math.uni-frankfurt.de/~dmst/teaching/SS2012/Vorlesung/Fiat.Shamir.pdf`

[7] Jakobsson, Markus; Juels, Ari; and Rivest, Ronald. Making mix nets robust for electronic voting by randomized partial checking. In *11th USENIX Security Symposium*, pages 339?353, August 2002. `https://people.csail.mit.edu/rivest/JakobssonJuelsRivest-MakingMixNetsRobustForElectronicVotingByRandomizedPartialCheckin` `pdf`

[8] Karlof, Chris; Sastry, Naveen; and Wagner, David. Cryptographic Voting Protocols: A Systems Perspective In *Proceedings 14th USENIX Security Symposium* (August 2005). `https://www.usenix.net/legacy/events/sec05/tech/full_papers/karlof/karlof.pdf`

[9] Lepinski, Matt; Micali, Silvio; Shelat, Abhi. Fair-zero knowledge. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, May 2005.`https://dl.acm.org/citation.cfm?id=2140023`

[10] Neff, Andrew C. A Verifiable Secret Shuffle and its Application to E-Voting. In *8th ACM Conference on Computer and Communications Security (CCS 2001)*, pages 116-125, November 2001. `http://web.cs.elte.hu/~rfid/p116-neff.pdf`

[11] Neff, Andrew C. Verifiable Mixing (Shuffling) of ElGamal Pairs `http://courses.csail.mit.edu/6.897/spring04/Neff-2004-04-21-ElGamalShuffles.pdf`

[12] Rivest, Ronald L. Electronic Voting. `https://people.csail.mit.edu/rivest/Rivest-ElectronicVoting.pdf`

[13] Rivest, Ronald L. and Smith, Warren D. Three Voting Protocols: ThreeBallot, VAV, and Twin. https://people.csail.mit.edu/rivest/RivestSmith-ThreeVotingProtocolsThreeBallotVAVAndTwin.pdf