CHAPTER 6

(FROM "APPLIED CRYPTOGRAPHY", BY BRUCE SCHNEIER, 2nd EDITION)
SECTION 6.1 DESCRIBES THE SECURE VOTING PROTOCOL TO BE USED FOR THE CLASS PROJECT.

Esoteric Protocols

6.1 Secure Elections

Computerized voting will never be used for general elections unless there is a protocol that both maintains individual privacy and prevents cheating. The ideal protocol has, at the very least, these six requirements:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. No one can determine for whom anyone else voted.
4. No one can duplicate anyone else's vote. (This turns out to be the hardest requirement.)
5. No one can change anyone else's vote without being discovered.
6. Every voter can make sure that his vote has been taken into account in the final tabulation.

Additionally, some voting schemes may have the following requirement:

7. Everyone knows who voted and who didn't.

Before describing the complicated voting protocols with these characteristics, let's look at some simpler protocols.

Simplistic Voting Protocol #1

(1) Each voter encrypts his vote with the public key of a Central Tabulating Facility (CTF).
(2) Each voter sends his vote in to the CTF.
(3) The CTF decrypts the votes, tabulates them, and makes the results public.

This protocol is rife with problems. The CTF has no idea where the votes are from, so it doesn't even know if the votes are coming from eligible voters. It has no idea if eligible voters are voting more than once. On the plus side, no one can change anyone else's vote; but no one would bother trying to modify someone else's vote when it is far easier to vote repeatedly for the result of your choice.

Simplistic Voting Protocol #2

(1) Each voter signs his vote with his private key.
(2) Each voter encrypts his signed vote with the CTF's public key.
(3) Each voter sends his vote to a CTF.
(4) The CTF decrypts the votes, checks the signatures, tabulates the votes, and makes the results public.

This protocol satisfies properties one and two: Only authorized voters can vote and no one can vote more than once—the CTF would record votes received in step (3). Each vote is signed with the voter's private key, so the CTF knows who voted, who didn't, and how often each voter voted. If a vote comes in that isn't signed by an eligible voter, or if a second vote comes in signed by a voter who has already voted, the facility ignores it. No one can change anyone else's vote either, even if they intercept it in step (3), because of the digital signature.

The problem with this protocol is that the signature is attached to the vote; the CTF knows who voted for whom. Encrypting the votes with the CTF's public key prevents anyone from eavesdropping on the protocol and figuring out who voted for whom, but you have to trust the CTF completely. It's analogous to having an election judge staring over your shoulder in the voting booth.

These two examples show how difficult it is to achieve the first three requirements of a secure voting protocol, let alone the others.

Voting with Blind Signatures

We need to somehow dissociate the vote from the voter, while still maintaining authentication. The blind signature protocol does just that.

(1) Each voter generates 10 sets of messages, each set containing a valid vote for each possible outcome (e.g., if the vote is a yes or no question, each set contains two votes, one for "yes" and the other for "no"). Each message also contains a randomly generated identification number, large enough to avoid duplicates with other voters.
(2) Each voter individually blinds all of the messages (see Section 5.3) and sends them, with their blinding factors, to the CTF.
(3) The CTF checks its database to make sure the voter has not submitted his blinded votes for signature previously. It opens nine of the sets to check that they are properly formed. Then it individually signs each message in the set. It sends them back to the voter, storing the name of the voter in its database.

(4) The voter unblinds the messages and is left with a set of votes signed by the CTF. (These votes are signed but unencrypted, so the voter can easily see which vote is "yes" and which is "no.")

(5) The voter chooses one of the votes (ah, democracy) and encrypts it with the CTF's public key.

(6) The voter sends his vote in.

(7) The CTF decrypts the votes, checks the signatures, checks its database for a duplicate identification number, saves the serial number, and tabulates the votes. It publishes the results of the election, along with every serial number and its associated vote.

A malicious voter, call him Mallory, cannot cheat this system. The blind signature protocol ensures that his votes are unique. If he tries to send in the same vote twice, the CTF will notice the duplicate serial number in step (7) and throw out the second vote. If he tries to get multiple votes signed in step (2), the CTF will discover this in step (3). Mallory cannot generate his own votes because he doesn't know the facility's private key. He can't intercept and change other people's votes for the same reason.

