Secure Voting with Elgamal Encryption

1 Elgamal Encryption

Elgamal encryption is a type of public key cryptography. It is useful because we can use it to do additive homomorphic encryption, meaning that multiple parties can use a public key to encrypt their secrets, and can then decrypt together resulting in the the sum of their secrets. This allows people to compute certain together functions without revealing their secrets to anyone else.

1.1 Walkthrough

We start with three system parameters (in our case, handed out by the server to all voters): p, q, g, and a list of generators. p is a large prime with some special properties, most notably that is equal to $2q + 1$ for some other prime q. g is any generator of $\mathbb{Z}_p^*$. In most cases, $g = 2$. The generators list is simply a list of linearly independent generators also for $\mathbb{Z}_p^*$. Let $n \in_R \mathbb{Z}_m$ denote choosing a random value n from $\mathbb{Z}_m$ for some positive integer m. Let there be n voters each with a unique ID i ranging from 1 to n. Let $G[i]$ denote the ith generator in the given list of generators. Let there be k candidates.

1. Each voter i picks $s_i \in_R \mathbb{Z}_m$.
2. Each voter shares the value $h_i = g^{s_i} \pmod{p}$ to every other voter.
3. Each voter computes $h = \prod_{j=1}^{n} h_j \pmod{p}$. h is the public key. Note that $h = g^s \pmod{p}$ for $s = \Sigma s_i$ but no voter knows the secret of any other voter.
4. The next step is for each voter to encrypt their vote using the public key. If they are voting for candidate c, then their vote becomes $G[c]$.

Their encrypted vote is a pair (x_i, y_i). To calculate these, they first pick some $\alpha \in_R \mathbb{Z}_q$ and then do:

$$x_i = g^\alpha \pmod p$$

$$y_i = h^\alpha \cdot G[v_i] \pmod p$$

Where v_i is the index of the candidate they have chosen.

5. Each voter shares their encrypted vote (x_i, y_i) with everyone else.
6. Each voter then calculates $x = \prod x_i \pmod p$ and $y = \prod y_i \pmod p$. They will then decrypt this final pair (which can be thought of an encryption of the result of the vote).
7. Each voter finds $w_i = x^{s_i} \pmod p$ and shares this value with everyone else.
8. Each voter computes $w = \prod w_i \pmod p$. Finally, they do $y/w \pmod p$. This final is the result of the vote.

Note that the final value everyone computes is equal to:

$$\frac{y}{w} = \frac{(h^{\alpha_1} G[v_1]) \cdot (h^{\alpha_2} G[v_2]) \cdot \ldots \cdot (h^{\alpha_1} G[v_n])}{x^{s_n} \cdot x^{s_2} \cdot \ldots \cdot x^{s_n}} = \frac{h^{\Sigma \alpha_i}(vote)}{x^s} \pmod p$$

$$= \frac{(g^s)^{\Sigma\alpha}(vote)}{(g^{\Sigma\alpha})^s} = (vote) \pmod p$$

Where $(vote) = G[1]^{l_1} \cdot G[2]^{l_2} \cdot \ldots \cdot G[k]^{l_k}$ where l_i is the number of votes for candidate i. Once everyone has agreed on the final value, we can than compute each of l_i's and get the result.

2 Zero Knowledge Proofs and Pedersen Commits

Though the Elgamal encryption scheme on its own guarantees voters cannot determine others' inputs, it does nothing to prevent cheating. To combat this, we simply require that players prove the validity of their votes using two zero knowledge proofs (ZKPs). The first is to prove that a player i's vote, (x_i, y_i), is the encryption of just one generator from the list of generators. This boils down to a proof of

$$\log_g x = \log_h(y/G[1]) \ \vee \ \log_g x = \log_h(y/G[2]) \ \vee \ \ldots \ \vee \ \log_g x = \log_h(y/G[k])$$

which can be accomplished with a ZKP.

The second one used during decryption to prove that a voters share $h_i = g^{s_i}$ of the public key is consistent with their decryption share of $w_i = x^{s_i}$. They can prove this using a ZKP which proves:

$$\log_g h_i = \log_x w_i$$

Depending on the voting algorithm, there might also be some other ZKPs, but these two are enough to do a normal majority win vote with any number of voters and candidates.