PrivApollo– Secret Ballot E2E-V Internet Voting

Hua Wu[1], Poorvi L. Vora[1], and Filip Zagórski[2]

[1] Department of Computer Science*,
The George Washington University
[2] Department of Computer Science**,
Wroclaw University of Science and Technology

Abstract. The Apollo voting protocol improves on the integrity properties of Helios by enabling voters to communicate to the public the failure of the cast-as-intended check, in the event that the voting terminal changes the vote on receiving the credential. It also enables the voter to detect a dishonest registrar and to prove misbehaviour. It provides an explicit description of the role of one or more computational *voting assistants* which help the voter perform the checks without obtaining information on the vote. Unfortunately, neither Helios nor Apollo provides ballot secrecy, because the voting terminal knows the vote. We present PrivApollo, a protocol that improves Apollo by providing ballot secrecy from the voting terminal.

1 Introduction

Since the first cryptographic protocol for secure voting, the area has grown considerably and has led to a number of protocols that have been used in real elections. We focus on Internet voting approaches—such as Helios [1] and Apollo [7]—whose privacy properties are conditional on the security of the cryptographic techniques used. In proposals of this kind, ballot secrecy is typically protected through the encryption of the vote by the voting terminal. A malicious entity on the terminal could leak the vote, which could lead to ballot selling, coercion or selective denial of service. We propose a voting system, PrivApollo, which ensures that the vote is private from the voting terminal.

PrivApollo is an extension of Apollo. In Apollo, the voter relies on a number of voting assistants—computational devices whose role is to assist the voter in checking the actions of the voting system and the voting terminal. The voting assistants do not obtain any information on the vote, while the voting terminal knows the vote. PrivApollo also relies on voting assistants. Additionally, in

* This material is based upon work supported in part by the Maryland Procurement Office under contract H98230-14-C-0127 and NSF Award CNS 1421373

** Author was partially supported by Polish National Science Centre contract number DEC-2013/09/D/ST6/03927 and by Wroclaw University of Science and Technology [0401/0052/18]

PrivApollo, one specific voting assistant is denoted the *active voting assistant*, and enables the stronger privacy properties of PrivApollo. It is expected that software for the voting assistants will be written by citizen interest groups, in much the same way that tally-verification software is written independently of the voting system.

The ballot secrecy property of PrivApollo is based on indirection: the active voting assistant generates one half of the indirection map, and the voting terminal the other half. The actions of the voting terminal and the active voting assistant can be checked for correctness using a version of what is known as the Benaloh Challenge, instantiated as Helios. The vote is secret from the voting terminal and the active voting assistant if at least one of the two is honest. Even if both are dishonest and collude, the privacy is no worse than that of Helios or Apollo (in which the voting terminal knows the vote). Integrity in PrivApollo holds if at least one voting assistant (not necessarily the active voting assistant) is honest.

Coded voting systems such as Surevote [3], Scantegrity II [4], DEMOS [6] and Remotegrity [13], as well the voting system Punchscan [10], also use indirection to provide privacy from the voting terminal. In these other proposals, however, the indirection map is generated ahead of time by the voting system. Pre-generated indirection maps need to be delivered to the voter before voting begins and after the candidate lists are finalized; this is generally a small time window. Delivery on paper through the postal system can be a challenge, especially for voters who are abroad or in remote locations. If the indirection map is not delivered on paper, the electronic entity performing the delivery knows the map, and the vote is not private from this entity.

In PrivApollo, the indirection map is generated in real time, jointly by the voting terminal and the active voting assistant. PrivApollo's splitting of the indirection map into two components is similar to Punchscan's approach of splitting it into two ballot halves. With Punchscan too, however, the information on both ballot halves is pre-generated by a single entity, the voting system. The Du-Vote protocol [8] also generates the indirection map in real time and splits it into two halves. However, both privacy and integrity properties of Du-Vote require that the two entities not collude. In PrivApollo, integrity properties hold even if the two entities do collude.

In spite of the benefit of ballot secrecy with respect to the voting terminal, the use of indirection is known to pose usability challenges. We do not study usability in this paper, and more usability research and improvements are required before PrivApollo may be deployed in real elections.

This paper is organized as follows. Section 2 presents related work, and section 3 the trust model. Section 4 describes the protocol, section 5 provides a brief description of security and usability properties and section 6 concludes.

2 Related work

Helios [1] is perhaps the earliest practical proposal for end-to-end-verifiable voting without the use of paper. Several elections have been held using it, including those for the IACR and ACM. In the Helios protocol the voter enters her vote into the voting terminal, which provides her with an encryption of the vote. The Apollo protocol improves the verifiability of Helios with a two-step vote-casting process; voters enter a second credential after the encrypted vote is posted on the bulletin board, thus confirming it is correct. In both systems, the terminal encrypts the vote and hence knows it.

Riva and Ta-shma [11] propose a precinct-based protocol where the voter prepares two ballots at home, each consisting of a list of encrypted votes indexed by candidate. The terminal in the voting booth re-encrypts all encrypted votes. The voter challenges one encryption for each candidate and then casts the unchallenged one for her choice. Voter privacy is protected if the voting terminal at the precinct and the voter's home computer do not collude.