The cut-and-choose protocol in step (3) is to ensure that the votes are unique. Without that step, Mallory could create a set of votes that are the same except for the identification number, and have them all validated.

A malicious CTF cannot figure out how individuals voted. Because the blind signature protocol prevents the facility from seeing the serial numbers on the votes before they are cast, the CTF cannot link the blinded vote it signed with the vote eventually cast. Publishing a list of serial numbers and their associated votes allows voters to confirm that their vote was tabulated correctly.

There are still problems. If step (6) is not anonymous and the CTF can record who sent in which vote, then it can figure out who voted for whom. However, if it receives votes in a locked ballot box and then tabulates them later, it cannot. Also, while the CTF may not be able to link votes to individuals, it can generate a large number of signed, valid votes and cheat by submitting those itself. And if Alice discovers that the CTF changed her vote, she has no way to prove it. A similar protocol, which tries to correct these problems, is [1195,1370].

Voting with Two Central Facilities

One solution is to divide the CTF in two. Neither party would have the power to cheat on its own.

The following protocol uses a Central Legitimization Agency (CLA) to certify voters and a separate CTF to count votes [1373].

(1) Each voter sends a message to the CLA asking for a validation number.

(2) The CLA sends the voter back a random validation number. The CLA maintains a list of validation numbers. The CLA also keeps a list of the validation numbers' recipients, in case someone tries to vote twice.

(3) The CLA sends the list of validation numbers to the CTF.

(4) Each voter chooses a random identification number. He creates a message with that number, the validation number he received from the CLA, and his vote. He sends this message to the CTF.

(5) The CTF checks the validation number against the list it received from the CLA in step (3). If the validation number is there, the CTF crosses it off (to prevent someone from voting twice). The CTF adds the identification number to the list of people who voted for a particular candidate and adds one to the tally.

(6) After all votes have been received, the CTF publishes the outcome, as well as the lists of identification numbers and for whom their owners voted.

Like the previous protocol, each voter can look at the lists of identification numbers and find his own. This gives him proof that his vote was counted. Of course, all messages passing among the parties in the protocol should be encrypted and signed to prevent someone from impersonating someone else or intercepting transmissions.

The CTF cannot modify votes because each voter will look for his identification string. If a voter doesn't find his identification string, or finds his identification string in a tally other than the one he voted for, he will immediately know there was foul play. The CTF cannot stuff the ballot box because it is being watched by the CLA. The CLA knows how many voters have been certified and their validation numbers, and will detect any modifications.

Mallory, who is not an eligible voter, can try to cheat by guessing a valid validation number. This threat can be minimized by making the number of possible validation numbers much larger than the number of actual validation numbers: 100-digit numbers for a million voters, for example. Of course, the validation numbers must be generated randomly.

Despite this, the CLA is still a trusted authority in some respects. It can certify ineligible voters. It can certify eligible voters multiple times. This risk could be minimized by having the CLA publish a list of certified voters (but not their validation numbers). If the number of voters on this list is less than the number of votes tabulated, then something is awry. However, if more voters were certified than votes tabulated, it probably means that some certified people didn't bother voting. Many people who are registered to vote don't bother to cast ballots.

This protocol is vulnerable to collusion between the CLA and the CTF. If the two of them got together, they could correlate databases and figure out who voted for whom.

Voting with a Single Central Facility

A more complex protocol can be used to overcome the danger of collusion between the CLA and the CTF [1373]. This protocol is identical to the previous one, with two modifications:

— The CLA and the CTF are one organization, and

— ANDOS (see Section 4.13) is used to anonymously distribute validation numbers in step (2).

Since the anonymous key distribution protocol prevents the CTF from knowing which voter got which validation number, there is no way for the CTF to correlate validation numbers with votes received. The CTF still has to be trusted not to give validation numbers to ineligible voters, though. You can also solve this problem with blind signatures.

Improved Voting with a Single Central Facility

This protocol also uses ANDOS [1175]. It satisfies all six requirements of a good voting protocol. It doesn't satisfy the seventh requirement, but has two properties additional to the six listed at the beginning of the section:

7. A voter can change his mind (i.e., retract his vote and vote again) within a given period of time.
8. If a voter finds out that his vote is miscounted, he can identify and correct the problem without jeopardizing the secrecy of his ballot.

Here's the protocol:

(1) The CTF publishes a list of all legitimate voters.

