E-Voting System Using Paillier Encryption

Team 19: Kareem El-Faramawi, Josh Ferrell, Young Liu

Deployment Instructions:

See ReadMe provided in the source code. It provide more details than the steps below.

Install: PyCrypto, pygubu with pip or equivalent
Specify candidates in candidates.txt: one line per candidate
Start: Run in this order: Registrar, Board, Voter/VoterClient
Retain a Board terminal
To move through phases of an election (Registration, Voting, Counting) press enter in the Board terminal

Final cleanup: If any windows get stuck: killall python2

Architecture:

Our system is composed of three components, a registrar of voters, a bulletin board, and voters. The registrar handles registration and stores registered voters in a database. It also handles the signing of messages providing a means of verifying that a vote is from a registered voter. The voters are required to register to vote. After registering they login to the voting system and interact with the bulletin board to submit their votes. The voters will make a selection via a GUI and send it to the bulletin board for verification via zero knowledge proof challenges. If the vote is accepted, the voter's vote will be recorded in an in-memory dictionary of the bulletin board. Once the election is over, the counting authority, which in our implementation is merged into the board, will report the total tally for each candidate.

Each of the components interacts with the other components via sockets. For this project, the system will run on the same host with each component running as a separate process or processes in the case of the voter.

From a real world perspective, the registrar and bulletin board would be hosted remotely with any attacker only having the access to the voter and exposed APIs of the registrar and bulletin board.

Voting Scheme:

Our system supports the choosing of 1 of N candidates. This vote is then encoded in such a way to leverage the Paillier cryptosystem to allow the votes to be summed. In particular, we take the number of registered voters and use that to determine the number of bits to allocate to a candidate. Possible votes are then numbers with 1 at different bit offsets as determined by the previous step. This allows us to sum all the votes in one step without fear of one candidate's votes interfering with another's. Every user's vote is stored as one entry in the dictionary of votes.

Security Considerations:

For votes we employ zero knowledge proofs to verify that a user's votes have not been tampered with and that a user's vote falls within a valid set of votes.

The bulletin board enforces that a user can't vote twice as it will reject any votes from a user that has already voted.

The registrar is used to ensure that votes are from registered voters via blind signatures which the bulletin board verifies before accepting a vote.

User's authenticate themselves with a password that they create upon registration.

Communication between board, registrar, and voters is encrypted via AES with RSA key sharing. We assume in real life we would have access to a certificate authority which would allow us to enhance the security of this communication.

We also assume that if we want to replicate more restrictive voter registration we could easily specify more requirements and verify more personal information with additional checks against other databases such as those belonging to the DMV.

We could also enhance the bulletin board to store use votes in a more persistent form that would allow future verification of the election results.

Work Cited

Baudron, O., Fouque, P. A., Pointcheval, D., Stern, J., & Poupard, G. (2001, August). Practical multi-candidate election system. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing* (pp. 274-283). ACM.

Libraries:

PyCrypto - https://github.com/dlitz/pycrypto
Pygubu - https://github.com/alejandroautalan/pygubu
TkInter - http://tkinter.unpythonic.net/wiki/