Code-voting systems, such as Surevote [3], provide voters with a code for each choice, and the voter casts the code in order to vote. The use of codes provides ballot secrecy and prevents the voting terminal from replacing the vote with another valid one. Remotegrity [13] is an extension of code-voting, enabling remote voting through the use of two credentials: one for casting the vote, and the other for confirming it.

Paper-based precinct voting system Punchscan [10] splits its indirection map onto two sheets, which when overlaid form the ballot and reveal the vote. Each sheet bears a mark, but the mark simply denotes the choice on one half of the indirection map, and each sheet by itself reveals no information about the vote. We use a similar idea to construct our electronic ballots; however, our ballots are constructed at the time of voting without secure communication with the voting system. While all voting systems, including PrivApollo, require secure communication with the voter for credential delivery, this may be done many months or weeks before the candidate lists are finalized, and does not pose the same challenge as that of delivering indirection maps in the small time window between finalization of candidates lists and the election.

Our use of indirection is similar to that of Chaum et al [5]. An important distinction is that the voting assistant communicates with the voting system only through the voting terminal in [5]. In PrivApollo*Codes* the voting assistant directly posts information on the bulletin board.

In addition to the voting systems described above, there are other proposals that address the privacy problem, but do so requiring participants or hardware to be trusted for integrity properties.

The Du-Vote [8] proposal has a voter experience similar to that of PrivApollo, but differs because it assumes a dedicated hardware token, and verifiability requires that the token and the voting terminal do not collude.

Michael Backes et al. [2] propose that voters encrypt the vote using a one-time pad, communicated through a trusted mobile phone using a secure channel.

The Pretty Good Democracy (PGD) voting system [12] has a vote casting phase that is identical to that of a generic code voting system. Its contribution is in the addition of a back-end: the return code that confirms vote receipt to the voter is sent only after a group of trustees verify that the corresponding vote is posted on the bulletin board. Thus the integrity property of PGD relies on the honesty of the group of trustees.

3 Model

We first introduce PrivApollo participants and then the assumptions.

3.1 Participants

Registrar: A registrar, R, generates, issues and checks credentials, which are delivered to the voter.
Election Authority: The election authority, EA, includes servers and election officials, and any software deployed on their behalf.
Voting Terminal: The voting terminal, VB, is the terminal and voting software used by the voter to cast her vote.
Voter: The human voter, V can read and compare strings, generate a cast or audit challenge and choose a candidate.
Voting Assistants: V has access to a number of voting assistants after the voting phase of the protocol is completed. The assistants help check that the voter's ballot was cast-as-intended and recorded-as-cast but do not participate in the protocol. The n additional devices are denoted VA_1, VA_2, ..., VA_n.
Active Voting Assistant: In PrivApollo, one of the voting assistants is an active voting assistant, AVA, which helps the voter generate ballots and check on VB and EA. AVA is a participant introduced in PrivApollo to protect privacy from the voting terminal.

3.2 Assumptions, including trust assumptions

First, we begin with the standard assumptions made by all internet voting systems, also shared by Apollo and PrivApollo.
Bulletin Board: A secure bulletin board, BB—with append-only-authenticated-write and public-read access—is available to all participants.
Channel Between Registrar and Voter: Credentials cannot be accessed by anyone and cannot be altered while in the channel.

Additionally, the following trust assumptions are standard regarding participants.
Registrar: The registrar is assumed honest[3].

[3] Apollo relaxes this requirement by using irrepudiable credentials (such as credentials under scratch-off as in Remotegrity) to thwart a registrar who attempts to use the voter's credentials to cast a vote. PrivApollo does not make any changes to Apollo's registrar and credentials.

Election Authority: The *EA* is not assumed honest for integrity properties, but is assumed honest for privacy properties. To achieve privacy in this context, a threshold-encryption scheme may be used.

For other participants, we first describe the standard assumptions, then any modifications due to Apollo, and, lastly, further modifications due to PrivApollo.
Voting Terminal: In the standard model, the voting terminal and any software on it (denoted *VB*, as in Apollo), like the *EA*, is not assumed honest for integrity properties, but is assumed honest for privacy properties. In particular, this assumption is made by Helios and Apollo. In PrivApollo, the voting terminal is not assumed honest for integrity *or* privacy properties.
Voter: The standard model assumes a human voter, *V*, who can read and compare strings, generate a cast or audit challenge and choose a candidate. *V* is not assumed honest, and among other things, may make false complaints against other participants.

In order to avoid clash attacks, Apollo additionally assumes *V* can generate low entropy strings to help create distinct ballots; this approach may be used in PrivApollo as well. This assumption is not easily satisfied by all voters, and a random number generator in the form of a token may be used instead, but the token would need to be trusted not to collude with the voting system or the voting terminal.

