Fixing the Achilles Heel of E-Voting: The Bulletin Board

Lucca Hirschi*
Inria & LORIA
Nancy, France
lucca.hirschi@inria.fr

Lara Schmid*
ETH Zurich
Zurich, Switzerland
schmidla@inf.ethz.ch

David Basin
ETH Zurich
Zurich, Switzerland
basin@inf.ethz.ch

***Abstract*—The results of electronic elections should be verifiable so that all cheating is detected. To support this, many protocols employ an electronic bulletin board (BB) for publishing data that can be read by participants and used to perform verifiability checks. We demonstrate that the BB is itself a security-critical component that has often been treated far too casually in previous designs and analyses. In particular, we present novel attacks on the e-voting protocols Belenios, Civitas, and Helios that violate some of their central security claims under realistic system assumptions. These attacks were outside the scope of prior security analyses as their verifiability notions assume an idealized BB.**

To enable the analysis of protocols under realistic assumptions about the BB, we introduce a new verifiability definition applicable to arbitrary BBs. We identify a requirement, called *final-agreement*, and formally prove that it is sufficient and, in most cases, necessary to achieve verifiability. We then propose a BB protocol that satisfies final-agreement under weak, realistic trust assumptions and provide a machine-checked proof thereof. Our protocol can replace existing BBs, enabling verifiability under much weaker trust assumptions.

Note: For reproducibility, our machine-checked proofs are available at [1].

I. Introduction

Physical bulletin boards are used to publish announcements, for example at the town hall. An *electronic bulletin board*, henceforth referred to as *BB*, has a similar purpose, but can be accessed *remotely*, *e.g.,* by publishing its content on a website. While BBs are deployed in various contexts, they are particularly important for electronic voting (e-voting) protocols.

For e-voting to be trustworthy, the participants must be convinced that the final tally is correctly computed from all eligible voters' ballots. To this end, the participants must be able to *verify* that all e-voting authorities behaved as specified, even when some of them are not trustworthy. For most e-voting protocols, verifiability is achieved by voters and auditors performing checks on data published on a BB. In this paper, we focus on those BBs that are used for verifiability checks, where readers read their content and check the data they read. For instance, a voter may check that her ballot was recorded correctly after casting it. Also, auditors or voters may check that all ballots were processed correctly by the authorities.

a) State of the art: For verifiability checks to be meaningful, the BB must provide some guarantees, such as all participants agree on its content. However, to the best of our knowledge, no prior work has studied which precise (minimal) guarantees must be satisfied by a BB for *verifiability* to hold. All verifiability definitions surveyed in [2] and most e-voting protocols that are formally proven to provide verifiability [3]–[5] make the overly conservative assumption of an *idealized BB* such as a shared, universally accessible memory, a broadcast channel, or even a storage where a *single* content is published at all times. Even prior works that claim to consider "malicious" BBs [6]–[10] where the BB's content is under adversarial control, still consider an idealized BB (a storage publishing a single content).

Some researchers regard the realization of BBs satisfying such strong requirements as an orthogonal problem [4], [5], [11]. Thus, it is unclear how and whether these assumptions can be met in practice. Other researchers have proposed concrete BB designs. For instance, Civitas [12] proposes a signed BB, Helios [13] suggests that BB contents are (re)posted by several auditors, [14] makes use of a Byzantine Fault Tolerant (BFT) protocol, and [15]–[17] suggest to use the BB protocol presented in [18]. Whereas these designs do not aim to realize an idealized BB, we shall see that they fail to provide sufficiently strong guarantees for verifiability. We shall also see why solutions based on distributed ledgers are unsuitable.

Idealized BBs are fine in theory but not in practice. Indeed, reference implementations of the state-of-the-art systems Belenios, Civitas, and Helios [4], [12], [13], that have been extensively used, notably in academia (*e.g.,* UCLouvain, Princeton, ACM, IACR), as well as the currently running web-based deployments of Belenios and Helios [19], [20] use BBs with too weak guarantees. Therefore, we shall see that the centralized entity running the BB must actually be trusted for verifiability to hold in practice. However, this assumption is unreasonable and at odds with the recent, substantial efforts to minimize the required trust assumptions in e-voting designs and proofs [6], [11], [21].

Thus, whereas e-voting designs make too strong *assumptions* about the BB, actual deployments provide too weak *guarantees* for them. This mismatch and the imprecise treatment of the BB in prior works call for a thorough analysis of the BB's role with respect to verifiability and raise the following questions. How does a BB that is under adversarial control impact verifiability in practice? What requirements must be satisfied by a BB for verifiability to hold? Can a concrete BB protocol

*Both authors contributed equally to this research.

achieve such requirements under realistic assumptions?

b) Contributions: We make four main contributions. First, we demonstrate that there is a mismatch between BB assumptions in verifiability claims and proofs and actual BB realizations. By considering idealized BBs like those mentioned above, prior works fall short of capturing realistic BB use cases and deployments. In particular, a malicious BB can provide different readers different contents independent of its current state (similar to the notion of *equivocation* [22], [23]). This opens even well-designed protocols to serious attacks. In particular, we present novel and practical attacks on the state-of-the-art e-voting systems Belenios [4], Civitas [12], and Helios [13]. Some of our attacks exploit the BB equivocating contents, but never the *election outcome*, to readers. We show that these systems fail to provide verifiability (and privacy) under the threat models for which they are claimed to be secure. As verifiability has previously only been proven with an idealized BB, these attacks were missed in prior formal analyses.

Second, we propose a new verifiability definition that accounts for malicious BBs. Consequently, our definition covers realistic scenarios and captures our attacks, as opposed to previous work assuming idealized BBs. We base our definition on the generic verifiability definition of [2] that subsumes all of the definitions surveyed in [2], which all assume an idealized BB. However, in stark contrast to [2], our new definition *verifiability*$^+$ is also suitable for malicious BBs. As expected, verifiability$^+$ does not hold for arbitrary BBs, which motivates our analysis of which BB properties are actually needed for the entire e-voting system to satisfy verifiability$^+$.

Third, we identify a new BB property, called *Final-Agreement* (FA for short), that is weaker than conventional BB requirements, yet sufficient to achieve verifiability$^+$. FA requires that any content read from the BB at some point in time is also contained in a distinguished *final* version of the BB, wherein the election result is published. Furthermore, all readers agree on this final content. FA does not, however, impose any relation between the writes and the reads or between different reads performed on non-final BBs. We formally show that, in most e-voting protocols, FA is the weakest BB requirement that suffices to achieve verifiability$^+$. Also, we prove that any protocol satisfying verifiability with an idealized BB, satisfies verifiability$^+$ for a BB satisfying FA, which requires weaker trust assumptions that can be met in practice.

Finally, we propose a BB protocol that satisfies FA. Similar to other approaches [14], [24], [25], we assume that the BB is implemented using multiple peers signing BB contents, only some of which must be trusted. However, in contrast to prior BB protocols that fail to provide FA, we explain how it can be enforced with carefully designed policies that peers must check prior to signing. Since such proofs are subtle, we formalize our protocol and the FA property as an event-based model and provide a machine-checked proof that the protocol satisfies FA. Our protocol requires weaker trust and system assumptions than previous BB approaches based on BFT such as [14].

Overall, our results show that unrealistically strong BBs in designs and verifiability proofs can be replaced by our realizable BB protocol and that verifiability is still (provably) satisfied. We thereby effectively and substantially weaken the required trust assumptions in e-voting protocols.

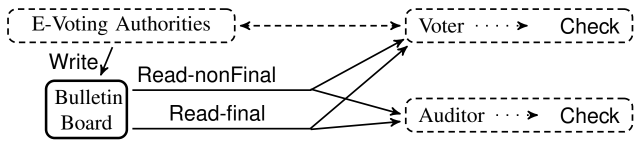


Fig. 1. Typical BB setup in e-voting. The BB functionalities are depicted by solid lines, the verifiability checks by dotted lines, and the remaining architecture by dashed lines.

c) Outline: In Section II, we present our system model, our threat model, and the specification language we use. In Section III, we review some e-voting designs with their BB auditing mechanisms and show how they can be attacked. We then propose a new verifiability definition that takes such scenarios into account. We define the FA property in Section IV and argue why it is sufficient and, in many cases, necessary for verifiability. We then present our BB protocol and formally establish that it satisfies FA in Section V. In Sections VI and VII we discuss related work, including those solutions based on distributed ledgers, and draw conclusions.

II. Bulletin Board (BB) Model

A. Setup, System, and Adversary Assumptions

1) Functionalities: BBs are used to publish information to a group of readers. A BB therefore provides, at a minimum, functionalities for *writing* and *reading* content to and from it. For many use cases, including e-voting, the BB's content is intended to reach a final state where the *final content* represents the result of the process that the BB tracks. For example, an election's final content includes the election outcome. We shall see that for e-voting, the guarantees required when reading the final content are different from the guarantees required when reading *non-final* contents. The former guarantees are strictly stronger and include a strong form of agreement between readers, whereas the latter can typically be relaxed. We thus propose a BB where the reading functionality is split into two *a priori* distinguished functionalities: Read-final for reading the BB's final content(s) and Read-nonFinal for reading any BB content, including non-final ones. The third and final functionality is Write.

2) Setup: We focus on the interactions between the BB and verifiability in e-voting. Thus, we consider BBs that are solely used to store election-relevant data and to retrieve data to check for *verifiability*. Checking verifiability intuitively entails checking that all participants followed their specification and the election's outcome is thus trustworthy. In this setting, it is common that writers are voting authorities and readers are auditors and voters (or their machines), who carry out some *verifiability checks* on the BB's content. A typical BB architecture for e-voting is depicted in Figure 1.

3) System Model: A concrete BB solution must be analyzed together with appropriate system and adversary assumptions. We discuss these next.

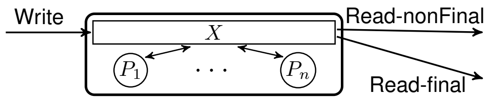


Fig. 2. System setup for our BB protocol: the BB peers are depicted by circles, the proxy peer X by a rectangle, and the communication channels by arrows.

a) System Assumptions: A concrete BB can be realized by a single role (*e.g.,* [18]) or by several (equal or different) roles, which we call *peers*, that run a protocol together (*e.g.,* [14]). The system assumptions state which communication channels are available between readers, writers, and peers.

We note that reader-interconnectivity would allow for BB solutions based on BFT protocols run by the readers, for example readers cross-checking their views of the BB. However, assuming reader-reader communication is a very strong assumption that is unrealistic for e-voting at scale, *i.e.,* medium and large scale elections where all voters can be readers. Indeed, this would require an infrastructure such as a Public Key Infrastructure (PKI) that voters use to identify and authenticate other genuine, eligible voters. Moreover, sufficiently many voters must be online at all times with sufficient bandwidth and storage (the BB content can be very large as it may contain large zero-knowledge proofs (ZKP) for many voters, see Section III-B2c). Finally, BFT would require that a given percentage of the voters' platforms are trusted and non-compromisable[1], which is hard to ensure in practice. As such assumptions are unrealistic for e-voting systems, we exclude reader-reader communication from our system model.

Since readers cannot directly communicate with each other to synchronize their BB views, they must rely on trusted third parties, which may be centralized or decentralized. We seek a decentralized solution as depicted in Figure 2 that uses a parameterized number n of peers $P_1, \ldots, P_n$. In addition, we introduce a distinguished entity, called the *proxy peer* X, which has communication channels with the readers, writers, and all peers. This setup is more realistic than setups requiring all readers and writers to be directly connected to all peers. Even though X is modeled as a single entity, it can be physically replicated on different servers to avoid a single point of failure.

b) Threat Model: We assume that all communication is over an insecure network controlled by the adversary. Additionally, some of the participants can be *malicious*, *i.e.,* the adversary knows all of their secrets and controls them. We allow for static and dynamic compromise, *i.e.,* agents can be compromised before or during the execution. All nonmalicious agents are *honest* and always follow their specification.

We denote by n_m the number of BB peers P_i that are malicious and by $n_h = n - n_m$ the number of honest peers. We do not require the proxy peer X to be honest. Because of this and our assumption that messages can be dropped by the adversary, it is always possible that a reader may be unable to read the BB. We will later give a lower bound on n_h that is required to achieve the BB security goals as a function of how many messages originating from the peers arrive at a reader (through X), hence balancing availability and trust assumptions.

[1]This is (only) problematic for e-voting protocols that do not assume honest voters' platforms and that rely on specialized devices instead [21], [26]–[28].

B. Formal Specifications in Event-B

1) Event-B Definitions: We use an event-based model based on Event-B [29] to formally describe protocols. First, we introduce standard notation and definitions for Event-B specifications. We shall use standard mathematical notation and functional programming concepts such as typed values, types as sets of values, $\equiv$ for the equality between values, and $\triangleq$ for the equality between types. We denote by $r = (\!| x_1 = t_1, \ldots, x_n = t_n |\!)$ a *record* that respectively stores the value t_i with the label x_i, *i.e.,* $r.x_i \equiv t_i$. For a set S, $\mathcal{P}(S)$ denotes the *powerset* of S. $\bar{\cdot}$ denotes vectors, and $\{a_i\}_{i \in J}$ denotes a set of elements a_i indexed by elements in the set J.