(2) Within a specified deadline, each voter tells the CTF whether he intends to vote.

(3) The CTF publishes a list of voters participating in the election.

(4) Each voter receives an identification number, I, using an ANDOS protocol.

(5) Each voter generates a public-key/private-key key pair: k, d. If v is the vote, he generates the following message and sends it to the CTF:

$$I, E_k(I, v)$$

This message must be sent anonymously.

(6) The CTF acknowledges receipt of the vote by publishing:

$$E_k(I, v)$$

(7) Each voter sends the CTF:

$$I, d$$

(8) The CTF decrypts the votes. At the end of the election, it publishes the results of the election and, for each different vote, the list of all $E_k(I, v)$ values that contained that vote.

(9) If a voter observes that his vote is not properly counted, he protests by sending the CTF:

$$I, E_k(I, v), d$$

(10) If a voter wants to change his vote (possible, in some elections) from v to v', he sends the CTF:

$$I, E_k(I, v'), d$$

A different voting protocol uses blind signatures instead of ANDOS, but is essentially the same [585]. Steps (1) through (3) are preliminary to the actual voting. Their

(h) He attaches a new random string to the result of step (g) and encrypts it with Bob's public key. He records the value of the random string.

(i) He attaches a new random string to the result of step (h) and encrypts it with Alice's public key. He records the value of the random string.

If E is the encryption function, R_i is a random string, and V is the vote, his message looks like:

$$E_A\{R_5,E_B\{R_4,E_C\{R_3,E_D\{R_2,E_A\{E_B\{E_C\{E_D\{V,R_1\}\}\}\}\}\}\}\}$$

Each voter saves the intermediate results at each point in the calculation. These results will be used later in the protocol to confirm that his vote is among those being counted.

(2) Each voter sends his message to Alice.

(3) Alice decrypts all of the votes with her private key and then removes all of the random strings at that level.

(4) Alice scrambles the order of all the votes and sends the result to Bob.

Each vote now looks like this:

$$E_B\{R_4,E_C\{R_3,E_D\{R_2,E_A\{E_B\{E_C\{E_D\{V,R_1\}\}\}\}\}\}\}$$

(5) Bob decrypts all of the votes with his private key, checks to see that his vote is among the set of votes, removes all the random strings at that level, scrambles all the votes, and then sends the result to Carol.

Each vote now looks like this:

$$E_C\{R_3,E_D\{R_2,E_A\{E_B\{E_C\{E_D\{V,R_1\}\}\}\}\}\}$$

(6) Carol decrypts all of the votes with her private key, checks to see that her vote is among the set of votes, removes all the random strings at that level, scrambles all the votes, and then sends the result to Dave.

Each vote now looks like this:

$$E_D\{R_2,E_A\{E_B\{E_C\{E_D\{V,R_1\}\}\}\}\}$$

(7) Dave decrypts all of the votes with his private key, checks to see that his vote is among the set of votes, removes all the random strings at that level, scrambles all the votes, and sends them to Alice.

Each vote now looks like this:

$$E_A\{E_B\{E_C\{E_D\{V,R_1\}\}\}\}$$

(8) Alice decrypts all the votes with her private key, checks to see that her vote is among the set of votes, signs all the votes, and then sends the result to Bob, Carol, and Dave.

Each vote now looks like this:

$$S_A\{E_B\{E_C\{E_D\{V,R_1\}\}\}\}$$

(9) Bob verifies and deletes Alice's signatures. He decrypts all the votes with his private key, checks to see that his vote is among the set of votes, signs all the votes, and then sends the result to Alice, Carol, and Dave.

Each vote now looks like this:

$$S_B\{E_C\{E_D\{V,R_1\}\}\}$$

purpose is to find out and publicize the total number of actual voters. Although some of them probably will not participate, it reduces the ability of the CTF to add fraudulent votes.

In step (4), it is possible for two voters to get the same identification number. This possibility can be minimized by having far more possible identification numbers than actual voters. If two voters submit votes with the same identification tag, the CTF generates a new identification number, I', chooses one of the two votes, and publishes:

$$I', E_k(I, v)$$

The owner of that vote recognizes it and sends in a second vote, by repeating step (5), with the new identification number.