PrivApollo further assumes that *V* can read, remember and compare short strings for the purpose of indirection, which is also challenging. Large scale attacks may be thwarted by a few voters who are capable of checking low entropy strings and detecting the attack.
Voting Assistants: The standard model assumes that at least one of the assistants does not collude with the voting terminal for integrity properties. The assistants are not trusted for privacy properties.
Active Voting Assistant: As in Apollo, integrity requires that at least one voting assistant is honest. (For this purpose, an *AVA* is like any other voting assistant.) The privacy properties require that at least one of *VB* and the *AVA* is honest and does not collude with the other to determine the vote.

4 The PrivApollo Protocol

In this section we present the PrivApollo protocol. In the steps outlined below, all steps except the pre-voting phase and the ballot generation step are exactly as in Apollo. The pre-voting phase differs slightly, as noted below. Ballot generation is completely different from Apollo and forms the main contribution of this paper. It is described in detail in sections 4.1 and 4.2.
Credentials: *V* receives her credentials from *R*: a set of k *casting codes* and a *lock-in code*. k is the number of times a voter may correct an incorrect vote posted on *BB* against one of her casting codes. This would happen if the voting terminal changed the encrypted vote after receiving the casting code. In such a case, the voter would change the terminal and try again. $k - 1$ is thus the maximum number of dishonest terminals (who change the vote, not simply those who deny

service by not posting the vote) the voter may encounter before successfully casting her vote. There could be another process to request more casting codes.
Pre-Voting Phase: Before the voting session begins, V chooses n voting assistants VA_1, VA_2, ... VA_n. Larger values of n improve the robustness of the integrity properties. New to PrivApollo, she chooses one AVA to be used for the ballot generation procedure.
Role of Voting Assistants: After each protocol step, VB, every VA and the AVA each checks BB and provides feedback to V. If V is satisfied with the outcome of the check, she moves to the next step. If she determines that there is a problem, she should try to vote again on another terminal. She should always reuse an old credential unless she hears from the EA that it has been used (which would imply it had been used to post a vote on BB). This is because the number of casting codes is limited, and if they were all used up, the voter would have to make an effort to contact the EA to obtain more.
Initialization: V opens the voting application on VB and provides a short string for the session title. VB displays the (voting) session ID and a QR-code, and sends session ID to BB, which displays it. V scans the QR-code into an AVA and any VAs, and checks that they display the session ID (and) Title. That is, that they are able to see these on BB. The QR code contains, in addition to the session-ID, a symmetric key k_{rand} shared by VB and VAs, including AVA. k_{rand} can be used to decrypt posted messages related to encryption audits, so that the voter may choose who sees them. The QR code, and hence k_{rand}, are not posted on BB.
Ballot Generation: Here our protocol differs considerably from Apollo. We have two approaches to ballot generation: *PrivApollo Colors* and *PrivApollo Codes*, which we describe in sections 4.1 and 4.2 respectively.
Lock-in Phase: Once the ballot has been generated and the voter casts it using her Apollo casting credential, the VAs check the BB and inform the voter what has been posted for her session. If the voter is satisfied that the string posted is the one originally represented to her as her encrypted vote, she may return at any time to lock-in her vote. She may do so from any computer by identifying her session ID and adding her lock-in code, a second Apollo credential used to communicate that the encrypted vote has been posted correctly. Finally, she may check that the lock-in code has been posted, again, from any (other) computer. If it is not, she may try to lock-in the vote again, from any other computer.

4.1 *PrivApollo Colors*

In this section we present the simpler protocol for ballot generation, *PrivApollo Colors*. Note that colors may be replaced with shapes, for example, or audio words for audio (as opposed to visual) presentation of the ballot.

1. **Candidate-Color Correspondence Display**: VB generates a pseudorandom permutation π of the colors, leading to a candidate-color correspondence. VB publishes, on BB, public-key encryptions of each candidate-color correspondence using the public key of the EA and symmetric-key encryption of the set of colors used with k_{rand}. *VB displays candidate-color pairs.*

2. **Color Display**: *AVA* obtains all the encrypted information from the *BB*, informs the voter that it is available, decrypts the list of colors with k_{rand}, generates a pseudo-random permutation γ and *displays the corresponding permuted list of colors*, see Figure 1.

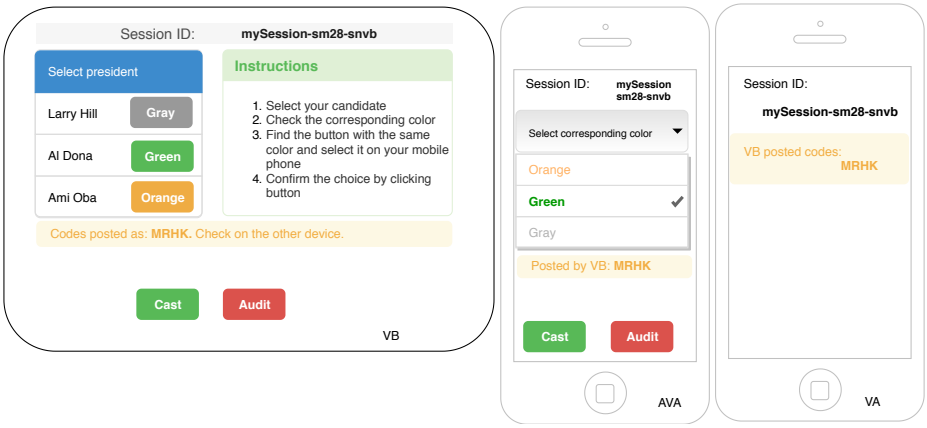

Fig. 1. Ballot Generation: *PrivApollo Colors*. *VB* permutes the colors and displays the candidate-color pairs. *VB* posts encryptions (here, the identifier of the encryption is *MRHK*). *AVA* displays that the encryptions were posted. The voter observes that the color for her candidate, Al Dona, is *Green*, selects *Green* on the *AVA* and confirms this choice.