Definition 1 (Specifications)**.** *A* transition system *is a tuple $\mathcal{T} = (\Sigma, \Sigma_0, \rightarrow)$, where Σ is the state space, $\Sigma_0 \subseteq \Sigma$ is the set of initial states, and $\rightarrow \in (\Sigma \times \Sigma)$ is the transition relation. A* behavior *σ of $\mathcal{T}$ is a sequence of states $\sigma = s_0.s_1.\cdots.s_n$ such that $s_0 \in \Sigma_0$ and $\forall i \in [0, n)$, $(s_i, s_{i+1}) \in \rightarrow$. $\mathsf{Be}(\mathcal{T})$ denotes the set of all behaviors.*

A specification *is defined by a set of typed* state variables*, which define a set of states Σ containing values for the state variables (which together can be seen as constituting a record), and a set of* events*, which define a transition relation over Σ. We denote a state whose state variables $a, b, \ldots$ have the values $v_1, v_2, \ldots$ by $(\!| a = v_1, b = v_2, \ldots |\!)$.* Events *are of the form $\mathsf{Ev}(\overline{x}) \equiv \{(s, s') \mid G(\overline{x}, s) \wedge s'.\overline{v} := \overline{f}(\overline{x}, s)\}$, where Ev is the event name, $\overline{x}$ are the event's parameters, $\overline{v}$ are the state variables, $G(\overline{x}, s)$ is a conjunction of* guards*, and $s'.\overline{v} := \overline{f}(\overline{x}, s)$ is an* action *with the* update function *$\overline{f}$. The guards are first-order formulae over $s.\overline{v}$ and $\overline{x}$ and determine when the event is* enabled*. If the event is enabled, the action $s'.\overline{v} := \overline{f}(\overline{x}, s)$ assigns values to state variables in the state s'. The set of all events defines a transition relation corresponding to applications of the events with arbitrary event parameters. Therefore, a specification and a set of initial states define a transition system. We assume that all specifications implicitly include the event $\mathsf{Skip}() \equiv \{(s, s') \mid s'.\overline{v} := s.\overline{v}\}$, which models stuttering steps. We denote by $V(S)$ the set of state variables of a specification S.*

Two specifications S_1 and S_2 can be combined into a new specification, denoted by $S_1 \cup S_2$, by taking the union of the state variables, events, and the state variables' initial values. Note that the initial values of the shared state variables in S_1 and S_2 must be equal.

2) E-voting Protocol and BB Specification Framework:

a) Parameterized BB Contents: For the sake of generality, we impose minimal restrictions on the contents that the BB may send and receive and we shall work with a *lattice* structure. Formally, we assume given an uninterpreted set $\mathcal{W}$ of all possible BB contents with a relation $\sqsubseteq_b$ that form a lattice whose join and meet are respectively written as $\cup_b$ and $\cap_b$. We also assume a bottom element $\mathrm{B}_\perp$. The relation $\sqsubseteq_b$ expresses inclusion between BB contents. The set $\mathcal{W}$

abstractly represents contents and does not necessarily match with the BB's actual internal representation. Furthermore, $\mathcal{W}$ may contain *partial bulletin boards*, such as a single item like a ballot. We establish our results in this general setting; but for intuition one can also work with the following instance of such a lattice: the power set of a given set $\mathcal{I}$ of items that the BB contents may contain. Namely, $\mathcal{W} := \mathcal{P}(\mathcal{I})$, $\sqsubseteq_b := \subseteq$, $\cup_b := \cup$, $\cap_b := \cap$, and $\mathrm{B}_\perp := \emptyset$.

We also assume that BB contents can be published at different stages in time, which we denote by *phases* that are interpreted by positive integers acting as counters. Each $\mathrm{B} \in \mathcal{W}$ is thus associated with a phase, according to the function $\mathsf{ps} : \mathcal{W} \to \mathbb{N}^+$ and $\mathsf{ps}(\mathrm{B}_\perp) = 1$.

b) Our Framework, the Big Picture: Our definitions model the BB's role in verifiability in e-voting and thus focus on the interactions between e-voting entities, the BB, and verifiability checks. The rest is abstracted away. We therefore formalize a protocol as the combination of two specifications, modeling the BB and the rest of the e-voting system, which can produce write and verifiability check requests. We describe how these two specifications interact through specific state variables, as explained next and depicted in Figure 3.

An agent A accessing the Write functionality to write some content, shown in solid, red arrows in Figure 3, is modeled as this content being added to the state variable wr (*write requests*). Depending on the BB's specification, the BB may then add this content to the state variable w (*writes*). Hence, the state variables wr and w respectively record the write requests sent to and actually processed by the BB.

To evaluate a verifiability check on some BB data, an agent A can send check requests to access one of the BB's two read functionalities. We distinguish a subset of verifiability checks, called *final-only* checks, that can only be evaluated on BB contents obtained by the Read-final functionality. An agent A evaluating a final-only verifiability check is modeled in two steps: (1) A's check is added to a set of *check requests* cr, and (2) depending on the BB specification, the BB may *process* this request and provide A with some content, which is stored with the original request in the fc (*final-checks*) state variable (depicted with dashed, blue arrows in Figure 3). We abstract away the actual reads (green dotted arrows on the right of Figure 3) and only keep track of check requests that were processed. The case of non-final check requests is similar, except that the processed check requests are stored in nfc (*non-final-checks*) rather than in fc. Finally, the e-voting specification has a state variable agents (not shown in Figure 3) that keeps track of all honest and malicious running agents.

c) Protocol and BB Specifications: We now formalize the above in Event-B. We respectively denote by $\mathcal{A}$ and $\mathcal{M}$ the uninterpreted sets of possible agents and messages, and by $\mathcal{C}$ the set of possible verifiability checks. A verifiability check $C \in \mathcal{C}$ is a predicate of the form $C(\mathrm{B}, \overline{x})$ that represents a property checked on the BB's content $\mathrm{B} \in \mathcal{W}$, possibly with additional data $\overline{x} \in \mathcal{M}^*$ in the checker's possession. We assume some $\mathcal{C}_\mathsf{f} \subseteq \mathcal{C}$ containing all the final-only verifiability checks, and let $\mathcal{C}_\mathsf{nf} = \mathcal{C} \backslash \mathcal{C}_\mathsf{f}$. We describe verifiability checks

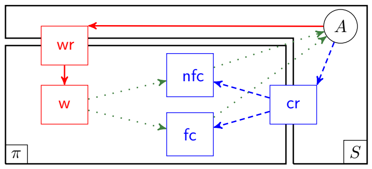


Fig. 3. Abstract representation of the interactions between the BB (π) and an e-voting specification (S) using dedicated state variables (in squares). Write actions for an agent A are depicted with red, solid arrows and read actions with blue, dashed arrows. Green dotted arrows denote interactions that we abstract away: respectively the construction of BB contents from writes (on the left) and the sending of BB contents to readers (on the right).

further in Section III-A. The check requests recorded in cr are of the form $(C, \overline{x}, a)$ and denote that an agent $a \in \mathcal{A}$ wants to read the BB, obtain some B, and check $C(\mathrm{B}, \overline{x})$. With this, we can formalize the e-voting specifications explained above.

Definition 2 (S)**.** *An* e-voting specification S *is a specification whose state variables contain:* cr : $\mathcal{P}(\mathcal{C} \times \mathcal{M}^* \times \mathcal{A})$, wr : $\mathcal{P}(\mathcal{W})$, *and* agents : $\mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A})$. *Moreover,* cr *and* wr *are initially empty and monotonically increasing; for instance, for* cr, $\forall(\sigma.s.\sigma'.s') \in \mathsf{Be}(S)$, $s.\mathsf{cr} \subseteq s'.\mathsf{cr}$.

We define BB specifications similarly. The state variables fc and nfc record checks of the form $(C, \overline{x}, a, \mathrm{B})$, where B is the content that the BB produced for some check request $(C, \overline{x}, a)$ in cr respectively through Read-final and Read-nonFinal.

Definition 3 (π)**.** *A BB specification* π *is a specification containing the state variables* cr *and* wr *(see Definition 2),* fc : $\mathcal{P}(\mathcal{C} \times \mathcal{M}^* \times \mathcal{A} \times \mathcal{W})$, nfc : $\mathcal{P}(\mathcal{C}_\mathsf{nf} \times \mathcal{M}^* \times \mathcal{A} \times \mathcal{W})$, *and* w : $\mathcal{P}(\mathcal{W})$ *such that* cr, wr, fc, nfc, *and* w *are initially empty and monotonically increasing.* π *has write access to neither* cr *nor* wr, *i.e., their values can only be used in guards.*

For a state s, checksA(s) denotes the set of all checks that have been actually processed (answered to) by the BB, namely $\{(C, \overline{x}, a) \mid (C, \overline{x}, a, \mathrm{B}) \in (s.\mathsf{fc} \cup s.\mathsf{nfc})\}$. For a set of checks c (*e.g.,* nfc), we denote by $\mathrm{B}(c)$ the set of all read BB contents, *i.e.,* $\{\mathrm{B} \mid \exists (C, \overline{x}, a, \mathrm{B}) \in c\}$.

Example 1. *Consider the BB specification* $\pi_\mathrm{B}^\mathrm{per}$ *defined below, which provides the functionalities depicted in Figure 2.* $\pi_\mathrm{B}^\mathrm{per}$ *models a* perfect *BB that acts as a shared variable: it returns all previously written messages for all check requests (*Rnf, Rf*) and is updated (*Write*) by all write requests, until a final check is processed and the content is frozen.*

$$
\begin{aligned}
\Sigma &\triangleq (\!|\mathsf{wr}, \mathsf{w}, \mathsf{cr}, \mathsf{fc}, \mathsf{nfc}|\!), \quad \Sigma_0 \equiv \{(\!|\mathsf{wr}, \mathsf{w}, \mathsf{cr}, \mathsf{fc}, \mathsf{nfc} = \emptyset|\!)\} \\
\mathsf{Write}(\mathrm{B}_w) &\equiv \{(s, s') \mid \mathsf{fc} = \emptyset \wedge \mathrm{B}_w \in s.\mathsf{wr} \\
&\qquad \wedge s'.\mathsf{w} = s.\mathsf{w} \cup \{\mathrm{B}_w\}\} \\
\mathsf{Rnf}(C, \overline{x}, a, \mathrm{B}) &\equiv \{(s, s') \mid s.\mathsf{w} = s.\mathsf{wr} \wedge (C, \overline{x}, a) \in s.\mathsf{cr} \\
&\qquad \wedge C \in \mathcal{C}_\mathsf{nf} \wedge \mathrm{B} = \cup_b s.\mathsf{w} \wedge s'.\mathsf{nfc} := s.\mathsf{nfc} \cup \{(C, \overline{x}, a, \mathrm{B})\}\} \\
\mathsf{Rf}(C, \overline{x}, a, \mathrm{B}) &\equiv \{(s, s') \mid s.\mathsf{w} = s.\mathsf{wr} \wedge (C, \overline{x}, a) \in s.\mathsf{cr} \\
&\qquad \wedge \mathrm{B} = \cup_b s.\mathsf{w} \wedge s'.\mathsf{fc} := s.\mathsf{fc} \cup \{(C, \overline{x}, a, \mathrm{B})\}\}
\end{aligned}
$$

Next, consider $\pi_\mathrm{B}^\mathrm{tru}$*, which is as* $\pi_\mathrm{B}^\mathrm{per}$ *but without the guard* $s.\mathsf{w} = s.\mathsf{wr}$ *in the actions* Rnf *and* Rf *and without the guard* $\mathrm{B}_w \in s.\mathsf{wr}$ *in the action* Write. $\pi_\mathrm{B}^\mathrm{tru}$ *models a* trustworthy *BB that has insecure and unreliable channels with writers (*w *and* wr *can be unrelated), but always provides readers with the previous writes and stops accepting new writes once a final check is processed.*

We define a protocol by combining an e-voting and a BB specification, where the checks $\mathsf{nfc} \cup \mathsf{fc}$ must correspond to check requests cr, *i.e.,* the BB only processes requested checks.

Definition 4 (Protocol)**.** *A* protocol *is the union $S \cup \pi$ of an e-voting specification S and a BB specification π. We require, moreover, that (1) π and S only interact through* cr *and* wr, *that is $V(S) \cap V(\pi) \subseteq \{\mathsf{cr}, \mathsf{wr}\}$, and (2) $\forall(\sigma.s) \in \mathsf{Be}(S \cup \pi)$, $\mathsf{checksA}(s) \subseteq s.\mathsf{cr}$. We write $P(S, \pi)$ for $S \cup \pi$ where these two conditions hold.*

Finally, we shall define several BB properties in this paper, all of which can be formalized as predicates.