Step (6) gives each voter the capability to check that the CTF received his vote accurately. If his vote is miscounted, he can prove his case in step (9). Assuming a voter's vote is correct in step (6), the message he sends in step (9) constitutes a proof that his vote is miscounted.

One problem with the protocol is that a corrupt CTF could allocate the votes of people who respond in step (2) but who do not actually vote. Another problem is the complexity of the ANDOS protocol. The authors recommend dividing a large population of voters into smaller populations, such as election districts.

Another, more serious problem is that the CTF can neglect to count a vote. This problem cannot be resolved: Alice claims that the CTF intentionally neglected to count her vote, but the CTF claims that the voter never voted.

Voting without a Central Tabulating Facility

The following protocol does away with the CTF entirely; the voters watch each other. Designed by Michael Merritt [452,1076,453], it is so unwieldy that it cannot be implemented practically for more than a handful of people, but it is useful to learn from nevertheless.

Alice, Bob, Carol, and Dave are voting yes or no (0 or 1) on a particular issue. Assume each voter has a public and private key. Also assume that everyone knows everyone else's public keys.

(1) Each voter chooses his vote and does the following:
   (a) He attaches a random string to his vote.
   (b) He encrypts the result of step (a) with Dave's public key.
   (c) He encrypts the result of step (b) with Carol's public key.
   (d) He encrypts the result of step (c) with Bob's public key.
   (e) He encrypts the result of step (d) with Alice's public key.
   (f) He attaches a new random string to the result of step (e) and encrypts it with Dave's public key. He records the value of the random string.
   (g) He attaches a new random string to the result of step (f) and encrypts it with Carol's public key. He records the value of the random string.

(10) Carol verifies and deletes Bob's signatures. She decrypts all the votes with her private key, checks to see that her vote is among the set of votes, signs all the votes, and then sends the result to Alice, Bob, and Dave.

Each vote now looks like this:

$$S_C(E_D(V,R_1))$$

(11) Dave verifies and deletes Carol's signatures. He decrypts all the votes with his private key, checks to see that his vote is among the set of votes, signs all the votes, and then sends the result to Alice, Bob, and Carol.

Each vote now looks like this:

$$S_D(V,R_1)$$

(12) All verify and delete Dave's signature. They check to make sure that their vote is among the set of votes (by looking for their random string among the votes).

(13) Everyone removes the random strings from each vote and tallies the votes.

Not only does this protocol work, it is also self-adjudicating. Alice, Bob, Carol, and Dave will immediately know if someone tries to cheat. No CTF or CLA is required. To see how this works, let's try to cheat.

If someone tries to stuff the ballot, Alice will detect the attempt in step (3) when she receives more votes than people. If Alice tries to stuff the ballot, Bob will notice in step (4).

More devious is to substitute one vote for another. Since the votes are encrypted with various public keys, anyone can create as many valid votes as needed. The decryption protocol has two rounds: round one consists of steps (3) through (7), and round two consists of steps (8) through (11). Vote substitution is detected differently in the different rounds.

If someone substitutes one vote for another in round two, his actions are discovered immediately. At every step the votes are signed and sent to all the voters. If one (or more) of the voters noticed that his vote is no longer in the set of votes, he immediately stops the protocol. Because the votes are signed at every step, and because everyone can backtrack through the second round of the protocol, it is easy to detect who substituted the votes.

Substituting one vote for another during round one of the protocol is more subtle. Alice can't do it in step (3), because Bob, Carol, or Dave will detect it in step (5), (6), or (7). Bob could try in step (5). If he replaces Carol's or Dave's vote (remember, he doesn't know which vote corresponds to which voter), Carol or Dave will notice in step (6) or (7). They wouldn't know who tampered with their vote (although it would have had to be someone who had already handled the votes), but they would know that their vote was tampered with. If Bob is lucky and picks Alice's vote to replace, she won't notice until the second round. Then, she will notice her vote missing in step (8). Still, she would not know who tampered with her vote. In the first round, the votes are shuffled from one step to the other and unsigned; it is impossible for anyone to backtrack through the protocol to determine who tampered with the votes.

Another form of cheating is to try to figure out who voted for whom. Because of the scrambling in the first round, it is impossible for someone to backtrack through the protocol and link votes with voters. The removal of the random strings during the first round is also crucial to preserving anonymity. If they are not removed, the scrambling of the votes could be reversed by re-encrypting the emerging votes with the scrambler's public key. As the protocol stands, the confidentiality of the votes is secure.

