Secure postal voting

Henri Devillez

UCLouvain, Crypto Group, Belgium

Abstract. There has been several recent attempts to enhance postal voting systems with the technologies of end-to-end voting systems to obtain the best of both worlds. Our contribution is two fold. We first propose a postal voting protocol that uses ballots interpretable by the voters, and then we give a security model in a simpler variant of the universally composable model (SUC) for which our protocol is provably secure.

1 Motivation and limitations

At the present time, the most common form of remote voting used in practice is postal voting. Despite its extreme simplicity, this naive system guarantees the remote elector that their vote is cast as intended through an human-readable format of the ballot and does not give any evidence of their choice after the ballot has been posted. However, the voter has no way to verify that their ballot has reached an election office and has been correctly counted in the tally. On the other end, there has been a lot of efforts to design verifiable electronic election systems with the use of cryptography.

Recently, there have been attempts to bring together these two approaches and keep the best of both worlds [1]. In this draft, we provide a protocol achieving these goals and an intuitive functionality in the universally composable model [2] capturing the desired privacy and verifiablity properties of vote-by-mail, for which our protocol is provably secure. We restrict ourselves to a strict setting in which ballots are downloaded and printed by the voter (hence preventing the use of special types of papers) and ballots cast by the voters are human-readable.

2 High level view of the protocol

We consider a protocol for remote elections in a vote-by-mail setting with approval voting ballots. By approval voting, we mean that given a list of candidates, a voter can choose to approve any subset of these candidates.

The election is overseen by a party called the election authorities (EA). This party sets the parameters of the election p, which consists in the list of valid voters, the list of candidates for which one can vote, the numbers of talliers.

Ballots are generated by an independent server, the ballots issuer (BI). When a voter asks the server for a ballot, the voter authenticates to the server (with an electronic ID for example) and receives a blank ballot. This blank ballot contains

a sheet with an unpredictable voting token and a list of candidate that the voter has to fill. The voter will then send this ballot to the election office (EO). The ballot also contains a sheet of codes and a note sheet. For each choice, the voter copies the code corresponding to their choices on the note sheet, then destroys the codes sheet. All the codes are also encrypted by the ballots issuer using a public key of a threshold encryption scheme run by the talliers. The encryptions of each of those codes are sent to the talliers, each time with the corresponding choice and an hash of the voting token associated to the voter and the choice.

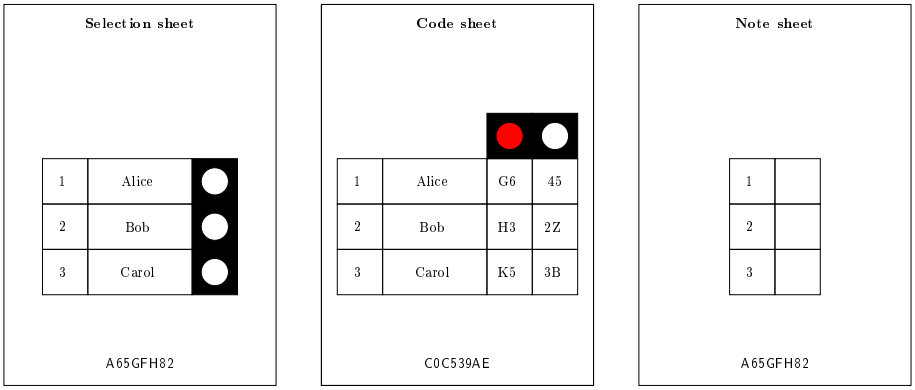

Fig. 1. example of ballot

During the tallying phase, the election office interacts with the talliers to compute the result of the election that can be published once every ballot has been counted. To do so, the election office sends to each tallier a hash of the voting token and the choices written on the ballot. The talliers recovers the corresponding choices and encryptions received from the ballot issuer. Hence each tallier can independently compute the tally of the election and if they all agree on the result, they decrypt together the codes matching each selected choice. These codes are then sent to a verification server (VS). Voters can later connect to this server to receive the decrypted codes and compare them with the codes they saved when they voted. As long as it is difficult to guess the code of the other choices, the voter has a guarantee that their choices have been correctly recorded and counted if they see the right codes when they interact with the verification server.

3 Security

We introduce a simulation-based security definition in a simpler variant of the universally composable model (SUC)[2] for the security of postal voting protocols. As usual in this framework, we define a trusted third party which would give natural security guarantees if it happened to exist. A protocol is secure if for

any adversary $\mathcal{A}$ against the protocol, there exists an adversary $\mathcal{S}$ (also called the simulator) such that a real execution of the protocol with the adversary $\mathcal{A}$ is indistinguishable from an ideal execution of the protocol with the trusted third party with $\mathcal{S}$.

In the ideal world, we define the ideal postal voting functionality as follows. Voters send their votes to this functionality instead of the election office and receive from it the output of the verification procedure at the end of the election.