Definition 5. *Let T be a predicate over behaviors. For a behavior σ, we write $\sigma \vdash T$ when T is satisfied on $\sigma_{|V_b}$ (i.e., σ restricted to the state variables in V_b), where $V_b = \{\mathsf{cr}, \mathsf{wr}, \mathsf{w}, \mathsf{fc}, \mathsf{nfc}\}$. We require that T holds for $\sigma = s_0$, where the values of the record $(s_0)_{|V_b}$ are empty sets. A BB specification π satisfies a predicate T, denoted by $\pi \vdash T$ when, for all S such that $P(S, \pi)$ is a protocol, $\forall \sigma \in \mathsf{Be}(P(S, \pi))$, $\sigma \vdash T$.*

Example 2. *The* written-as-requested *predicate* War *denotes that all write requests were processed before reading, i.e., $\sigma \vdash$* War *when for any prefix of σ of the form $\sigma_0.s.s'$, if $s'.\mathsf{nfc} \cup s'.\mathsf{fc} \neq s.\mathsf{nfc} \cup s.\mathsf{fc}$, then $s.\mathsf{wr} = s.\mathsf{w}$. Also, the* read-as-written *predicate* RaW *denotes that read contents are identical to the previously written contents and that no writes are made once a final check has been processed. That is, $\sigma \vdash$* RaW *when for any prefix $\sigma_0.s.s'$ of σ, the two following conditions hold: (1) $\forall(C, \overline{x}, a, \mathrm{B}) \in ((s'.\mathsf{fc} \cup s'.\mathsf{nfc}) \backslash (s.\mathsf{fc} \cup s.\mathsf{nfc}))$, $B = \cup_b s.\mathsf{w}$ and (2) $s.\mathsf{fc} \neq \emptyset \Rightarrow s'.\mathsf{w} = s.\mathsf{w}$.*

We call a BB that satisfies RaW *trustworthy, as assumed by e.g., Alethea [5]. We call it perfect if it also satisfies* War, *as assumed by e.g., Belenios [30] (see Section III-B2a). It is easy to see that $\pi_{\mathrm{B}}^{\mathrm{per}}$ (respectively $\pi_{\mathrm{B}}^{\mathrm{tru}}$) from Example 1 is a* perfect *(respectively* trustworthy*) BB.*

III. Verifiability and the BB

Verifiability requires that voters and auditors can verify the election's integrity by performing checks on data that is usually stored on the BB. Hence, verifiability critically relies on the BB's properties. We investigate now the relationship between verifiability and BBs.

A. Defining Verifiability for Malicious BBs

We formally define verifiability in our framework. First, we explain that most prior verifiability definitions assumed a *trustworthy* or even a *perfect* BB and we formalize these in our framework. Then, we generalize verifiability by relaxing the restrictions on the BBs found in prior works, and allowing BBs to be malicious.

The Big Picture. Verifiability enables voters and auditors to detect any malicious behavior by the election authorities. The core property is *end-to-end verifiability*, which states that the election's result has been correctly computed based on all eligible voters' votes [2], [10]. It is common to divide this into sub-properties targeting individual steps of the election process [10], [31], [32]. For instance, *Individual Verifiability* (IV) states that when a voter checks that his ballot is in the list of recorded ballots (on the BB), then his ballot is indeed recorded correctly and will be considered in the tallying process. *Universal Verifiability* (UV) states that when auditors or voters verify checks on the end result (on the BB), such as verifying given ZKPs, then the election's result was correctly computed from the recorded ballots. Finally, *Eligibility Verifiability* (EV) ensures that the election's result is only computed from eligible voters' votes and contains at most one vote from each voter.

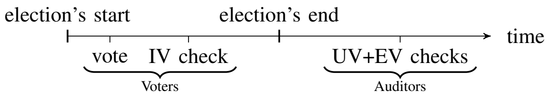


Fig. 4. Election timeline for which the verifiability checks satisfy *Vote&Go*.

More generally, verifiability states that, when the verifiability checks hold for a given execution, then this execution, along with the corresponding final BB content (including the final outcome), meet some *verifiability goal* [2]. These goals can be *quantitative* or *qualitative*. In our work, we focus on qualitative definitions in the possibilistic setting, where agents may perform verifiability checks and where goals are expressed with respect to the set of agents who perform these checks. However, it should be straightforward to generalize our results to the probabilistic case, *e.g.,* using probabilistic transition systems rather than possibilistic ones; we leave this task as future work.

Usability requirements. For verifiability to be achievable in practice, some usability requirements are often considered. Indeed, if verifiability checks are impractical (*e.g.,* too time-consuming or complex), then few voters will actually perform these checks (correctly). One prominent requirement is *Vote&Go* [33]. It requires that voters can perform the IV checks right after voting and verifiability is thereby not jeopardized by human voters who are no longer active in the election process after voting; see footnote in Section VI-B. A timeline for an election with the different verifiability checks that satisfies *Vote&Go* is depicted in Figure 4. We will ensure that our verifiability definition is expressive enough to capture such requirements, should one wish to consider them.

Verifiability for Honest BBs. To formally define verifiability, we first formalize verifiability checks and goals.

Definition 6. *A verifiability check is a predicate $C : \mathcal{P}(\mathcal{W} \times \mathcal{M}^*) \in \mathcal{C}$. C is* final-only *when $C \in \mathcal{C}_{\mathsf{f}} \subseteq \mathcal{C}$. We require that all verifiability checks that are not final-only are* monotonic *in* B, *i.e.,* $\forall \mathrm{B}, \mathrm{B}', \overline{x}, C(\mathrm{B}, \overline{x}) \wedge \mathrm{B} \sqsubseteq_b \mathrm{B}' \Rightarrow C(\mathrm{B}', \overline{x})$.

Monotonicity may appear restrictive. However, as we argue next, it only excludes verifiability checks that cannot be meaningfully evaluated on non-final contents. Fortunately, these verifiability checks are therefore final-only and need not be monotonic. Monotonicity essentially states that a verifiability check cannot be violated by extending the BB content (*e.g.,* adding more items). The lack of monotonicity thus means that verifiability checks evaluated on non-final contents or on partial contents provide no guarantees about the final BB content containing the election outcome. For instance, the check that at most one ballot has been registered per voter is not monotonic. To provide meaningful guarantees such checks must therefore be evaluated on the final, complete contents only and are thus final-only. Monotonicity is typically met by IV checks, as they express that the BB content contains specific items (like a ballot). UV must usually be checked on the final, full BB content and thus these checks are typically final-only.

A goal's satisfiability may depend not only on the final BB content, but also on which checks have been performed and on additional information about the agents' honesty and intended choices. We thus define a goal as a predicate over the final BB content, the set of all checks that have been responded to by the BB, and the sets of honest and malicious agents and their intended choices. For the latter, we assume the goal can use a partial function that returns the intended choice of a voter.

Definition 7. *A* verifiability goal τ *is a predicate over* $\mathcal{W} \times \mathcal{P}(\mathcal{C} \times \mathcal{M}^* \times \mathcal{A}) \times (\mathcal{A} \times \mathcal{A})$ *that is initially satisfied for* $\mathrm{B}_\perp$*, i.e., the initial BB content.*

We next present a generic verifiability definition that is inspired by [2], where different verifiability notions are cast in the same framework. All the verifiability properties analyzed in [2] are only defined for protocols that assume a *trustworthy* or *perfect* BB (see Section VI-B), such as $\pi_{\mathrm{B}}^{\mathrm{tru}}$ or $\pi_{\mathrm{B}}^{\mathrm{per}}$ from Example 1, which we formally characterize by the RaW predicate. For any protocol $P(S, \pi)$ such that $\pi \vdash \mathsf{RaW}$ and for any $(\sigma.s) \in \mathsf{Be}(P(S, \pi))$, we define the *final BB content in* s as the union of all writes, that is $\mathsf{finalB}(s) = \cup_b s.\mathsf{w}$.

Definition 8 (Verifiability)**.** *A protocol* $P(S, \pi)$ *with* $\pi \vdash \mathsf{RaW}$ *provides* verifiability for a verifiability goal τ *when*

$$\forall(\sigma.s) \in \mathsf{Be}(P(S,\pi)), \bigwedge_{(C,\overline{x},a,\mathrm{B})\in(s.\mathsf{fc}\cup s.\mathsf{nfc})} C(\mathrm{B},\overline{x}) \Rightarrow \tau(\mathsf{finalB}(s), \mathsf{checksA}(s), s.\mathsf{agents}).$$

Note that verifiability relies on the BB in two ways. First, the verifiability checks are performed on data read from the BB. Second, the verifiability goal is evaluated with respect to the final BB content, *i.e.,* the content of the final BB must satisfy some properties. Whereas the read and the final BB contents are well-defined for BB specifications satisfying RaW (through $\mathsf{finalB}(\cdot)$ for the final content), this is not the case for arbitrary, possibly malicious BBs. These BBs can, for example, provide different readers completely different (final) BB contents (*equivocation*), possibly unrelated to previous writes. Thus, the above verifiability definition cannot be used for malicious BBs. Hence we next propose a more generic definition thereof, called verifiability$^+$.

Verifiability for Malicious BBs. Intuitively, even when the BB is under adversarial control, we would like checks performed on the BB contents provided by the malicious BB to guarantee that a goal holds with respect to the final content. First, note that for this to be well-defined, the final BB must be well-defined and unique. That is, we can only define verifiability for BBs that present the same content to all readers using Read-final. We thus define the predicate *final-consistency* (FC for short), which holds for a behavior $\sigma.s$ when $|\mathrm{B}(s.\mathsf{fc})| \leq 1$. (Note that RaW strictly implies FC). Any BB specification π satisfying FC never provides two final check requests with two different BB contents, but it can behave arbitrary otherwise. In particular, it can provide different contents to two readers using Read-nonFinal or provide readers with a final BB that is totally unrelated to the writes ($s.\mathsf{w}$ and $s.\mathsf{wr}$). For specifications satisfying FC, we define the unique final content in a state s, denoted by $\mathsf{finalB}^+(s)$, as either $\mathrm{B}_\perp$ when $s.\mathsf{fc} = \emptyset$ (no one has read the final BB), or B_f, where $\mathrm{B}(s.\mathsf{fc}) = \{\mathrm{B}_f\}$, otherwise (when at least one reader has read the final BB). Given this, we define verifiability$^+$ as a variant of verifiability suitable for any BB that is FC.

Definition 9 (Verifiability$^+$)**.** *A protocol* $P(S, \pi)$ *with* $\pi \vdash \mathsf{FC}$ *provides* verifiability$^+$ for a verifiability goal τ *when*

$$\forall(\sigma.s) \in \mathsf{Be}(P(S,\pi)), \bigwedge_{(C,\overline{x},a,\mathrm{B})\in(s.\mathsf{fc}\cup s.\mathsf{nfc})} C(\mathrm{B},\overline{x}) \Rightarrow \tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents}).$$

B. Practical Attacks with Malicious BBs

We now investigate the security impact of malicious BBs. We present several new, practical attacks on the e-voting systems Civitas [12], Belenios [4], and Helios [13], where an adversary controlling the BB can manipulate the election without being detected. In particular, even though *some* of our attacks *may* leave some evidence, they are not detected by the specified verifiability checks even when the protocols' respective BB auditing mechanisms are used. Hence they will remain unnoticed and verifiability is therefore violated. We stress that our attacks do not require the BB to provide different readers with different election outcomes, as in actual elections, readers may also learn this information on other channels (*e.g.,* TV). Finally, we emphasize that our attacks can be carried out with respect to threat models under which these schemes were claimed to be secure [4], [12], [13], thus refuting these claims. We summarize our attacks, their underlying threat models, and the properties they violate in Figure 5. Moreover, we discuss additional attacks arising from a lack of agreement on initial data (*e.g.,* public keys) and argue why this is an orthogonal issue in Appendix A3c.

Many of our attacks and the insights we gain from them also apply to other schemes. Our attacks demonstrate that the BB in e-voting is often the weakest link for realistic threat scenarios, which has been largely overlooked in the design and analyses of e-voting protocols. For instance, Civitas [12] and Helios [13] explicitly consider a malicious BB but greatly underestimate its impact on security, resulting in security claims that our attacks directly refute. The security proofs for Belenios [30] assume a *perfect* BB, which is too strong an assumption as discussed earlier. In [4], it is then claimed that

	TM: BB	TM: other roles	Violate	Exploited BB weakness
C.1	none	none (hon. tellers)	IV	inconsistent views
C.2	none	none (hon. tellers)	IV, UV	partial BB not on final
C.3	none	tabulation tellers	IV, UV	no unique final
C.4	none	none (hon. tellers)	IV, UV	no unique final
C.5	none	none (hon. tellers)	EV, CR	inconsistent views
B.1	voting server	decryption trustees	IV, UV	no unique final
B.2	voting server	none	IV	partial BB not on final

Fig. 5. Summary of attacks on Civitas [12] (C.1-C.5) and Belenios [4] (B.1-B.2) (also affecting Helios [13]). Each attack violates at least one security claim from [4], [12] under the same Threat Model (TM). We denote by *basic threat model*, the weakest adversary considered in [12] and [4]. The 2nd and 3rd columns denote the threat model required for the attacks. It shows, compared to the basic threat model (none), which additional entities must be malicious (shown in red) and which entities may additionally be honest (hon.). The 4th column denotes the properties violated by the attacks.

the (supposedly secure) BB can be realized by a (possibly) malicious BB together with BB auditing mechanisms. We show that these mechanisms are too weak and thus refute these claims. Recall that verifiability may include some form of IV, UV, and EV. Our attacks fundamentally violate verifiability independently of its specific definition in that they manipulate the election outcome without being detected by verifiability checks. Therefore, we consider verifiability informally here and refer to [12] and [4] for the formal definition of verifiability (and its sub-properties) used in Civitas and Belenios.

1) Civitas: Civitas [12], [34] builds on JCJ [3] and is designed to achieve coercion resistance. This means that a voter cannot prove to the adversary whether or how he voted, even when collaborating with the adversary. The protocol includes a *supervisor*, which manages the BB, *registration tellers (RTs)* that produce anonymous credentials for voters using secret sharing, *tabulation tellers (TTs)* that share the *election's secret key*, and *ballot boxes* that collect the ballots. We explain the protocol next, enumerating its main steps for reference.