3. **Color Choice**: *V* sees the correspondence between candidates and colors displayed on *VB*, selects her candidate, observes the corresponding color on *VB*, selects the same color on *AVA* and confirms it.
4. **Color Encryption**: *AVA* encrypts the chosen color and posts the encryption on *BB*. *V* checks on *VB* that, indeed, an encryption has been posted by *AVA* (see Figure 2).
5. **Encryption Challenge**: *V* makes a choice: whether to audit or cast the generated (encrypted) ballot.
   cast *V* enters an unused *casting code* (Apollo casting credential).
   audit *V* checks if her encrypted ballot represents the candidate she chose.
6. (audit) If the voter chooses to audit, *VB* and *AVA* each reveal, on *BB*, the randomness required to check the encryption. Each *VA* checks the encryption by trial and error over all possibilities, and communicates the result of the check:
   - The correspondence between candidates and assigned colors as generated by *VB*.
   - Which color was encrypted (and submitted by *AVA* to the BB).

   The voter may repeat the audit step as many times as she wishes. Each time, a fresh candidate-color correspondence is generated (goto step: 3).

7. (cast) If the voter enters a cast code (on either VB or AVA), each VA displays the code she entered and informs her that her vote is ready for locking.

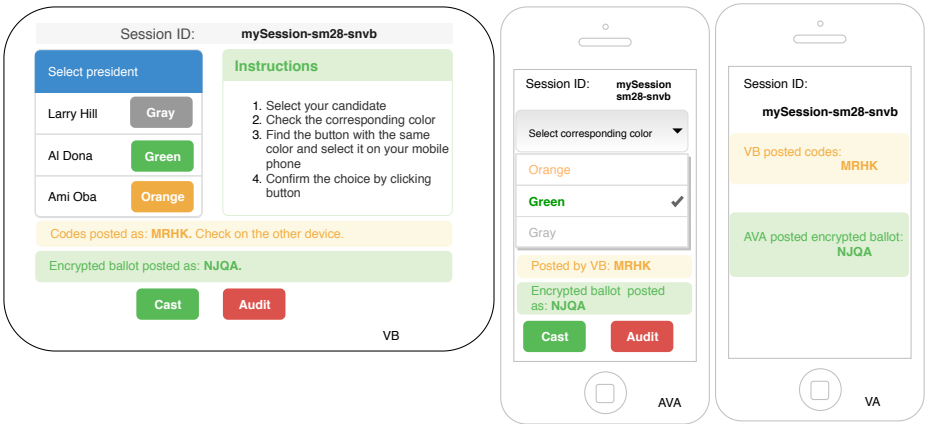

Fig. 2. Encryption Challenge: *PrivApollo Colors*. VB informs the voter that a string purporting to be the encrypted choice (of selected color *Green*) was published on BB by the AVA (in this case, the identifier of the encryption is $NJQA$). The voter may initiate an audit on either device: VB or AVA.

4.2 *PrivApollo Codes*

In this version of the ballot-generation procedure, VB presents to the voter, as in the *PrivApollo Colors* procedure, a list of candidate-color pairs, where the correspondence is pseudorandomly generated (see Figure 3). The AVA does more work than in the *PrivApollo Colors* procedure, generating a list of short codes, a distinct one for each color, and presenting a correspondence between color and code.

The two correspondences taken together result in a correspondence between candidate and code, and neither VB nor AVA can determine any information on this correspondence without collusion with the other. The voter identifies the color for her candidate on VB, then the code for the color on the AVA, and enters the code into VB, see Figure 3.

4.3 Tallying

Each of N cast ballots consists of several encryptions.