Even more strongly, because of the initial random string, R_1, even identical votes are encrypted differently at every step of the protocol. No one knows the outcome of the vote until step (11).

What are the problems with this protocol? First, the protocol has an enormous amount of computation. The example described had only four voters and *it* was complicated. This would never work in a real election, with tens of thousands of voters. Second, Dave learns the results of the election before anyone else does. While he still can't affect the outcome, this gives him some power that the others do not have. On the other hand, this is also true with centralized voting schemes.

The third problem is that Alice can copy anyone else's vote, even though she does not know what it is beforehand. To see why this could be a problem, consider a three-person election between Alice, Bob, and Eve. Eve doesn't care about the result of the election, but she wants to know how Alice voted. So she copies Alice's vote, and the result of the election is guaranteed to be equal to Alice's vote.

Other Voting Schemes

Many complex secure election protocols have been proposed. They come in two basic flavors. There are mixing protocols, like "Voting without a Central Tabulating Facility," where everyone's vote gets mixed up so that no one can associate a vote with a voter.

There are also divided protocols, where individual votes are divided up among different tabulating facilities such that no single one of them can cheat the voters [360,359,118,115]. These protocols only protect the privacy of voters to the extent that different "parts" of the government (or whoever is administering the voting) do not conspire against the voter. (This idea of breaking a central authority into different parts, who are only trusted when together, comes from [316].)

One divided protocol is [1371]. The basic idea is that each voter breaks his vote into several shares. For example, if the vote were "yes" or "no," a 1 could indicate "yes" and a 0 could indicate "no"; the voter would then generate several numbers whose sum was either 0 or 1. These shares are sent to tabulating facilities, one to each, and are also encrypted and posted. Each center tallies the shares it receives (there are protocols to verify that the tally is correct) and the final vote is the sum of all the tallies. There are also protocols to ensure that each voter's shares add up to 0 or 1.

Another protocol, by David Chaum [322], ensures that voters who attempt to disrupt the election can be traced. However, the election must then be restarted without the interfering voter; this approach is not practical for large-scale elections.

Another, more complex, voting protocol that solves some of these problems can be found in [770,771]. There is even a voting protocol that uses multiple-key ciphers

[219]. Yet another voting protocol, which claims to be practical for large-scale elections, is in [585]. And [347] allows voters to abstain.

Voting protocols work, but they make it easier to buy and sell votes. The incentives become considerably stronger as the buyer can be sure that the seller votes as promised. Some protocols are designed to be **receipt-free**, so that it is impossible for a voter to prove to someone else that he voted in a certain way [117,1170,1372].

6.2 SECURE MULTIPARTY COMPUTATION

Secure multiparty computation is a protocol in which a group of people can get together and compute any function of many variables in a special way. Each participant in the group provides one or more variables. The result of the function is known to everyone in the group, but no one learns anything about the inputs of any other members other than what is obvious from the output of the function. Here are some examples:

Protocol #1

How can a group of people calculate their average salary without anyone learning the salary of anyone else?

(1) Alice adds a secret random number to her salary, encrypts the result with Bob's public key, and sends it to Bob.

(2) Bob decrypts Alice's result with his private key. He adds his salary to what he received from Alice, encrypts the result with Carol's public key, and sends it to Carol.

(3) Carol decrypts Bob's result with her private key. She adds her salary to what she received from Bob, encrypts the result with Dave's public key, and sends it to Dave.

(4) Dave decrypts Carol's result with his private key. He adds his salary to what he received from Carol, encrypts the result with Alice's public key, and sends it to Alice.

(5) Alice decrypts Dave's result with her private key. She subtracts the random number from step (1) to recover the sum of everyone's salaries.

(6) Alice divides the result by the number of people (four, in this case) and announces the result.

This protocol assumes that everyone is honest; they may be curious, but they follow the protocol. If any participant lies about his salary, the average will be wrong. A more serious problem is that Alice can misrepresent the result to everyone. She can subtract any number she likes in step (5), and no one would be the wiser. Alice could be prevented from doing this by requiring her to commit to her random number using any of the bit-commitment schemes from Section 4.9, but when she revealed her random number at the end of the protocol Bob could learn her salary.