At the protocol's setup, (s1) the supervisor publishes the election parameters on the BB and (s2) the RTs produce private anonymous credentials for all voters and post on the BB their public counterpart, *i.e.,* their encryption under the election's public key. To vote, a voter (v1) obtains a private credential from the RTs and (v2) computes a ballot containing: the encrypted vote, the encrypted private credential, and a ZKP of well-formedness. All encryptions are computed under the election's public key. The voter then (v3) sends the ballot to at least one ballot box over an anonymous channel.

Finally, the TTs compute the tally as follows. (t1) They retrieve the ballots from the ballot boxes and eliminate those that are not well-formed or contain duplicate credentials, using Plaintext Equivalence Tests (PETs), (t2) they retrieve the list of authorized (public) credentials (from (s2)) from the BB, (t3) they shuffle the lists of authorized credentials and ballots in a mix net and only keep the ballots whose credential is in the list of authorized credentials (checked by PETs), and (t4) they decrypt the remaining ballots and post the result on the BB, as well as ZKPs showing that they followed the protocol.

It is assumed that a voter trusts his voting platform, at least one of the ballot boxes he sends his ballot to, and at least one RT. Voters can verify that their ballots were correctly recorded by checking that they are contained in the list of ballots stored on the BB and taken as input by the TTs. Under these assumptions, Civitas claims to achieve verifiability [12]. Furthermore, it is claimed that, under the additional assumption that at least one TT is honest, coercion resistance holds.

a) The BB in Civitas: With respect to the BB, it is stated that the BB is an "insert-only" storage realized by writers signing the messages they write to the BB and the BB signing read contents. The BB is managed by the supervisor, who is not assumed to be honest. We thus consider a malicious supervisor and hence a malicious BB. In particular, [12, p.6] explains that the BB can delete messages but that only availability can be attacked this way. We refute this claim by presenting attacks that violate crucial security properties, which are far more critical than availability. Note that the original formal proof [3] (for JCJ) considers an honest BB and is thus too weak to back up the aforementioned security claims.

b) Attack C.1: The information published by the supervisor at Step (s1) includes a list of ciphertexts $C = (c_1, \dots, c_n)$ associated with the choices v_i that voters can select. According to [34, p.46], a voter reads this list from the BB at step (v2) and then computes the ballot by re-encrypting the c_i corresponding to her choice v_i and produces a ZKP proving that the underlying c_i is contained in C. At Step (t1), the TTs also read C from the BB and discard ballots that do not have a valid ZKP with respect to C.

In the attack C.1, when a targeted voter A requests a BB content for casting a vote, a malicious BB can provide A with a BB content containing a tampered list $C' \neq C$. A will then compute and cast a ballot (v2-v3) using C'. The TTs discard A's ballot at step (t1) since the ZKP proof for this ballot cannot be verified against C and also produce ZKPs showing that they followed the protocol. As the ZKPs are correct, the UV checks are verified. Moreover, A's specified IV check holds as A only checks that b_A is in the list of ballots processed by the TTs, which is the case. (The list of discarded ballots remains secret to achieve coercion-resistance.) Therefore, even though no verifiability checks are violated, the announced result does not account for A's ballot. This attack thus violates IV. It can be carried out by a malicious BB (and hence a malicious supervisor) even when all other entities are honest.

c) Attack C.2: When the TTs retrieve the credentials at Step (t2), a malicious BB can provide them with a content from which a selected voter A's public credential has been deleted. The BB does not delete this credential from contents sent in earlier steps, *e.g.,* if A or the RTs retrieve the list of credentials. As a consequence, the TTs discard A's ballot b_A at Step (t3), since b_A has no matching authorized public credential, and produce valid ZKPs showing that they followed the protocol. As for C.1, no verifiability checks are violated, yet, the announced result does not account for A's ballot. This attack violates IV and UV, as a valid recorded ballot is not included in the tally.

d) Attack C.3: It is claimed that verifiability is satisfied even when all TTs are malicious. But in this case there is no honest entity authenticating the set of ballots considered for tallying. Therefore, the TTs and the BB can violate verifiability by showing different final contents to different readers as explained next. The adversary first chooses a final outcome of his choice. When a voter or an auditor requests the BB to perform some check, the adversary includes in the answer a set of ballots that: (i) yield the (fixed) outcome of his choice when tallied, and (ii) contain the reader's ballots if any.[2] From this set of ballots, the adversary computes the tally and also includes in the answer to the reader all required proofs showing that the chosen set of ballots leads to the chosen outcome.

e) Attack C.4: Due to performance issues, ballots are processed in relatively small batches of voters (ca. 100), called *blocks*. Each ballot is bound to a block identifier and all TTs independently compute Steps (t2)–(t5) for each block. If there is no prior agreement on the number and on the identifiers of the blocks, the following attack violates IV and UV. Instead of showing the readers the tallies from all block's outcomes, as it is supposed to, the BB selectively drops the results from some specific blocks and thereby shows the readers a final, global result of its choice. Yet, all per-block UV checks are satisfied. A voter performing an IV check can be provided with a BB content where the voter's block has not been dropped. Therefore, no verifiability check detects this attack.

f) Attack C.5: Public credentials are not bound to block identifiers but are delivered to voters by RTs upon checking the inclusion of the voters in the block. A ballot computed by a voter contains a block identifier but no single entity can extract either this identifier or the voter's identity. This and a malicious BB allow a coerced voter to successfully vote in a different block, which defeats coercion-resistance, a central goal of Civitas, the RTs' per-block authorization mechanism at setup (v1) [34, p.45], and some form of EV, which is not an explicit goal. We describe this attack in detail in Appendix A3. No specified checks or auditing mechanisms detect this attack.

g) BB Reads by Authorities: The attacks C.2 and C.5 require a malicious BB that shows inconsistent contents to different e-voting authorities (registration tellers and TTs). The BB is used here as a broadcast channel between authorities. Even though this is not the use case we focus on in this paper (we focus on BBs used for verifiability) and will not be covered by our FA property, the BB protocol that we will propose in Section V would nevertheless prevent these attacks.

2) Belenios: We now discuss Belenios [4], [35], which builds upon Helios [13]. For space reasons, we focus here on Belenios and explain in Appendix A3a why (variants of) our attacks on Belenios also apply to Helios [13].

Belenios improves Helios by providing voters with credentials to avoid *ballot stuffing*, where the adversary adds illegitimate ballots to effect the election's outcome. These changes aim to achieve security under weaker trust assumptions [6], *i.e.,* when the ballot box is dishonest. Our attacks reveal that Belenios' security still crucially relies on the BB's honesty. This assumption seems as strong as the ballot boxes' honesty in the current implementation, where the BB and the ballot box are managed by the same entity.

[2]Learning a reader's identity could be difficult. However, the adversary may target specific voter platforms, track their accesses with sufficiently high probability, and mount the attack only tampering with these voters' ballots.

The main parties in Belenios are: the *registrar* who creates and delivers the voters' credentials to the voters (private part) and to the BB (public part), the *decryption trustees (DTs)* who collectively compute the shared election's secret key, and the *voting server* who maintains the BB, receives the voters' ballots (in the ballot boxes), and communicates with the DTs. To vote, a voter encrypts her vote under the election's public key, computes a ZKP that the vote is in the allowed set of votes, and signs the ciphertext with her (private) credential. This ballot is sent to the voting server, which adds it to the current BB content. When tallying, the DTs collectively compute the election's outcome from the ballots on the BB as follows: they check the correctness of all ZKPs and the ballots' signatures, they use homomorphic encryption's properties to aggregate all ballots and then decrypt this value, yielding the election's outcome, and they compute ZKPs that prove they followed the protocol, which are all published on the BB.

a) The BB in Belenios: Belenios' security proofs [30] assume a *perfect* BB (as it is modeled as a shared variable), but this assumption is not always explicit and not met by practical deployments. In contrast to this assumption, it is claimed in [4] that, when the voting platform and the registrar are honest, verifiability holds, even when the DTs and the voting server are compromised. Since the BB is maintained by the voting server [4], we shall consider a malicious BB and see that this claim is refuted by the attacks B.1 and B.2. In particular, we show next that the BB auditing mechanism fails to prevent such attacks. [4] also considers a "degraded mode", where a centralized entity implements the registrar, the DTs, and the voting server, and claims that, even when this centralized entity is malicious, IV holds, which our attacks also refute.

When discussing the BB in practice, [4] acknowledges that the current implementation as a web page (delivered by the voting server) yields the requirement for "enough parties [to] monitor [the BB], so that it is consistent." The monitoring tools that are proposed (i) check $\sqsubseteq_b$ between two snapshots and (ii) verify all signatures and ZKPs in the BB's content. However, such BB monitoring and auditing is insufficient. As auditing tools only verify that the successive local views of the BB are append-only, they do not guarantee any agreement on the BB contents obtained by different readers, leaving Belenios exposed to the attacks we describe next.

b) Attack B.1: When the BB and all DTs are malicious, an attack very similar to the attack C.3 violates IV and UV.

c) Attack B.2: We now consider a much weaker and more realistic threat model where a threshold or all of the DTs are honest but the BB is malicious. In this scenario, at most one valid final BB outcome can be produced, as this is authenticated by the DTs. Therefore, all readers that successfully perform UV checks see the same BB content. However, this is not true for the IV checks. Since Belenios only requires voters to check that their ballots are in the ballot box [4], [20], which should be published on the BB, when a voter reads the BB to perform an IV check, the BB may provide her with content that contains her ballot but then drop this ballot when displaying the set of ballots to the DTs or the auditors checking UV. Thus, even though the IV check is verified, the voter's ballot is not considered in the tally, which violates IV. As a countermeasure, the Belenios specification could be modified to mandate that voters must *simultaneously* perform IV and UV checks on the same final BB content. This would greatly weaken the usability[3] of the voters' checks. We suggest using a secure BB instead.

3) Conclusion: Even though Civitas signs BB contents and Belenios allows each reader to check that BB contents are locally only monotonically increasing, these mechanisms do not prevent BB cheating, *e.g.,* by providing different readers different BB contents. We conclude that, as currently specified and deployed, Civitas, Belenios, and Helios all fail to provide verifiability. We recommend that they explicitly consider realistic BB requirements (see Section IV) and use a secure BB protocol like ours (see Section V).

Note also that it has recently been shown [36] that a lack of IV allows an attacker to compromise ballot privacy. Our conclusion that an insecure BB violates IV therefore implies that it also violates privacy. Finally, our work also demonstrates that e-voting protocol specifications should explicitly describe *which* verifiability checks need to be carried out, by *whom*, and *when*. This should also be reflected in the User Interface.

IV. Final-Agreement (FA)

Our attacks demonstrate that it is crucial to consider a realistic BB model when making security claims. In particular, rather than assuming idealized BBs, e-voting protocol designers should assume or use BBs providing requirements that can be met in practice under realistic trust assumptions. We next introduce such a BB requirement that is achievable in practice under weak trust assumptions (as shown in Section V), but nevertheless is sufficient for verifiability.

A. Definition

As explained in Section III-A, verifiability can only be meaningfully defined if there is one well-defined final BB, i.e., the BB must satisfy FC. A BB providing FC also prevents the attacks C.3, C.4, and B.1 from Section III-B. Additionally, we argued in Sections III-A and III-B2c that voters should be able to perform checks right after voting (*Vote&Go*) and should not have to read full BB contents. Thus, they should be able to read non-final, partial BBs. However, the attacks C.2 and B.2 show that a BB that can drop items between non-final reads and final reads has dramatic security consequences. To ensure that checks on intermediate BB contents provide meaningful guarantees, we define a BB requirement stating that, in addition to FC, a BB must ensure that all non-final BB contents shown to readers are included in the final BB content (with respect to $\sqsubseteq_b$). These two requirements together also prevent attacks based on the BB showing inconsistent contents to different readers (see attack C.2). We lift $\sqsubseteq_b$ to sets of BB contents as follows: $\mathrm{Bs} \sqsubseteq_b \mathrm{Bs}'$ when $\forall \mathrm{B} \in \mathrm{Bs}, \exists \mathrm{B}' \in \mathrm{Bs}', \mathrm{B} \sqsubseteq_b \mathrm{B}'$.

Definition 10. Final-agreement *(*FA*) holds for* $\sigma.s$ *when (i)* $\sigma.s \vdash \mathsf{FC}$ *and (ii)* $s.\mathsf{fc} = \emptyset \vee \mathrm{B}(s.\mathsf{nfc}) \sqsubseteq_b \mathrm{B}(s.\mathsf{fc})$.

FA neither provides guarantees with respect to the order of data on the BB, relates the successive BB contents that have been read through different accesses to $\mathsf{Read\text{-}nonFinal}$, nor relates the writes and the reads. Nevertheless, if a BB with FA was used by Helios, Civitas, and Belenios, then all ours attacks would be prevented (except C.2 and C.5; see Section III-B1g).

We discuss next FA in e-voting and refer to Appendix A4 for a presentation of other scenarios for which FA is also suitable.

B. FA *in E-voting*

We show next that any protocol satisfying Definition 8 (verifiability) with a *trustworthy* or *perfect* BB as proven in many prior works, also satisfies Definition 9 (verifiability$^+$) under a malicious BB satisfying FA. Additionally, we prove the converse for a large class of checks and goals: it is impossible to achieve Definition 9 with a BB that does not satisfy FA.

