Electronic Voting

We finish our discussion of electronic voting with a scheme to verify that each voter is voting honestly. Since the protocol we described last time allows votes to be cast secretly, a voter can exceed the specfied range of values (usually $\{0, 1\}$) without fear of discovery. The last piece of the protocol is a scheme that establishes each voters honesty:

ZKP of Voter Honesty

Recall that each voter has a vote v, and a random secret r that hides their vote in a commitment:

$$B_r(v) = g^r h^v (\text{mod } p)$$

. The goal of this ZKP is to prove that in $B_r(v)$, v is either zero or one. This proof is a particular example of a general style of ZK proof that one of k claims is true. We actually construct ZKP proofs that $v = 0$ and $v = 1$, but we cheat on one proof. The verification step is arranged so that the verifier is convinced we have only cheated once.

Both proofs are three-step interactive proofs with a random challenge from the verifier. Here's a proof that the vote is zero. We assume that p, q, g and h are known to everyone:

1. Prover (voter) chooses random $u \in \mathbb{Z}_q$ and sends $\alpha = g^u (\text{mod } p)$ to verifier.
2. Verifier sends a random challenge c to prover.
3. Prover sends back $w = rc + u (\text{mod } q)$ to verifier.
4. Verifier checks that $g^w = \alpha (B_r(v))^c (\text{mod } p)$

This is actually the zero-knowledge proof of discrete log we gave earlier. The prover is proving knowledge of the discrete log of g^r without revealing r, since when v is zero, $B_r(v)$ is just g^r. To adapt this prove to prove that $v = 1$, we need only change the last step so that we are again proving knowledge of the discrete log of g^r:

3. Verifier checks that $g^w = \alpha (B_r(v) h^{-1})^c (\text{mod } p)$

Because this is a zero-knowledge proof, we know that the series of messages could be simulated by a third party. Here is a simulation of the proof that $v = 0$:

1. Choose w, c at random first
2. Compute $\alpha = g^w (B_r(v))^{-c} (\text{mod } p)$.

These values clearly satisfy the verifier's check in step 3. But we dont need to know r to create this simulation. Obviously it gives us no information about r. This simulation can work for $v = 1$ by changing $\alpha = g^w(B_r(v)h^{-1})^{-c} \pmod{p}$.

The difference between the real proof and the simulation is that the challenge from the verifier in the real proof comes after the prover has sent α. If we run both protocols (for $v = 0$ and $v = 1$) in parallel, the prover could not cheat. But if we run both in parallel and the verifier issues a *single challenge* at step 2., which must be the sum of the two challenges, then the prover can cheat at most once.

So to make a proof that $v = 0$ or $v = 1$, we *either* combine a true proof that $v = 0$ with a simulation that $v = 1$, or we combine a true proof that $v = 1$ with a simulation that $v = 0$. The verifier can't tell which is true. They know that one or the other must be true, because the proof shows that we didnt cheat twice. But they can't tell which. Here is the protocol for the case that $v = 0$ (in real life):

1. Prover (voter) chooses random $u_0 \in \mathbb{Z}_q$ and sends $\alpha_0 = g^{u_0} \pmod{p}$ to verifier. Prover runs simulation for $v = 1$, and chooses random w_1, c_1 and sends $\alpha_1 = g^{w_1}(B_r(v)h^{-1})^{-c_1} \pmod{p}$ to verifier.
2. Verifier sends a random challenge c to prover.
3. Prover computes $c_0 = c - c_1$. For proof that $v = 0$, prover sends back $w_0 = rc_0 + u_0 \pmod{q}$ and c_0 to verifier. For proof (simulation) that $v = 1$, prover sends w_1 and c_1 to verifier.
4. Verifier checks that $c = c_0 + c_1$ and that
$$g^{w_0} = \alpha_0(B_r(v))^{c_0} \pmod{p}$$
$$g^{w_1} = \alpha_1(B_r(v)h^{-1})^{c_1} \pmod{p}$$

The verifier knows that prover couldnt have chosen both c_0 and c_1 in advance, and therefore must have computed one of them after step 1. Thus one of the proofs must be valid. If $v = 1$, the prover use a real proof that $v = 1$ and a simulation that $v = 0$, which looks like this:

1. Prover (voter) chooses random $u_1 \in \mathbb{Z}_q$ and sends $\alpha_1 = g^{u_1} \pmod{p}$ to verifier. Prover runs simulation for $v = 0$, and chooses random w_0, c_0 and sends $\alpha_0 = g^{w_0}(B_r(v))^{-c_0} \pmod{p}$ to verifier.
2. Verifier sends a random challenge c to prover.
3. Prover computes $c_1 = c - c_0$. For proof that $v = 1$, prover sends back $w_1 = rc_1 + u_1 \pmod{q}$ and c_1 to verifier. For proof (simulation) that $v = 0$, prover sends w_0 and c_0 to verifier.
4. Verifier checks that $c = c_0 + c_1$ and that
$$g^{w_0} = \alpha_0(B_r(v))^{c_0} \pmod{p}$$
$$g^{w_1} = \alpha_1(B_r(v)h^{-1})^{c_1} \pmod{p}$$

Question: How could we make this protocol non-interactive?

Now whenever a voter votes, one or more centers challenge them to show that their vote is really zero or one using this protocol. The checksum of commitments verifies that all the voters secret shares match their bit commitments. With a little more work, we can design protocols such that voters are required to prove that each individual share of their vote matches the commitment. Thus each center can check that an individual's vote is valid as soon as they receive it.

The protocol is reasonably efficient. It requires $O(nN)$ messages, and at most $O(N^2)$ work per voter to do the secret sharing. It still has some weaknesses, mainly the need to synchronize all the centers lists of votes to be tallied. But it shows that some very robust e-voting schemes are possible.