PrivApollo Colors Protocol 1.1

- Encryption of the ballot layout (sent by VB to BB in Step 1(c))

PrivApollo: ***Colors***

1. *VB* generates an encoded ballot with:
   (a) a canonical list of voting options $\langle o_1, o_2, \ldots, o_k \rangle$,
   (b) a random permutation π,
   (c) the permuted list of colors $\langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(k)} \rangle$ (see Figure 1).
   VB submits to the bulletin board the following encrypted values:

$$ballotLayout \leftarrow \begin{bmatrix} \langle \mathsf{Enc}_{EA}(o_1, r_{o_1}), \mathsf{Enc}_{EA}(c_{\pi(1)}, r_{c_1}) \rangle, \ldots \\ \ldots, \langle \mathsf{Enc}_{EA}(o_k, r_{o_k}), \mathsf{Enc}_{EA}(c_{\pi(k)}, r_{c_k}) \rangle \end{bmatrix}$$

$$innerCodes \leftarrow \mathsf{Enc}_{k_{rand}}(\langle c_1, \ldots, c_k \rangle).$$

   VB displays the ballot to *V*: $\begin{bmatrix} o_1 & c_{\pi(1)} \\ o_2 & c_{\pi(2)} \\ \ldots & \\ o_k & c_{\pi(k)} \end{bmatrix}$
2. Each *VA* performs the following steps:
   (a) downloads encrypted *ballotLayout* from *BB* and informs *V* that encryptions were posted by *VB*.
   (b) decrypts *innerCodes* with k_{rand} to obtain $\langle c_1, \ldots, c_k \rangle$.
3. Each *VA* displays the color options: $\begin{bmatrix} c_{(1)} \\ c_{(2)} \\ \ldots \\ c_{(k)} \end{bmatrix}$
4. *V* finds her candidate o_i and a corresponding color $c_{\pi(i)}$ on *VB*. Then it picks the same color on *AVA* – position j such that $c_{\pi(i)} = c_{(j)} = x$.
5. *AVA* does the following:
   (a) computes the encryption of the ballot: $c \leftarrow \mathsf{Enc}_{EA}(x, r)$, where r is the randomness used during encryption,
   (b) sends the encrypted vote to *BB*: $AVA \xrightarrow{c} BB$
6. *VB* and *VA*s inform the voter that c is posted on *BB* in the transcript of her $sessionID$
7. *V* makes a decision about cast/audit:
   Audit is selected: Protocol 1.3 is carried out.
   Cast is selected:
   (a) *V* is asked to enter: *Login* and *CastCode* (these can be combined into a single long string)
   (b) *VA*s display the *Login*/*CastCode* pair; *V* checks if they are as expected.

Protocol 1.1. The vote-casting procedure of PrivApollo *Colors* (simple; see Figures 1 and 2).

PrivApollo: Codes

1. *VB* generates an encoded ballot with:
   (a) a canonical list of voting options $\langle o_1, o_2, \ldots, o_k \rangle$,
   (b) a randomly selected list of inner-codes $\langle c_1, c_2, \ldots, c_k \rangle$ (see Figure 3 where the c_is are colors).
   (c) a permutation π.
   VB submits to the bulletin board the following encrypted values:

$$ballotLayout \leftarrow \begin{array}{l}[\langle \mathsf{Enc}_{EA}(o_1, r_{o_1}), \mathsf{Enc}_{EA}(c_1, r_{c_1}) \rangle, \ldots \\ \ldots, \langle \mathsf{Enc}_{EA}(o_k, r_{o_k}), \mathsf{Enc}_{EA}(c_k, r_{c_k}) \rangle],\end{array}$$

$$innerCodes \leftarrow \mathsf{Enc}_{k_{rand}}(\langle c_{\pi(1)}, \ldots, c_{\pi(k)} \rangle).$$

   VB displays the ballot to *V*: $\begin{array}{|ll|}\hline o_1 & c_1 \\ o_2 & c_2 \\ \ldots & \\ o_k & c_k \\ \hline\end{array}$
2. Each *VA* performs the following steps:
   (a) downloads $ballotLayout, innerCodes$ from *BB* and informs *V* that encryptions were posted by *VB*.
   (b) decrypts $innerCodes$ with k_{rand} to obtain $\langle c_{\pi(1)}, \ldots, c_{\pi(k)} \rangle$.
   (c) displays list of decrypted $innerCodes$.
3. *AVA*
   (a) generates randomly selected list of vote-codes $\langle v_1, v_2, \ldots, v_k \rangle$.
   (b) submits to *BB* an encryption of the correspondence between inner-codes and vote-codes:

$$voteCodes \leftarrow \begin{array}{l}[\langle \mathsf{Enc}_{EA}(c_{\pi(1)}, r_{\pi_1}), \mathsf{Enc}_{EA}(v_1, r_{v_1}) \rangle, \ldots \\ \ldots, \langle \mathsf{Enc}_{EA}(c_{\pi(k)}, r_{\pi_k}), \mathsf{Enc}_{EA}(v_k, r_{v_k}) \rangle].\end{array}$$

   (c) displays the code-sheet: $\begin{array}{|ll|}\hline c_{\pi(1)} & v_1 \\ c_{\pi(2)} & v_2 \\ \ldots & \\ c_{\pi(k)} & v_k \\ \hline\end{array}$
4. *V* who wants to cast a ballot for candidate o_i:
   (a) finds the corresponding inner code c_i ($o_i \leftrightarrow c_i$) displayed on *VB*,
   (b) finds the corresponding vote code $x = v_{\pi^{-1}(i)}$
   sends vote choice to *VB*: $V \xrightarrow{x} VB$
5. *VB* sends to the bulletin board $VB \xrightarrow{c} BB$ the encryption of the vote code $c \leftarrow \mathsf{Enc}_{EA}(x, r)$, where r is the randomness used for encryption
6. *VA*s inform the voter that x is posted on *BB* in the transcript of her $sessionID$. Moreover *AVA* higlights color c_i corresponding to x.
7. *V* makes a decision about cast/audit:
   Audit is selected then Protocol 1.3 is performed.
   Cast is selected:
   (a) *V* is asked to enter: *Login* and *CastCode* (these can be combined to be a single long string)
   (b) *VA*s display the *Login*/*CastCode* pair; *V* checks if they are as expected.