1) FA *is Sufficient for Verifiability:* Recall $\pi_{\mathrm{B}}^{\mathrm{per}}$ and $\pi_{\mathrm{B}}^{\mathrm{tru}}$ from Example 1. These respectively model a BB acting as a shared variable and one that is similar, except that the channels from writers to the BB are insecure and unreliable. We show that FA is a sufficient BB requirement for verifiability by proving that any protocol satisfying verifiability with $\pi_{\mathrm{B}}^{\mathrm{per}}$ or $\pi_{\mathrm{B}}^{\mathrm{tru}}$ also satisfies verifiability with any, possibly malicious BB π, provided that $\pi \vdash \mathsf{FA}$. First, we relate verifiability under $\pi_{\mathrm{B}}^{\mathrm{tru}}$ with verifiability$^+$ under π. Second, we relate verifiability under $\pi_{\mathrm{B}}^{\mathrm{per}}$ with verifiability$^+$ under π with the additional assumptions that (i) writers check that their messages have been received by the BB π (through an inclusion verifiability check), and (ii) write requests are authenticated and, thus, the verifiability checks and goal are unaffected by malicious write requests. We formally define these assumptions and prove the following theorem in Appendix A.

Theorem 1. *Let S be an e-voting specification, τ a verifiability goal, and π be an arbitrary BB specification, which can, in particular, specify a malicious BB. Assume that $P(S, \pi_{\mathrm{B}}^{\mathrm{per}})$, $P(S, \pi_{\mathrm{B}}^{\mathrm{tru}})$, and $P(S, \pi)$ are protocols.*

(1) *When $P(S, \pi_{\mathrm{B}}^{\mathrm{tru}})$ provides* verifiability for τ *and* $\pi \vdash \mathsf{FA}$*, then $P(S, \pi)$ provides* verifiability$^+$ for τ.

(2) *When $P(S, \pi_{\mathrm{B}}^{\mathrm{per}})$ provides* verifiability for τ, $\pi \vdash \mathsf{FA}$, *and $P(S, \pi)$ checks all writes and authenticates write requests, then $P(S, \pi)$ provides* verifiability$^+$ for τ.

In practice, this means that prior results established with respect to a *trustworthy* or a *perfect* BB can directly be lifted to the more realistic setting where the BB is only assumed to provide FA, which in turn can be realized under weak trust assumptions (see Section V). For instance, Belenios [30] assumed a *perfect* BB and Alethea [5] assumed a *trustworthy*

[3]Indeed, this would violate the *Vote&Go*-paradigm introduced in Section III-A. Moreover, it is unrealistic to assume that all voters have the computational power, bandwidth, and memory to perform the full UV checks. For instance, checking the integrity of the ballot box requires downloading and storing more than 400MB of data, even assuming only 20,000 ballots.

BB. The formal definition and security claim of verifiability for Civitas originate from [3] and also assume a *perfect* BB. Hence, our results allow weaker, more realistic trust assumptions than those currently used for existing e-voting schemes.

2) Necessity of FA *for Verifiability:* Next, we explain intuitively in which cases FA is also *necessary* for verifiability$^+$ and we refer to Appendix A2 for more details and a formalization of our assumptions and results.

There are protocols, such as CHvote [37], that do not utilize a BB and for which verifiability relies instead on so-called *verification codes* (see Appendix A2). For such protocols, FA is (obviously) not necessary for verifiability. For those protocols where verifiability relies on a BB, we have already argued that FA (i) is necessary, as otherwise the final BB relevant for defining the goal is not well-defined. A protocol can specify checks that are *critical* in that the goal can only hold when such checks are satisfied, but it might also specify checks that are noncritical, that is the goal can hold even if these checks are violated. For the latter, verifiability can be satisfied even if the BB contents read for these checks do not satisfy any condition. Intuitively, we prove in Appendix A2 that FA (ii) is necessary for critical checks evaluated on partial, non-final BB contents. This implies that FA is necessary for many realistic scenarios, for example for all protocols that specify critical IV checks that can be performed before the election's end.

V. A Protocol for Achieving FA

We now present our BB protocol satisfying FA that could replace existing BBs, which all require stronger trust assumptions. We start by presenting our design rationale.

A. Design Rationale and Generic Protocol

1) The Big Picture: Recall our system model from Section II-A3: there is one proxy peer X and n BB peers $P_1, \ldots, P_n$, of which n_h are honest. We assume that each peer P_i has a private signing key sk_i and all readers know the peers' public verification keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$. In our protocol, each peer locally stores its current BB view. When peers receive write requests with new content (forwarded by X), they perform some validity checks, update their BB view accordingly, and sign the updated content. The validity checks serve to enforce FA locally, *i.e.,* on each peer's local view. The signed contents are then collected by X. When a reader reads the BB, she only accepts the read content when she also receives sufficiently many peers' signatures on this content, according to a threshold we describe next. As we shall see, this threshold is chosen such that FA is enforced globally, *i.e.,* for all readers.

2) BB Peer Role:

a) BB Peers' Policies: We require that each peer updates its BB view with a new content only if the latter extends its previous view. That is, it only updates B to B' if $\mathrm{B} \sqsubseteq_b \mathrm{B}'$. In our protocol, this holds as updates are of the form $\mathrm{B}' = \mathrm{B} \cup_b \mathrm{B}_w$, where B_w is the written content.

The peers must each sign at most one final BB. This is required to achieve FA (i) on their view and to ensure that two readers obtaining a signature from the same honest BB

```
- State: current view $\mathrm{B}$ (initially $\mathrm{B}_\perp$), key $\mathsf{sk}_i$
- Input: new content $\mathrm{B}_w$ (supposedly received by X)
- Output: signed contents $\{\sigma_l\}_{l \in L}$ (to send to X)
If $\mathsf{ps}(\mathrm{B}) = p_f$ then return $\emptyset$
$\mathrm{B} := \mathrm{B} \cup_b \mathrm{B}_w$
If $\mathsf{ps}(\mathrm{B}) \neq p_f$
then $\{\mathrm{B}_l\}_{l \in L} := \mathsf{partial}(\mathrm{B})$; return $\{\mathsf{sign}(\mathrm{B}_l, \mathsf{sk}_i)\}_{l \in L}$
else return $\{\mathsf{sign}(\mathrm{B}_l, \mathsf{sk}_i)\}_{l \in \{\mathsf{f}\}}$
```

Fig. 6. Peers specification (peer P_i)

peer agree on the final, signed content. To enforce this policy, we use the notion of phases from Section II-B2, which denote the different stages when the BB is updated. In particular, we assume that there is a pre-defined, agreed upon *final phase* $p_f \in \mathbb{N}^+$ and require that the readers only accept BB contents whose phase is p_f when using Read-final. To ensure that peers sign at most one BB content with phase p_f, our protocol specifies that, when a peer obtains a final content, it updates its view, and afterwards neither accepts further updates nor signs other contents than his view.

Recall that, when performing non-final checks, readers may only read a partial BB content. For example, to check IV, a voter need not retrieve the full BB content, but can just read his ballot to check that it is contained on the BB. As long as the BB content is not final, peers may sign any partial content B_j of their current view B, *i.e.,* $\mathrm{B}_j \sqsubseteq_b \mathrm{B}$. Since BB updates are monotone (w.r.t. $\sqsubseteq_b$), this locally implies FA(ii).

b) Partial BB Contents: For the above policy on partial contents to be useful in practice, it must be defined what partial BB contents are valid and how to compute them. We introduce an uninterpreted set L whose elements label contents according to a function partial defined next.

Definition 11 (Partial BB Contents)**.** *Let* $\mathsf{partial} : \mathcal{W} \to (\mathsf{L} \mapsto \mathcal{W})$ *be a function that computes from a BB content a set of partial BB contents indexed by a subset of* L*, which we can write as* $\mathsf{partial}(\mathrm{B}) = \{\mathrm{B}_l\}_{l \in L}$ *for some* $L \subseteq \mathsf{L}$*. We assume given a distinguished label* $\mathsf{f} \in \mathsf{L}$ *corresponding to final contents and we assume that* $\forall \mathrm{B} \in \mathcal{W}, \mathsf{partial}(\mathrm{B})(\mathsf{f}) = \mathrm{B}$*. We additionally assume that* $\forall \mathrm{B} \in \mathcal{W}, \mathrm{B}' \in \mathsf{partial}(\mathrm{B})$, $\mathrm{B}' \sqsubseteq_b \mathrm{B}$.

In e-voting, L typically includes the voters' identities and partial returns the content associated with a given identity (*e.g.,* a voter's ballot). Note that the peers must agree on partial for the readers to be able to read partial contents with a reasonable success rate, but FA does not rely on this assumption.

c) BB Peers' Actions: The BB peers' specification is depicted in Figure 6. Under the above policies, a peer updates and signs new BB contents, thereby also singing all partial BBs if the BB is not in the final phase. Based on this specification, we will prove that honest peers locally enforce FA, *i.e.,* FA holds from each peer's perspective.

3) BB Readers' Role: To request the non-final, possibly partial, content labeled by $l \in \mathsf{L}$ at a phase p, a reader sends the pair (p, l) to the proxy X. X retrieves all existing signatures for this phase and label and sends them back to the reader. The reader then tries to find among the latter at least γ many signatures of peers that are all valid and that all sign the same content B_k whose phase is p. If this succeeds, the BB content B_k is considered to be successfully read and can be used to evaluate some verifiability check. Reading a final BB is similar, except that readers additionally check that $p = p_f$.

4) Threshold γ: We could require that readers only accept BB contents when they obtain *all* peers' signatures on this content, *i.e.,* choosing $\gamma := n$. However, successful reads would then only be possible if all peers are online, respond, and no response is lost on the insecure network. In practice, these availability assumptions are likely too strong.

Instead, we give a tight lower bound on γ for FA to be satisfied, which depends on the number n_h of honest peers: $\gamma > n - \frac{n_h}{2}$. Intuitively, this bound ensures that when two readers each read a BB content, the two underlying sets of peers who signed the contents share at least one honest peer. Since this honest peer enforces FA locally, we deduce (with machine-checked proofs) in Section V-D that FA holds globally for all readers. We show that the bound is tight, *i.e.,* FA is violated otherwise, in Appendix B2. We thus fix $\gamma := \lfloor n - \frac{n_h}{2} + 1 \rfloor$.

B. Putting Everything Together

The proxy X receives write requests from writers and forwards them to peers. Upon receiving new content, the peers produce signatures thereof as specified in Section V-A2 and Figure 6. These signatures are collected by X.

At any time, a reader can make a read request by sending to X the requested phase p and label $l \in L$. X then sends all (previously) collected signatures for that phase and label to the reader, who processes them as specified in Section V-A3. We give a more detailed specification of the protocols for writing on and reading from the BB in Appendix B.

1) Availability and Scalability: We focus on providing a simple protocol that achieves the necessary requirements for security, *i.e.,* verifiability, which were not met by prior BB protocols (see Section VI). Our protocol may, however, not meet non security-related requirements, such as availability or scalability. We now discuss how our protocol could be combined with existing mechanisms to address these.

a) Availability: We provided a tight bound γ that minimizes the number of peers that must be available. However, if too few peers agree on the content to sign, for example when X sends them different contents B_w, then the readers cannot obtain sufficiently many matching signatures and thus cannot complete read requests. To improve the overall availability of the BB, a BFT protocol could be used by peers to agree on the new content, prior to signing it. For example, similarly to [14], X could be omitted and the writers be connected with all peers who could run a BFT protocol at each writing request to agree on a BB content before signing it. This would not impact our security analysis as we already consider arbitrary write requests, which cover the outcome of any BFT protocol.

b) Scalability: Each peer must produce a large number of signatures that X and readers must fetch and store. Such a set of signatures could be efficiently computed and represented compactly using standard data structures such as hash-trees. In particular if the number of peers is very large (thousands), the decentralized witness cosigning technique [38] could enhance our minimal signing protocol.

C. Using our BB for E-voting

Our protocol could, for example, be instantiated for e-voting as follows. The readers could be instantiated by voters and auditors, the BB peers by independent parties, such as political parties and NGOs, and the proxy peer by a (possibly replicated) e-voting web-server as it need not be trusted. The partial BBs could consist of BBs containing only information associated with one voter, labeled by the voter's identity, and of the final BB, labeled by the distinguished label f. Authorities could publish the received ballots right after their reception and voters could read the partial BB containing their ballot and perform an IV check. Also, at the election's end, auditors can read the final BB and perform all required UV checks.

1) Consequences for Verifiability: As FA requires trusting some peers and verifiability requires FA (Section IV-B2), we can conclude that *verifiability requires trust*. Looking at the bigger picture, this is at odds with some prior works' claims that verifiability can be achieved with no trust assumptions at all [39] or no trust assumptions with respect to the BB [4], [6], [7].

Due to the trust assumptions required for FA, another interesting insight is that *UV checks can be outsourced to the BB peers*. That is, even if voters *could* perform UV checks themselves, they would need to trust some external entities such as BB peers. Therefore, as any entity can carry out UV checks, it is possible to leave these checks to the peers without requiring trust assumptions beyond those needed anyway. The same holds for EV checks. Our analysis supports this conclusion as we are explicit about the BB requirements and the necessary trust assumptions. Delegating the UV checks also has the practical advantage that voters need not download the large amount of data required for these checks (see Section III-B2c). This enhances flexibility in how the voters access the BB, *e.g.,* it enables the use of specialized trusted devices as discussed next.

