Electronic Voting

Mechanical voting schemes have been receiving a lot of press lately, none of it good. But for now people are also wary of electronic schemes because of their perception of computer systems as playgrounds for hackers - computers might be more accurate but they are also more susceptible to tampering. At least you need to be handy with a screwdriver to mess with a machine. It sometimes feels like any kid with a computer can mess with government information systems.

Its important to understand how we can protect voter information in electronic schemes. Electronic voting is also interesting because it is an example of a *multi-party protocol*. Rather than simply hiding or revealing data, we can put the data to work without revealing it.

We have several goals in digital voting schemes. First of all, we want them to be accurate: the final count should be the true count of voters' votes. The surest way to do this is to make the outcome verifiable. Anyone can check the total and verify that its correct. But we also want to protect voters' privacy, especially if many parties are allowed to verify the total. So ideally we want a method for checking the total of some votes without disclosing individual votes. Finally, keeping voters' votes private may allow them to cheat: the methods we use are numerical, and if a voter can vote zero or one, they may be able to cheat by sending in huge numbers as their vote. So we would like user votes to be verifiable. There are several voting schemes that satisfy these criteria. Here is a practical one.

Homomorphism

The first idea we need is homomorphism: a homomorphism is a function h together with group operations defined on its domain and range which are compatible with the function, that is:

$$h(a \oplus b) = h(a) \otimes h(b)$$

The operations $\oplus$ and $\otimes$ are generally different because the groups in the domain and range of h are different. Suppose for instance that $h(x) = g^x (\bmod\ p)$ for some prime p. Then $h(a + b) = h(a) * h(b)$, so that $\oplus = +$ and $\otimes = \times$. We will need this homomorphism later.

Secret sharing is another homomorphism. Let the shares of a secret a be $a_1, \ldots, a_n$ such that $t + 1$ of them are enough to reconstruct a. Recall that we can compute the secret from the shares using a linear combination, that is

$$a = \sum_{i=1}^{t+1} a_i L_i(0)$$

The coefficient $L_i(0)$ depends on which secret shares we are using, but not on the shares themselves. So if we have another secret b that is split into shares $b_1, \ldots, b_n$, its reconstruction is

$$b = \sum_{i=1}^{t+1} b_i L_i(0)$$

It follows easily that if we define $h(s_1, \ldots, s_{t+1}) = \sum s_i L_i(0)$, then h is a homomorphism:

$$h(a_1 + b_1, \ldots, a_{t+1} + b_{t+1}) = h(a) + h(b)$$

In this case, $\oplus$ is vector addition, and $\otimes$ is addition of scalars.

Question: Why couldnt we define the homomorphism the other way: $h(a) \rightarrow (a_1, \ldots, a_n)$?

Distributing Votes

Using the secret-sharing homomorphism, we can develop a voting scheme that is verifiable and hides individual votes quite easily. Suppose there are N voters and n "voting centers" that collect votes. We assume that these centers are independent of each other, and reasonably trustworthy (some might be operated by major political parties).

Let v_i be person i's vote, and lets say that its required to be 0 or $+1$. Person i splits their vote into n pieces using secret sharing:

$$v_i \longrightarrow (v_{i1}, \ldots, v_{in})$$

and they send the j^{th} piece to voting center j. To avoid a spy on the network seeing all the shares, we assume that each center has a public encryption key E_i and that every message to a center is encrypted with this key. After every person has voted, the n centers each have a share of every vote.

Aside: There is a non-trivial distributed computing problem here. The voters need to send their votes to the centers independently or they lose privacy. Its very likely that some votes will not make it to all the centers. But the centers must use exactly the same set of votes for the reconstruction step to work. Agreeing on which votes to use is a byzantine agreement problem. If we only used votes received by all centers, it would be very easy for one dishonest center to discard votes. Fortunately, if $t + 1$ or more centers receive a vote, its possible to reconstruct the other shares without exposing the vote. So the centers can agree to count all votes with $t + 1$ or more pieces, and that prevents small coalitions of centers from cheating.

now each center computes the total of all of its votes, and

$$T_j = \sum_{i=1}^{N} v_{ij}$$

is the total computed by center j. The homomorphism property implies that

$$T = \sum_{j=1}^{t+1} T_j L_j(0)$$

is the correct total of the votes. We can compute such a total for various subsets of $t+1$ centers. All the totals should agree. If one center cheats, we will get different totals. It will be obvious that there is a problem, and it will not be difficult to find the offending center. Assuming they all agree, we have a reliable total T of all the votes.

The weakness with this scheme is that the arithmetic calculations are done $\bmod\ p$ for some large prime (required for secret sharing), and any number in $\mathbb{Z}_p$ could be used as a vote. Any voter could give 10^6 votes for one candidate. Because their vote is hidden by secret sharing, no center could detect this fraud.

Verifiable Votes

The solution is to require each voter to make a certain bit commitment to their vote, which they make public. The commitment keeps the vote secret, but allows the centers and other parties to verify that the voter has voted honestly.

Suppose p is of the form $kq+1$ for some large prime q. Let g and h be elements of the subgroup of $\mathbb{Z}_p$ of order q. To commit to a vote v, we choose a random r in $\{1, \ldots, q\}$ and define

$$B_r(v) = g^r h^v (\bmod\ p)$$

Assuming that the discrete log problem is hard, a person that knows $B_r(v)$ and g and h, cannot determine r or v. Notice that this bit commitment scheme is also a homomorphism:

$$B_{r_1+r_2}(v_1+v_2) = B_{r_1}(v_1) B_{r_2}(v_2)$$

We need r to hide the vote because v is supposed to be a very small number. It is not hard to compute a discrete log when the log is zero or one.

This time, each voter uses secret sharing to split *both* their vote v_i and the secret number r_i into n pieces. They send the j^{th} shares of both pieces to center j. Center j computes its tally T_j as before, but also computes a tally of r values:

$$R_j = \sum_{i=1}^{N} r_{ij}$$

These R_j's can be combined like the T_j's to reconstruct R which is the total of all the user's r_i values. Then because of the homomorphism of $B()$, we can check the total:

$$\prod_{i=1}^{N} B_{r_i}(v_i) = g^R h^T$$