Protocol 1.2. The vote-casting procedure of *PrivApollo Codes* (see Figure 3).

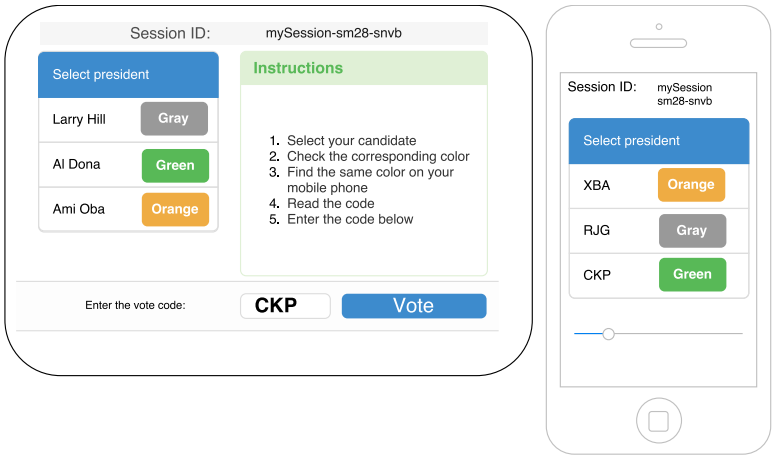

Fig. 3. Ballot Generation: *PrivApollo Codes*. The voter observes that the color corresponding to her candidate *Al Dona*, as displayed by *VB*, is *Green*. The corresponding code for *Green* displayed on *AVA* is *CKP*. The voter enters *CKP* in *VB*.

Audit (*PrivApollo Colors* and *PrivApollo Codes*)

1. *VB* sends to *BB* the randomness used to generate the ballot and encrypt the vote:
$$c_{\text{VB}} = \text{Enc}_{k_{rand}}(r, \langle r_{o_1}, \ldots, r_{o_k} \rangle, \langle r_{c_1}, \ldots, r_{c_k} \rangle)$$
2. *AVA* sends to *BB* the randomness:
$$c_{\text{AVA}} = \text{Enc}_{k_{rand}}(r), \quad \textit{in color version}$$
$$c_{\text{AVA}} = \text{Enc}_{k_{rand}}(r, \langle r_{\pi_1}, \ldots, r_{\pi_k} \rangle, \langle r_{v_1}, \ldots, r_{v} \rangle), \quad \textit{in code version}$$
.
3. The *VA*s decrypt c_{VB} and c_{AVA} and present the vote x' to V (with the randomness, *VA*s can recover the plaintext by brute force).
4. V accepts or not based on what the other VAs say the vote decrypted to:
$x = x'$ Prepares new encryption; goto step (1) of the Protocols 1.1 1.2
$x \neq x'$ Begins again with new *VB* and, if necessary, new *VA*s

Protocol 1.3. The audit procedure PrivApollo– both versions.

- List of encrypted inner codes (sent by *VB* to *BB* in step 1c)
- Encryption of inner code (color) selected by the voter (inner code sent by *V* to *AVA* in step 4, encryption sent by *AVA* to *BB* in step 5).

PrivApollo Codes Protocol 1.2

- Encryption of the ballot layout (sent by *VB* to *BB* in Step 1(c))
- List of encrypted inner codes (sent by *VB* to *BB* in step 1c)
- Encryption of the correspondence between inner codes and vote codes (sent by *AVA* to *BB* in step 3b)
- An encrypted vote code selected by the voter (inner code sent by *V* to *VB* in step 4, encryption sent by *VB* to *BB* in step 5).

For both vote casting methods, the tally phase consists of two phases. The role of the first phase is to select a valid row of the ballot layout that corresponds to the submitted vote code (or color). To protect ballot privacy, the corresponding row – that is (re)encrypted selected option, goes through the second phase of mixing and re-encryption.

PrivApollo: VoteCodes ReEncryption – Tallying Phase 1a
Input: $\langle ballotLayout_i, voteCodes_i, c_i\rangle_{i=1}^{N} = \langle bL_i, vC_i, c_i\rangle_{i=1}^{N}$

1. pick at random σ a permutation of N elements.
2. for each $i = 1 \ldots N$ do:
   (a) select k-element permutations $\pi_{i,1}$, $\pi_{i,2}$
   (b) on input:
   $bL_i = [\langle \alpha_1, \beta_1\rangle, \ldots, \langle \alpha_k, \beta_k\rangle]$;
   $vC_i = [\langle \gamma_1, \delta_1\rangle, \ldots, \langle \gamma_k, \delta_k\rangle]$;
   c_i.
   (c) output (for $j = 1 \ldots k$):
   $bL_{\sigma(i)}[j] := \left\langle \mathsf{ReEnc}(\alpha_{\pi_{i,1}(j)}), \mathsf{ReEnc}(\beta_{\pi_{i,1}(j)})\right\rangle$;
   $vC_{\sigma(i)}[j] := \left\langle \mathsf{ReEnc}(\gamma_{\pi_{i,2}(j)}), \mathsf{ReEnc}(\delta_{\pi_{i,2}(j)})\right\rangle$;
   $c_{\sigma(i)} := \mathsf{ReEnc}(c_i)$.