2) Practical Considerations: In Appendix B3 we discuss practical considerations. In particular, voters must trust their machines in practice to achieve any guarantees and we explain that our protocol can be used in settings where this trust is put in *specialized devices* [5], [21], [26]–[28], which have limited capabilities and connectivity. Moreover, our BB peers are *distributed servers that must be online* during the election process. We explain that this assumption might be impractical for low-stake elections and describe the resulting trade-offs.

D. Security Analysis

We use TLA+ [40] to specify our protocol and FA and formally establish that the former satisfies the latter. TLA+ has a rich specification language [29] based on Temporal Logic of Actions (TLA) and Zermelo-Fraenkel set theory with choice (ZFC) that we use to encode our specifications. Our protocol model is as generic as possible and we make few assumptions about the adversary. Namely, the adversary is only forbidden to forge the n_h honest peers' signatures but can sign any content for the other peers, block messages, choose the written contents, and control X. For the sake of generality, we make strictly fewer assumptions about $\mathcal{W}$, $\sqsubseteq_b$, and $\mathsf{partial}$ than presented in this paper. Using the embedded proof system TLAPS [41], we prove that our protocol satisfies FA. We first establish key invariants of our protocol: namely that FA is locally enforced by honest peers and that any read content is associated with at least γ signatures. We prove the property stated in Section V-A4 and establish that FA is an invariant of our protocol. All our specifications and machine-checked proofs (ca. 800 LOC in total) can be found at [1].

VI. Related Work

There has been extensive prior work on voting and BB designs in the broad sense, e.g., consensus protocols, distributed ledgers, etc. Here we focus on the closest related work and refer the interested reader to Appendix C for further details.

A. BB Realizations

[14], [42] designed for the poll-site voting scheme *vVote* [24], [43], is a distributed BB protocol, later improved in [25]. The main differences to our work are (1) their security goals, which do not explicitly include agreement among the readers, and thus do not imply FA, and (2) the consensus mechanism that is leveraged, namely a BFT protocol that requires strictly stronger trust and system assumptions than our protocol. Their requirements are not formally related to verifiability in e-voting and are too weak for verifiability as there is no agreement on the final content. The protocol is shown [14], [25] to meet its security goals when more than two-thirds of the BB peers are honest *and available*. In contrast, our protocol meets its security goal with strictly weaker trust assumptions (shown in detail in Appendix C2). We achieve this thanks to a more precise security analysis that identifies which assumptions are required *for security* and that distinguishes peers that are dishonest from peers that are unavailable. This is very rarely done in the BFT literature (with notable exceptions like [44]) as security and liveness are usually considered together. In general, note also that the consensus a BFT protocol would provide is not enough to enforce FA.

Some proposed BBs and e-voting systems are built using distributed ledgers [45]–[49]. Most of them are permissionless ledgers and readers must either (i) run a full node, which requires too strong assumptions regarding the voters' capabilities and trust in their platforms (see Section II-A3 and [50]), or (ii) trust external full nodes. These solutions crucially rely on economic incentives, which are hard to quantify for elections, and are often not decentralized in practice due to *pools* [51]. [52] draws similar conclusions. Other solutions use permissioned ledgers where (i) some nodes establish a consensus on data that can be publicly accessed by all nodes or (ii) all e-voting participants act as full nodes. Whereas (i) boils down to the BFT solutions discussed above, (ii) is impractical as argued above.

Finally, the BB proposed in [18] is used in several e-voting schemes [15]–[17]. However, its stated properties do not include agreement among readers, the protocol does not provide FA, and it thus suffers from all the attacks in Section III-B. The decentralized variant is only proposed for improving robustness and still requires that all peers are trusted.

B. BBs in E-Voting Protocols

Most e-voting protocols state insufficient requirements for the BB. Others state wrongly, or with insufficient precision, how such requirements can be met.

We already discussed Civitas [12], Belenios [4], and Helios [13] in Section III-B. The *JCJ* e-voting protocol [3] does not explain how the BB is realized and just assumes a *universally accessible memory* that all agents can write to in an append-only manner (*perfect* BB). *Alethea* [5] assumes a *trustworthy* BB. *Prêt à Voter* [15], which is not a remote but a poll-site scheme, does not specify an explicit BB and refers to [18] instead (see Section VI-A). Building on [15], *vVote* [24], [43] (also poll-site) makes use of [14] (discussed in Section VI-A) for its *private BB* from which voters cannot read and assumes a *public BB* that is an authenticated public broadcast channel with memory. To realize the latter, *vVote* proposes to use radio or newspapers to broadcast (hashes of) the BB contents. This requires voters to cross-check information from different media and violates *Vote&Go*.[4] *Scantegrity* [54] only states the *append-only* property of the BB and no agreement property with respect to the final BB, which is too weak to entail FA.

Some e-voting protocol analyses [6]–[11], [32], [55] consider a BB where the written contents are controlled by the adversary. Even though some of these works qualify their BB as "malicious" or "dishonest", these BBs are still idealized in that a *single* content is produced and *consistently* provided to all readers at all times, and can thus be modeled as a *trustworthy* BB. By making such strong assumptions about the BB, these prior analyses fall short of capturing realistic, malicious BB behaviors, such as equivocation or content modification over time in between reads, and thus miss our attacks.

[56] proposes an all-purpose, parametric notion of verifiability stating that a *goal* always holds when a *judge* is satisfied. This is instantiated for voting protocols such as Bingo Voting [57] in [56], Helios [13] in [10], and sElect [58]. All verifiability definitions for voting protocols based on [56] that we are aware of, including the instantiations in [10], [56], [58], make the assumption of an idealized BB in the sense explained above (even though the BB is called "dishonest" in [10]).

VII. Conclusion

We propose a BB property (FA) that is sufficiently strong to achieve verifiability in e-voting and sufficiently weak that it can be achieved in practice under weak trust assumptions. We support the latter by proposing a concrete BB protocol and by formally proving that it satisfies FA. Our protocol could be

[4] [53] reports on a user study about vVote in the Victorian state election that reveals that only 13% of the voters used the BB to perform the IV check. This in turn yields a security margin of 95% (chances of cheating detection). Considering that only a fraction of these voters would additionally cross-check the hash (say 25%), this results in worrying security margins (ca. 50%).

deployed in existing e-voting schemes to replace the current insecure BBs that were shown to constitute e-voting's Achilles heel by our attacks. Hence, our work can significantly weaken the required trust assumptions of entire e-voting systems.

Our work raises several interesting follow-up research questions. First, to account for malicious BBs, we focused on the *possibilistic* setting and adapted the verifiability definition from [2]. Yet, our modifications appear generic and we speculate that one can similarly propose a probabilistic definition. Second, our BB protocol requires that independent BB peers are available during elections, which is challenging and costly to deploy in practice. For low-stake elections, where a weaker threat model is suitable and the deployment costs are more critical, a simpler BB protocol may be preferable. We plan to investigate this trade-off between stronger threat assumptions and weaker system assumptions in future work.

References

[1] "Our TLA+ models and proofs," https://drive.google.com/open?id=1SDTTDCUd1p11UwBYHrn4nGTd-TGTQcp7, 2020, accessed: 2020-04-06.

[2] V. Cortier, D. Galindo, R. Küsters, J. Mueller, and T. Truderung, "Sok: Verifiability notions for e-voting protocols," in *Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 779–798.

[3] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant Electronic Elections," in *Towards Trustworthy Elections*. Springer-Verlag, 2010, pp. 37–63.

[4] V. Cortier, P. Gaudry, and S. Glondu, "Belenios: a simple private and verifiable electronic voting system," in *Foundations of Security, Protocols, and Equational Reasoning*. Springer, 2019, pp. 214–238.

[5] D. Basin, S. Radomirovic, and L. Schmid, "Alethea: A Provably Secure Random Sample Voting Protocol," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 283–297.

[6] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, "Election Verifiability for Helios under Weaker Trust Assumptions," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 327–344.

[7] V. Cortier, J. Lallemand, and B. Warinschi, "Fifty Shades of Ballot Privacy: Privacy against a Malicious Board," 2020, https://eprint.iacr.org/2020/127.

[8] R. Küsters, T. Truderung, and A. Vogt, "Verifiability, privacy, and coercion-resistance: New insights from a case study," in *2011 IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 538–553.

[9] B. Smyth, S. Frink, and M. R. Clarkson, "Election Verifiability: Cryptographic Definitions and an Analysis of Helios, Helios-C, and JCJ," Cryptology ePrint Archive, Report 2015/233, 2015, https://eprint.iacr.org/2015/233.

[10] R. Kusters, T. Truderung, and A. Vogt, "Clash attacks on the verifiability of e-voting systems," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 395–409.

[11] V. Cortier, A. Filipiak, and J. Lallemand, "BeleniosVS: Secrecy and Verifiability against a Corrupted Voting Device," in *32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019.

[12] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a Secure Voting System," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 354–368.

[13] B. Adida, "Helios: Web-based Open-audit Voting," in *Conference on Security Symposium*. USENIX Association, 2008, pp. 335–348. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496734

[14] C. Culnane and S. Schneider, "A Peered Bulletin Board for Robust Use in Verifiable Voting Systems," in *2014 IEEE 27th Computer Security Foundations Symposium*, 2014, pp. 169–183.

[15] D. Demirel, M. Henning, J. van de Graaf, P. Y. Ryan, and J. Buchmann, "Prêt à Voter Providing Everlasting Privacy," in *E-Voting and Identify*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 156–175.

[16] P. Y. Ryan, D. Bismark, J. A. Heather, S. A. Schneider, and Z. Xia, "Prêt à Voter: a Voter-Verifiable Voting System," *Transactions on Information Forensics and Security*, vol. 4, no. 4, pp. 662–673, 2009.

[17] P. Y. Ryan, P. B. Rønne, and V. Iovino, "Selene: Voting with transparent verifiability and coercion-mitigation," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 176–192.

[18] J. Heather and D. Lundin, "The Append-Only Web Bulletin Board," in *Formal Aspects in Security and Trust*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 242–256.

[19] https://vote.heliosvoting.org/, 2020, accessed: 2020-04-06.

[20] https://belenios.loria.fr/admin, 2020, accessed: 2020-04-06.

[21] S. Bursuc, C.-C. Drăgan, and S. Kremer, "Private votes on untrusted platforms: models, attacks and provable scheme," in *Proceedings of the 4th IEEE European Symposium on Security and Privacy (EuroS&P'19)*. Stockholm, Sweden: IEEE, Jun. 2019.

[22] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz, "Attested Append-Only Memory: Making Adversaries Stick to Their Word," in *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP 07. New York, NY, USA: Association for Computing Machinery, 2007, p. 189204.

[23] T. Ruffing, A. Kate, and D. Schröder, "Liar, Liar, Coins on Fire! Penalizing Equivocation By Loss of Bitcoins," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS 15. New York, NY, USA: Association for Computing Machinery, 2015, p. 219230.

[24] C. Culnane, P. Y. A. Ryan, S. Schneider, and V. Teague, "vvote: a verifiable voting system (version 4.0)," *CoRR*, vol. abs/1404.6822, 2014. [Online]. Available: http://arxiv.org/abs/1404.6822

[25] A. Kiayias, A. Kuldmaa, H. Lipmaa, J. Siim, and T. Zacharias, "On the Security Properties of e-Voting Bulletin Boards," in *Security and Cryptography for Networks*, 2018, pp. 505–523.

[26] G. S. Grewal, M. D. Ryan, L. Chen, and M. R. Clarkson, "Du-vote: Remote electronic voting with untrusted computers," in *Computer Security Foundations Symposium*. IEEE, 2015, pp. 155–169.

[27] R. Haenni and R. E. Koenig, "Voting over the Internet on an Insecure Platform," in *Design, Development, and Use of Secure Electronic Voting Systems*. IGI Global, 2014, pp. 62–75.

[28] S. Neumann and M. Volkamer, "Civitas and the real world: problems and solutions from a practical point of view," in *International Conference on Availability, Reliability and Security*. IEEE, 2012, pp. 180–185.

[29] D. Cansell and D. Mery, "Tutorial on the event-based B method," https://cel.archives-ouvertes.fr/inria-00092846, 2006.

[30] V. Cortier, C. C. Dragan, F. Dupressoir, and B. Warinschi, "Machine-checked proofs for electronic voting: privacy and verifiability for Belenios," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 298–312.

[31] V. Cortier, F. Eigner, S. Kremer, M. Maffei, and C. Wiedling, "Type-Based Verification of Electronic Voting Protocols," in *Principles of Security and Trust*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 303–323.

[32] S. Kremer, M. Ryan, and B. Smyth, "Election Verifiability in Electronic Voting Protocols," in *Computer Security – ESORICS 2010*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 389–404.

[33] S. Kremer and P. B. Rønne, "To du or not to du: A security analysis of du-vote," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 473–486.

[34] M. R. C. S. C. Andrew and C. Myers, "Civitas: Toward a Secure Voting System," *Computing and Information Science Technical Report TR*, vol. 2081, 2007.

[35] S. Glondu, "Belenios specification," *Version 1.6. http://www.belenios.org/specification.pdf*, 2019.