Secure Postal Voting functionality

Setup Phase

When receiving a command $\mathsf{setup}(\mathsf{p})$ from EA, forwards the election parameters p to every voter and the simulator. In particular, p contains a probability $\mathsf{p\_limit}$ which is an upper bound on the probability of success of adversary changing the vote of a voter without him noticing anything. Then, assign to each voter a random distinct string handle. For each handle, keep in memory $\mathsf{ballots}^{\mathsf{cast}}_{\mathsf{handle}} = \{\}$ and $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}} = \{\}$. Also, send to the simulator the mapping between the corrupted voters and their handle.

If the simulator replies with a command abort, send abort to EA. Otherwise if the simulator replies with a command $\mathsf{continue}$, enter the Voting phase.

Voting phase

- When receiving a $\mathsf{vote}(\mathsf{id}, \mathsf{v})$ command from the voter with identity id, append v to $\mathsf{ballots}^{\mathsf{cast}}_{\mathsf{handle}}$ and $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}$ where handle is linked to id.
- When receiving a $\mathsf{vote}(\mathsf{id}, \mathsf{v})$ command from the simulator, append v to $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}$ where handle is associated to id.
- If the postal channel is corrupted, the functionality has two additional commands. When receiving a command $\mathsf{get\_vote}(\mathsf{id}, \mathsf{i})$ from the simulator, send to the simulator the i-th vote v in the list $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}$. When receiving a command $\mathsf{modify\_vote}(\mathsf{id}, \mathsf{i}, \mathsf{new_v})$, modify the i-th vote in the list $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}$ into $\mathsf{new\_v}$. If $\mathsf{new\_v} = \perp$, drop the ballot instead.
- When receiving a $\mathsf{stop\_vote}$, enter in the Tallying phase.

Tallying phase

- When entering the $\mathsf{tallying}$ phase, send to the simulator a list of pairs of handle and the corresponding $\mathsf{vote}^{\mathsf{counted}}_{\mathsf{handle}}$.
- When receiving a command $\mathsf{modify\_ballot}(\mathsf{handle}, \mathsf{i}, \mathsf{new\_v})$ from the simulator, update the i-th vote of $\mathsf{vote}^{\mathsf{counted}}_{\mathsf{handle}}$ into $\mathsf{new\_v}$. If $\mathsf{new\_v} = \perp$, drop the ballot instead.
- When receiving a command abort from the simulator, send a message abort to EA.
- When receiving a command $\mathsf{publish}$ from the simulator, send to the dummy EA and to every voter the result of the election. The result r of the election is computed as follows:
  1. Set $r = \{\}$

2. For each handle, append $\perp$ to r if $|\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}| \neq 1$. Otherwise append v to r where v is the unique vote of $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}$.

Then enter in the Verification phase.

Verification phase

For every voter, send a command verify(id) to the simulator. The simulator sends one of the following command in return:

- When receiving a command verification_fail from the simulator, send cheat to the voter with identity id.
- When receiving a command verification_success(p_success) from the simulator, ignore the command if p_success $>$ p_limit. Otherwise, send honest to the voter with identity id with probability p_success or cheat with probability $1 -$ p_success
- When receiving a command ballot_based from the simulator, send to the voter with identity id either:
  - cheat if the votes computed with $\mathsf{ballots}^{\mathsf{cast}}_{\mathsf{handle}}$ and $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}$ are not the same
  - nothing_received if $\mathsf{ballots}_{\mathsf{cast}} = \mathsf{ballots}_{\mathsf{counted}} = \{\}$
  - honest otherwise

Intuitively, every voter has an handle hiding their identity. The functionality will later know the vote of each handle but the mapping between the handles and the voters' id will remain secret, hence preserving the privacy (except for the corrupted voters or the leak caused by the postal channel corruption).

Regarding the individual verifiability of the election, the variables $\mathsf{ballots}^{\mathsf{cast}}_{\mathsf{handle}}$ and $\mathsf{ballots}^{\mathsf{counted}}_{\mathsf{handle}}$ respectively represent the voter's ballots that are cast by the voter and counted in the tallying procedure. These might be different because of the adversarial behavior, but in that case the voter will receive a cheat message with a probability at least p_limit.

Given this functionality, we can prove that the protocol sketched in Section 2 satisfies the following property:

Definition 1 (Secure postal voting). *A PVP $\mathcal{V}$ is secure if for any adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{S}$ such that for any environment $\mathcal{E}$, the probability that the environment distinguishes between the execution of the real protocol and an interaction with the ideal postal voting functionality is negligible. In this game, at most one of these sets of official parties is corrupted:* {EO *and all but one tallier*}, {BI} *and* {VS}. *Any number of voters and the postal channel can be corrupted.*

References

1. Benaloh, J.: Strobe-voting: Send two, receive one ballot encoding. In: International Joint Conference on Electronic Voting. pp. 33–46. Springer (2021)
2. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: Annual Cryptology Conference. pp. 3–22. Springer (2015)