Protocol 1.4. Tallying phase 1a for *PrivApollo Codes*. A code for a mix-server. ReEnc() denotes (ElGamal) re-encryption.

After the end of phase 1 (1a and 1b), the last element (c_i) of each 3-tuple of vote i gets decrypted: revealing the vote code that was entered by the voter. Since $c = \mathsf{Enc}_{EA}(x, r)$ and $x = v_{\pi^{-1}(i)}$, the value $v_{\pi^{-1}(i)}$ becomes public.

Also each of $\delta_{i,j}$ is decrypted ($\delta_{i,j} = \mathsf{Enc}_{EA}(c_{\pi(j)})$). For each ballot i, exactly one decrypted $\delta_{i,j}$ will match decrypted c_i. The mix-server deletes all unmatched rows of $\langle \gamma_{i,j}, \delta_{i,j}\rangle$.

PrivApollo: VoteCodes Decryption – Tallying Phase 1b
Input: $\langle ballotLayout_i, voteCodes_i, c_i\rangle_{i=1}^{N} = \langle bL_i, vC_i, c_i\rangle_{i=1}^{N}$
Shared key: $\mathcal{K}_m$

1. pick at random σ a permutation of N elements.
2. for each $i = 1 \dots N$ do:
   (a) select k-element permutations $\pi_{i,1}$, $\pi_{i,2}$
   (b) on input:
   $bL_i = [\langle \alpha_1, \beta_1 \rangle, \dots, \langle \alpha_k, \beta_k \rangle]$;
   $vC_i = [\langle \gamma_1, \delta_1 \rangle, \dots, \langle \gamma_k, \delta_k \rangle]$;
   c_i.
   (c) output (for $j = 1 \dots k$):
   $bL_{\sigma(i)}[j] := \left\langle \mathsf{ReEnc}(\alpha_{\pi_{i,1}(j)}), \mathsf{ReEnc}(\beta_{\pi_{i,1}(j)}) \right\rangle$;
   $vC_{\sigma(i)}[j] := \left\langle \mathsf{ReEnc}(\gamma_{\pi_{i,2}(j)}), \mathsf{Dec}_{\mathcal{K}_m}(\delta_{\pi_{i,2}(j)}) \right\rangle$;
   $c_{\sigma(i)} := \mathsf{Dec}_{\mathcal{K}_m}(c_i)$.

Protocol 1.5. Tallying phase 1b (*PrivApollo Codes*). A code for a mix-server. $\mathsf{Dec}_{\mathcal{K}_m}(\cdot)$ denotes shared key of threshold encryption which was used to generate *EA*'s public key.

After the Phase 1 the following cryptograms remain:

$$\langle ballotLayout_i, \gamma_{i,j_i}\rangle_{i=1}^{N} = \langle bL_i, c_i\rangle_{i=1}^{N}.$$

In Phase 2 Protocol 1.4 and Protocol 1.5 are run sequentially (with bL playing the role of vC). After Phase 2 is completed, one can decode the only matching α which encodes the voting option selected by the voter, o_x.

5 Security and Usability Discussion

In the *PrivApollo Colors* procedure, the *AVA* could easily replace the vote with another valid one (change the color to another valid color), though it would not know which candidate the color corresponded to (the randomization attack). On the other hand, while the *PrivApollo Codes* procedure requires the voter to deal with two indirections and is hence more complex, it is hard for *VB* to change the vote to another valid vote. This is because it does not know other valid codes without colluding with *AVA*, or maliciously attempting to determine a valid code by, for example, photographing the *AVA* screen. While the *PrivApollo Codes* procedure makes the randomization attack harder, it does allow the adversary to easily invalidate the vote. As in Apollo, each attack would be detected by the alert voter. The randomization attack and vote validation can be successfully carried out on Apollo and Helios as well.

Note that, while the integrity properties of the protocol are resilient to collusion between the *BB* and the *AVA*, we do rely on the assumption that neither party knows all the valid credentials (casting codes and lock-in codes). That is,

for example, *AVA* is not able to photograph the credential sheet that the voter may have received in the post.

If either *AVA* or *VB* wishes to learn the ballot contents, it needs to cooperate with the other in order to learn the indirection. Such cooperation includes the surreptitious access to data by one of the parties.

The tallying process is divided into two phases; the information revealed at the end of each phase helps neither *VB* nor *AVA* get more knowledge about the cast ballot.