[36] V. Cortier and J. Lallemand, "Voting: You Can't Have Privacy without Individual Verifiability," in *Conference on Computer and Communications Security*. ACM, 2018, pp. 53–66.

[37] R. Haenni, R. E. Koenig, P. Locher, and E. Dubuis, "CHVote System Specification (Version 3.0)," *IACR Cryptology ePrint Archive*, vol. 2017, p. 325, 2017.

[38] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities "honest or bust" with decentralized witness cosigning," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 526–545.

[39] G. Gallegos-Garcia, V. Iovino, A. Rial, P. B. Roenne, and P. Y. Ryan, "(Universal) Unconditional Verifiability in E-Voting without Trusted Parties," *arXiv preprint arXiv:1610.06343*, 2016.

[40] L. Lamport, "The TLA+ hyperbook," 2015, https://lamport.azurewebsites.net/tla/hyperbook.html.

[41] D. Cousineau, D. Doligez, L. Lamport, S. Merz, D. Ricketts, and H. Vanzetto, "TLA+ proofs," in *International Symposium on Formal Methods*. Springer, 2012, pp. 147–154.
[42] C. Culnane and S. Schneider, "A Peered Bulletin Board for Robust Use in Verifiable Voting Systems," *CoRR*, vol. abs/1401.4151, 2014. [Online]. Available: http://arxiv.org/abs/1401.4151
[43] C. Culnane, P. Y. Ryan, S. A. Schneider, and V. Teague, "vVote: A Verifiable Voting System," *ACM Transactions on Information and System Security (TISSEC)*, vol. 18, no. 1, pp. 3:1–3:30, 2015.
[44] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, "Upright cluster services," in *Symposium on Operating Systems Principles (SIGOPS)*, ser. SOSP '09. Association for Computing Machinery, Oct. 2009, pp. 277–290.
[45] https://followmyvote.com/blockchain-voting-the-end-to-end-process/, 2020, accessed: 2020-04-06.
[46] P. Tarasov and H. Tewari, "Internet Voting Using Zcash," *IACR Cryptology ePrint Archive*, vol. 2017, p. 585, 2017.
[47] B. Yu, J. K. Liu, A. Sakzad, S. Nepal, R. Steinfeld, P. Rimba, and M. H. Au, "Platform-independent secure blockchain-based voting system," in *International Conference on Information Security*. Springer, 2018, pp. 369–386.
[48] Z. Zhao and T.-H. H. Chan, "How to vote privately using bitcoin," in *International Conference on Information and Communications Security*. Springer, 2015, pp. 82–96.
[49] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 357–375.
[50] S. Hauser and R. Haenni, "Modeling a Bulletin Board Service Based on Broadcast Channels with Memory," in *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2019, pp. 232–246.
[51] A. Gervais, G. Karame, S. Capkun, and V. Capkun, "Is Bitcoin a Decentralized Currency?" *IEEE Security Privacy*, vol. 12, no. 3, pp. 54–60, 2014.
[52] S. Heiberg, I. Kubjas, J. Siim, and J. Willemson, "On trade-offs of applying block chains for electronic voting bulletin boards," *E-Vote-ID 2018*, p. 259, 2018.
[53] C. Burton, C. Culnane, and S. Schneider, "Secure and Verifiable Electronic Voting in Practice: the use of vVote in the Victorian State Election," *arXiv preprint arXiv:1504.07098*, 2015.
[54] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora, "Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting," *IEEE Security Privacy*, vol. 6, no. 3, pp. 40–46, May 2008.
[55] V. Cortier and B. Smyth, "Attacking and fixing Helios: An analysis of ballot secrecy," *Journal of Computer Security*, vol. 21, no. 1, pp. 89–148, 2013.
[56] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *17th ACM conference on Computer and communications security*. ACM, 2010, pp. 526–535.
[57] J.-M. Bohli, J. Müller-Quade, and S. Röhrich, "Bingo Voting: Secure and Coercion-Free Voting Using a Trusted Random Number Generator," in *E-Voting and Identity*, A. Alkassar and M. Volkamer, Eds. Springer Berlin Heidelberg, 2007, pp. 111–124.
[58] R. Ksters, J. Mller, E. Scapin, and T. Truderung, "sElect: A Lightweight Verifiable Remote Voting System," in *29th IEEE Computer Security Foundations Symposium CSF 2016*, 2016, pp. 341–354.
[59] D. Basin, S. Radomirovic, and M. Schlaepfer, "A Complete Characterization of Secure Human-Server Communication," in *Computer Security Foundations Symposium (CSF)*. IEEE, 2015, pp. 199–213.
[60] R. Focardi, F. L. Luccio, and H. A. Wahsheh, "Usable cryptographic QR codes," in *2018 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2018, pp. 1664–1669.

Appendix

Acknowledgments

We would like to thank Stephan Merz for his highly valuable help regarding TLA+ specifications and proofs.

A. *Verifiability,* FA*, Attacks*

1) FA *is Sufficient for Verifiability:* This section is dedicated to the proof of Theorem 1.

Lemma 1. *Let S be an e-voting specification, τ a verifiability goal, and π be an arbitrary BB specification, which can in particular specify a malicious BB. We assume that $P(S, \pi_{\mathrm{B}}^{\mathrm{tru}})$ and $P(S,\pi)$ are protocols. When $P(S, \pi_{\mathrm{B}}^{\mathrm{tru}})$ provides* verifiability *for τ and $\pi \vdash \mathsf{FA}$, then $P(S,\pi)$ provides* verifiability$^+$ *for τ.*

Proof. Let $\sigma.s$ be a behavior of $P(S,\pi)$ such that $\bigwedge_{(C,\overline{x},a,\mathrm{B})\in(s.\mathsf{fc}\cup s.\mathsf{nfc})} C(\mathrm{B},\overline{x})$. Note that since $\pi \vdash \mathsf{FA}$ and $P(S,\pi)$ is a protocol, $\sigma.s$ satisfies FA and thus FC too. We establish $\tau(\mathsf{finalB}^+(\sigma), \mathsf{checksA}(s), s.\mathsf{agents})$.

Since $\sigma.s \vdash \mathsf{FA}$, either (i) $\mathrm{B}(s.\mathsf{fc}) = \emptyset$ or (ii) $\mathrm{B}(s.\mathsf{fc}) = \{B_f\}$. In case (i), $\mathsf{finalB}^+(s) = \mathrm{B}_\perp$ by the definition of finalB^+ and thus $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ holds by the definition of τ. We next establish $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ for case (ii) (where $\mathsf{finalB}^+(s) = B_f$). The outline of the proof is as follows: (1) We first prove that all checks performed in $\sigma.s$ also hold on the final content $\mathsf{finalB}^+(s)$. (2) We build a behavior $\sigma_f.s_f \in \mathsf{Be}(P(S, \pi_{\mathrm{B}}^{\mathrm{tru}}))$, similar to $\sigma.s$, where checks are performed on final contents only and such that $\mathsf{finalB}^+(s) = \mathsf{finalB}(s_f)$. (3) We prove that the verifiability hypothesis together with (1) and (2) imply $\tau(\mathsf{finalB}(s_f), \mathsf{checksA}(s_f), s_f.\mathsf{agents})$. (4) We relate satisfaction of τ before and after the transformation (2) to conclude.

(1) Since $\pi \vdash \mathsf{FA}$ and $P(S,\pi)$ is a protocol, $\mathrm{B}(s.\mathsf{nfc}) \sqsubseteq_b \mathrm{B}(s.\mathsf{fc}) = \{B_f\}$. Since π is a BB specification, one has that $\forall (C,\overline{x},a,\mathrm{B}) \in s.\mathsf{nfc}$, $C \in \mathcal{C}_{\mathsf{nf}}$, *i.e.,* C is non-final. By the monotonicity of non-final verifiability checks, $\forall (C,\overline{x},a,\mathrm{B}) \in s.\mathsf{nfc}$, $C(\mathrm{B}_f,\overline{x})$ holds since $\mathrm{B} \sqsubseteq_b \mathrm{B}_f$ and $C(\mathrm{B},\overline{x})$ holds. We have also established above that $\mathrm{B}(s.\mathsf{fc}) = \{\mathrm{B}_f\}$. Thus, $\forall (C,\overline{x},a,\mathrm{B}) \in s.\mathsf{fc}$, $C(\mathrm{B}_f,\overline{x})$ holds since $\mathrm{B} = \mathrm{B}_f$ and $C(\mathrm{B},\overline{x})$ holds. Therefore, $\bigwedge_{(C,\overline{x},a,\mathrm{B})\in(s.\mathsf{fc}\cup s.\mathsf{nfc})} C(\mathrm{B}_f,\overline{x})$, where $\mathrm{B}_f = \mathsf{finalB}^+(s)$.

(2) We prove that there is a behavior $\sigma_f.s_f \in \mathsf{Be}(P(S, \pi_{\mathrm{B}}^{\mathrm{tru}}))$ where:

- (a) $\mathsf{finalB}(s_f) = \mathsf{finalB}^+(s)$,
- (b) $\forall (C,\overline{x},a,\mathrm{B}) \in (s_f.\mathsf{fc} \cup s_f.\mathsf{nfc})$, $\mathrm{B} = \mathsf{finalB}^+(s)$,
- (c) $\bigwedge_{(C,\overline{x},a,\mathrm{B})\in(s_f.\mathsf{fc}\cup s_f.\mathsf{nfc})} C(\mathrm{B},\overline{x})$,
- (d) $\mathsf{checksA}(s_f) = \mathsf{checksA}(s)$, and
- (e) $s_f.\mathsf{agents} = s.\mathsf{agents}$.

We build $\sigma_f^0.s_f^0.s_w.\sigma_r \in \mathsf{Be}(P(S, \pi_{\mathrm{B}}^{\mathrm{tru}}))$ from $(\sigma.s) \in \mathsf{Be}(P(S,\pi))$ as follows. First, any action from S is left unchanged and any action from π is replaced by $\mathsf{Skip}()$. We obtain this way a behavior $\sigma_f^0.s_f^0 \in \mathsf{Be}(P(S, \pi_{\mathrm{B}}^{\mathrm{tru}}))$. We now complete this behavior as follows: (1) we trigger the action Write with $\mathrm{B}_w := \mathrm{B}_f$ that yields a state s_w and then (2) for any $(C,\overline{x},a) \in s_w.\mathsf{cr} \cap \mathsf{checksA}(s)$, we trigger a RF action (see Example 1) that adds $(C,\overline{x},a,\cup_b s.\mathsf{w})$ to the state variable $s'.\mathsf{fc}$, where s' is the current state. Note that for any such state s', $s'.\mathsf{w} = s_w.\mathsf{w} = \{\mathrm{B}_f\}$ and $\cup_b s_w.\mathsf{w} = \mathrm{B}_f$. Since all guards are satisfied, we obtain $\sigma_f^0.s_f^0.s_w.\sigma_r \in \mathsf{Be}(P(S, \pi_{\mathrm{B}}^{\mathrm{tru}}))$. Let s_f be the last state of σ_r. We now establish $(\dagger)$: $s_f.\mathsf{fc} = \{(C,\overline{x},a,\mathrm{B}_f) \mid (C,\overline{x},a) \in \mathsf{checksA}(s)\}$. By the definition of a protocol, one has that $\mathsf{checksA}(s) \subseteq s.\mathsf{cr}$. By construction, $s.\mathsf{cr} = s_w.\mathsf{cr}$ and thus $s_w.\mathsf{cr} \cap \mathsf{checksA}(s) = \mathsf{checksA}(s)$. Finally, $\dagger$ follows from the construction of σ_r.

(†) entails (b) as s_f.nfc is empty by construction, and (d). We have that $s_f.\mathsf{w} = s_w.\mathsf{w} = \{\mathrm{B}_f\}$ and thus $\mathsf{finalB}(s_f) = \mathrm{B}_f$, hence (a). (c) follows from (†) and (1). (e) is by construction as the actions triggered by S remained unchanged.

(3) By (2), (2)(c) and by hypothesis, it holds that $\tau(\mathsf{finalB}(s_f), \mathsf{checksA}(s_f), s_f.\mathsf{agents})$.

(4) The equations (2)(a), (2)(d), and (2)(e) imply $\tau(\mathsf{finalB}(s_f), \mathsf{checksA}(s_f), s_f.\mathsf{agents}) \implies$
$\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$.
This and (3) entail $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ concluding the proof. □

Lemma 2, corresponding to the second part of Theorem 1, relies on extra assumptions, which we define next. We first define a predicate WrF that requires FC and that all write requests are included in the (unique) final BB content.

Definition 12. *WrF is such that $\sigma.s \vdash$ WrF if and only if* $(\sigma.s \vdash$ *FC and* $s.\mathsf{fc} = \emptyset \vee (\forall \mathrm{B}_x \in s.\mathsf{wr}, \mathrm{B}_x \sqsubseteq_b \mathsf{finalB}^+(s)))$.