As in Apollo, the use of the *Benaloh challenge* enables voters to detect attempts to manipulate the vote. This approach hence has all the strengths and weaknesses of a cast-and-challenge approach, including the fact that voters may believe they had already verified when they had not [9]. The use of lock-in codes allows her to communicate that a problem occurred, and allows her to attempt to vote again. The inclusion of an active voting assistant does not change these properties, because the actions of the voting assistant are included in the audit. The integrity properties of Apollo are unchanged. The use of the active voting assistant may be viewed as a splitting of the voting terminal into two entities, reducing the reliance on a single entity to protect ballot secrecy, and improving on the privacy properties of PrivApollo.

We do assume independent devices, but, even if the devices are not independent, or malware is transmitted from *VB* to all the *VA*s through the QR code, the verifiability properties hold if the voter performs all the checks using a different *VA* into which the voter herself keys in the session ID, or if the voter goes through the verification steps as in an end-to-end-verifiable protocol that does not explicitly incorporate a voting assistant. The voter, may, in fact, use a number of out-of-band voting assistants, in which case the interaction with the voting assistant will be no less usable than in, say, Helios, though privacy with respect to the voting terminal will be greater. If the voter uses a single *AVA* and no other *VA*s, one of the *VB* or the *AVA* need to be honest for the integrity property, which is the same requirement as Apollo used with a single *VA*.

The scheme seems to be well suited for elections with few candidates. When the number of candidates is large, it would not be possible to find a large enough number of colors that are sufficiently easily distinguished by voters. Beyond a couple of candidates, it is likely that the error rate of the indirection would increase. Similar problems would arise if one used shapes instead of colors. In such a case, the use of a short alphanumeric code would be a better choice. Even so, voters might make errors while comparing alphanumeric strings. Finally, the encryption audit check requires the *VA* to check every possibility with the randomness revealed, and hence this would be very inefficient for a large number of candidates or for more complex elections (other than plurality elections).

We are not aware of any protocols that offer similar security properties given a similar trust model. We acknowledge that this has been with a loss in usability; however, that presents interesting future work.

6 Conclusions

We have presented a fully electronic scheme that is end-to-end voter verifiable and also provides ballot secrecy from the devices used to cast a ballot. The privacy property holds if the Voting Booth does not collude with the Active Voting Assistant. Integrity is achieved as long as at least one Voting Assistant used by the Voter is honest.

References

1. B. Adida. Helios: web-based open-audit voting. In *USENIX Security Symposium*, pages 335–348, 2008. 1, 2
2. M. Backes, M. Gagné, and M. Skoruppa. Using mobile device communication to strengthen e-voting protocols. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 237–242. ACM, 2013. 2
3. D. Chaum. Surevote, International Patent WO 01/55940 A1. Technical report, 2001. 1, 2
4. D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. Vora. Scantegrity: End-to-end voter verifiable optical-scan voting. *IEEE Security and Privacy*, 6(3):40–46, May/June 2008. 1
5. D. Chaum, A. Florescu, M. Nandi, S. Popoveniuc, J. Rubio, P. L. Vora, and F. Zagórski. Paperless independently-verifiable voting. In *International Conference on E-Voting and Identity*, pages 140–157. Springer Berlin Heidelberg, 2011. 2
6. N. Chondros, B. Zhang, T. Zacharias, P. Diamantopoulos, S. Maneas, C. Patsonakis, A. Delis, A. Kiayias, and M. Roussopoulos. D-DEMOS: A distributed, end-to-end verifiable, internet voting system. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*, pages 711–720, 2016. 1
7. D. G. Gawel, M. Kosarzecki, P. L. Vora, H. Wu, and F. Zagorski. Apollo - end-to-end verifiable internet voting with recovery from vote manipulation. In *International Joint Conference on Electronic Voting E-VOTE-ID 2016*, 2016. 1
8. G. S. Grewal, M. D. Ryan, L. Chen, and M. R. Clarkson. Du-Vote: Remote electronic voting with untrusted computers. In *2015 IEEE 28th Computer Security Foundations Symposium (CSF)*, pages 155–169. IEEE, 2015. 1, 2
9. K. Marky, O. Kulyk, K. Renaud, and M. Volkamer. What did I really vote for? On the usability of verifiable e-voting schemes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, United States, 4 2018. Association for Computing Machinery (ACM). 5
10. S. Popoveniuc and B. Hosp. An introduction to Punchscan. In *WOTE*, 2006. 1, 2
11. B. Riva and A. Ta-Shma. Bare-handed electronic voting with pre-processing. In *EVT*, 2007. 2
12. P. Y. A. Ryan and V. Teague. Pretty Good Democracy. In *Security Protocols XVII, 17th International Workshop, Cambridge, UK, April 1-3, 2009. Revised Selected Papers*, pages 111–130, 2009. 2
13. F. Zagórski, R. T. Carback, D. Chaum, J. Clark, A. Essex, and P. L. Vora. Remotegrity: Design and use of an end-to-end verifiable remote voting system. In *Applied Cryptography and Network Security*, volume 7954. Springer, 2013. 1, 2