Definition 13. *Let $P(S, \pi)$ be a protocol. Let $C_{\text{in}}(\mathrm{B}, \mathrm{B}_x)$ be the non-final, monotonic verifiability check that holds if and only if* $\mathrm{B}_x \sqsubseteq_b \mathrm{B}$. *We say that $P(S, \pi)$* checks all writes *when the two following conditions hold:*

1) $\forall(\sigma.s) \in \mathsf{Be}(P(S,\pi)), \mathrm{B}_w \in s.\mathsf{wr}, \exists a \in \mathcal{A}$, $(C_{\text{in}}, \mathrm{B}_w, a) \in s.\mathsf{cr}$ *and*
2) $\forall(\sigma.s) \in \mathsf{Be}(P(S,\pi)), s_0 \in \sigma, (C_{\text{in}}, \mathrm{B}_w, a) \in (s_0.\mathsf{cr})$, $s.\mathsf{fc} \neq \emptyset \Rightarrow \exists \mathrm{B} \in \mathcal{W}$, $(C_{\text{in}}, \mathrm{B}_w, a, \mathrm{B}) \in (s.\mathsf{nfc} \cup s.\mathsf{fc})$.

Intuitively, 1) holds when each write request is followed by a check of inclusion in the current BB and 2) implies that such checks are processed by the BB, at least after the first final read. These properties can be enforced by a BB providing the BB content B_x, acting as a receipt that the write request has been taken into account. Intuitively, write actions must then be considered completed only when the corresponding verifiability check of inclusion has been completed. Writers are expected to abort and complain if such a receipt is not delivered.

We now show that for a protocol that satisfies FA, checking all writes enforces WrF.

Property 1. *Let $P(S, \pi)$ be a protocol that checks all writes and that satisfies FA. For any behavior* $(\sigma.s) \in \mathsf{Be}(P(S,\pi))$, *if* $\bigwedge_{(C,\overline{x},a,\mathrm{B}) \in (s.\mathsf{fc} \cup s.\mathsf{nfc})} C(\mathrm{B}, \overline{x})$, *then $\sigma.s \vdash$ WrF.*

Proof. Note that since $\sigma.s \vdash$ FA, $\sigma.s \vdash$ FC. Thus, if $s.\mathsf{fc} = \emptyset$, then $\sigma.s \vdash$ WrF holds. Otherwise, as $\sigma.s \vdash$ FC, there exists some $\mathrm{B}_f \in \mathcal{W}$ such that $\mathsf{finalB}^+ = \mathrm{B}_f$. Let $\mathrm{B}_w \in \mathsf{wr}$. By hypothesis, there exists $a \in \mathcal{A}$ such that $(C_{\text{in}}, \mathrm{B}_w, a) \in s.\mathsf{cr}$. Since $\mathsf{fc} \neq \emptyset$, there exists some $\mathrm{B} \in \mathcal{W}$ such that $(C_{\text{in}}, \mathrm{B}_w, a, \mathrm{B}) \in (s.\mathsf{nfc} \cup s.\mathsf{fc})$. Since $\bigwedge_{(C,\overline{x},a,\mathrm{B}) \in (s.\mathsf{fc} \cup s.\mathsf{nfc})} C(\mathrm{B}, \overline{x})$, then $C_{\text{in}}(\mathrm{B}, \mathrm{B}_w)$ and thus $\mathrm{B}_w \sqsubseteq_b \mathrm{B}$. By FA, one has that $\mathrm{B} \sqsubseteq_b \mathrm{B}_f$. By the transitivity of $\sqsubseteq_b$, we conclude that $\mathrm{B}_w \sqsubseteq_b \mathrm{B}_f = \mathsf{finalB}^+(s)$. □

Definition 14. *Let $P(S, \pi)$ be a protocol and τ a verifiability goal. We say that $P(S, \pi)$ and τ* authenticate write requests *when τ and verifiability checks in S are insensitive to BB contents that have been maliciously extended, i.e., contents that are not included in the union of all write requests. Formally, this holds when for any behavior $\sigma.s \in \mathsf{Be}(P(S,\pi))$ and BB content $\mathrm{B} \in \mathrm{B}(s.\mathsf{fc} \cup s.\mathsf{nfc})$, the following conditions hold:*

(1) $\tau(\mathrm{B} \cap_b (\cup_b s.\mathsf{wr}), \mathsf{checkA}(s), s.\mathsf{agents}) \Rightarrow$
$\tau(\mathrm{B}, \mathsf{checkA}(s), s.\mathsf{agents})$ *(verifiability cannot be violated by adding malicious data; i.e., not in $\cup_b s.\mathsf{wr}$),*

(2) $\forall(C, \overline{x}, a) \in s.\mathsf{cr}, C(\mathrm{B}, \overline{x}) \Rightarrow C(\mathrm{B} \cap_b (\cup_b s.\mathsf{wr}), \overline{x})$ *(checkers cannot be tricked into validating a verifiability check only due to malicious data).*

In practice, S can enforce these two properties by authenticating write requests so that unauthenticated additional data that a malicious BB may add to the BB contents does not impact the verifiability checks and the verifiability goal. That is τ is stable under the addition of unauthenticated data (*i.e.,* not in $\cup_b s.\mathsf{wr}$) and verifiability checks are stable under the removal of unauthenticated data.

For instance, for a UV check, we should ensure that the ballots tallied are authenticated by the writers (*e.g.,* the voters) so that fake but unauthenticated ballots are recognized as such by checkers and are thus irrelevant for the evaluation of verifiability checks and goals. Here, authentication is crucial to prevent *ballot stuffing*. This is exactly the primary improvement of Belenios [4], [6] over Helios [13].

Lemma 2. *Let S be an e-voting specification, τ a verifiability goal, and π be an arbitrary BB specification, which can in particular specify a malicious BB. We assume that $P(S, \pi_{\mathrm{B}}^{\text{per}})$ and $P(S, \pi)$ are protocols. When $P(S, \pi_{\mathrm{B}}^{\text{per}})$ provides* verifiability *for τ, $\pi \vdash$ FA, $P(S, \pi)$ checks all writes, and $P(S, \pi)$ and τ authenticate write requests, then $P(S, \pi)$ provides* verifiability$^+$ for τ.

Proof. Let $\sigma.s$ be a behavior of $P(S, \pi)$ where $\bigwedge_{(C,\overline{x},a,\mathrm{B}) \in (s.\mathsf{fc} \cup s.\mathsf{nfc})} C(\mathrm{B}, \overline{x})$. Since $\pi \vdash$ FA and $P(S, \pi)$ is a protocol, $\sigma.s$ satisfies FA and thus FC too. We establish $\tau(\mathsf{finalB}^+(\sigma), \mathsf{checksA}(s), s.\mathsf{agents})$.

Since $\sigma.s \vdash$ FA, either (i) $\mathrm{B}(s.\mathsf{fc}) = \emptyset$ or (ii) $\mathrm{B}(s.\mathsf{fc}) = \{B_f\}$. In case (i), $\mathsf{finalB}^+(s) = \mathrm{B}_\perp$ by the definition of finalB^+ and thus $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ holds by the definition of a verifiability goal. We next establish $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ for case (ii) (where $\mathsf{finalB}^+(s) = B_f$). We adopt a similar proof structure to Lemma 1, except that $\pi_{\mathrm{B}}^{\text{per}}$ has more guards than $\pi_{\mathrm{B}}^{\text{tru}}$ and the behavior of $P(S, \pi_{\mathrm{B}}^{\text{per}})$ we shall build from $\sigma.s$ in **(3)** will be different.

(1) Since $\pi \vdash$ FA and $P(S, \pi)$ is a protocol, $\mathrm{B}(s.\mathsf{nfc}) \sqsubseteq_b \mathrm{B}(s.\mathsf{fc}) = \{B_f\}$. Since π is a BB specification, then $\forall(C, \overline{x}, a, \mathrm{B}) \in s.\mathsf{nfc}$, $C \in \mathcal{C}_{\mathsf{nf}}$, *i.e.,* C is non-final. By the monotonicity of non-final verifiability checks, $\forall(C, \overline{x}, a, \mathrm{B}) \in s.\mathsf{nfc}$, $C(\mathrm{B}_f, \overline{x})$ since $\mathrm{B} \sqsubseteq_b \mathrm{B}_f$ and $C(\mathrm{B}, \overline{x})$ holds. We have also established above that $\mathrm{B}(s.\mathsf{fc}) = \{\mathrm{B}_f\}$. Thus, $\forall(C, \overline{x}, a, \mathrm{B}) \in s.\mathsf{fc}$, $C(\mathrm{B}_f, \overline{x})$ holds since $\mathrm{B} = \mathrm{B}_f$ and $C(\mathrm{B}, \overline{x})$ holds. Therefore, $\bigwedge_{(C,\overline{x},a,\mathrm{B}) \in (s.\mathsf{fc} \cup s.\mathsf{nfc})} C(\mathrm{B}_f, \overline{x})$ holds, where $\mathrm{B}_f = \mathsf{finalB}^+(s)$.

(2) We now establish that the final BB content B_f contains all write requests. We let $\mathrm{B}_f^i := \cup_b s.\mathsf{wr}$ and we shall establish that $\mathrm{B}_f^i \sqsubseteq_b \mathrm{B}_f$. First note that Property 1 implies $\sigma.s \vdash \mathsf{WrF}$, which yields $\forall \mathrm{B}_w \in s.\mathsf{wr}$, $\mathrm{B}_w \sqsubseteq_b \mathrm{B}_f$ since $s.\mathsf{fc} \neq \emptyset$. By the algebraic properties of a lattice, $\mathrm{B}_f^i \sqsubseteq_b \mathrm{B}_f$, which yields $\mathrm{B}_f \cap_b \mathrm{B}_f^i = \mathrm{B}_f^i$. Since $P(S,\pi)$ and τ authenticate write requests, we have that (1) implies $\bigwedge_{(C,\overline{x},a,\mathrm{B}) \in (s.\mathsf{fc} \cup s.\mathsf{nfc})} C(\mathrm{B}_f^i, \overline{x})$.

(3) We now prove that there is a behavior $\sigma_f.s_f \in \mathsf{Be}(P(S,\pi_{\mathrm{B}}^{\mathrm{per}}))$ for which

- (a) $\mathsf{finalB}(s_f) = \mathrm{B}_f^i$,
- (b) $\forall (C,\overline{x},a,\mathrm{B}) \in (s_f.\mathsf{fc} \cup s_f.\mathsf{nfc})$, $\mathrm{B} = \mathrm{B}_f^i$,
- (c) $\bigwedge_{(C,\overline{x},a,\mathrm{B}) \in (s_f.\mathsf{fc} \cup s_f.\mathsf{nfc})} C(\mathrm{B}, \overline{x})$,
- (d) $\mathsf{checksA}(s_f) = \mathsf{checksA}(s)$, and
- (e) $s_f.\mathsf{agents} = s.\mathsf{agents}$.

We build $\sigma_f^0.s_f^0.\sigma_w.s_w.\sigma_r \in \mathsf{Be}(P(S,\pi_{\mathrm{B}}^{\mathrm{per}}))$ from $(\sigma.s) \in \mathsf{Be}(P(S,\pi))$ as follows. To begin with, any action from S is left unchanged and any action from π is replaced by $\mathsf{Skip}()$. We obtain this way a behavior $\sigma_f^0.s_f^0 \in \mathsf{Be}(P(S,\pi_{\mathrm{B}}^{\mathrm{per}}))$. Next, we complete this behavior as follows. First, for any $\mathrm{B} \in s_f^0.\mathsf{wr} = s.\mathsf{wr}$, we trigger the action Write with $\mathrm{B}_w := \mathrm{B}$ that yields a series of state $\sigma_w.s_w$. Note that $s_w.\mathsf{wr} = s_w.\mathsf{w}$ and $s_w.\mathsf{wr} = s_f^0.\mathsf{wr} = s.\mathsf{wr}$. Second, for any $(C,\overline{x},a) \in s_w.\mathsf{cr} \cap \mathsf{checksA}(s)$, we trigger an RF action (see Example 1) that adds $(C,\overline{x},a,\cup_b s'.\mathsf{w})$ to the state variable $s'.\mathsf{fc}$, where s' is the current state. Note that for any such state s', $s'.\mathsf{w} = s'.\mathsf{wr} = s_w.\mathsf{w} = s.\mathsf{wr}$. Therefore, $\cup_b s'.\mathsf{w} = \mathrm{B}_f^i$. Since all guards of $\pi_{\mathrm{B}}^{\mathrm{per}}$ hold, we obtain $\sigma_f^0.s_f^0.\sigma_w.s_w.\sigma_r \in \mathsf{Be}(P(S,\pi_{\mathrm{B}}^{\mathrm{per}}))$. Let s_f be the last state of σ_r. Note that $\mathsf{finalB}(s_f) = \cup_b s_f.\mathsf{w} = \mathrm{B}_f^i$ and hence (a) holds. We now establish $(\dagger) : s_f.\mathsf{fc} = \{(C,\overline{x},a,\mathrm{B}_f^i) \mid (C,\overline{x},a) \in \mathsf{checksA}(s)\}$. By the definition of a protocol, $\mathsf{checksA}(s) \subseteq s.\mathsf{cr}$. By construction, $s.\mathsf{cr} = s_w.\mathsf{cr}$ and thus $s_w.\mathsf{cr} \cap \mathsf{checksA}(s) = \mathsf{checksA}(s)$. Finally, $\dagger$ follows from the construction of σ_r and the aforementioned invariants. $(\dagger)$ entails (b) and (d). (c) follows from $(\dagger)$ and (2). (e) is by construction as the actions triggered by S remained unchanged.

(4) By (3)(c) and by hypothesis, it holds that $\tau(\mathsf{finalB}(s_f), \mathsf{checksA}(s_f), s_f.\mathsf{agents})$.

(5) Finally, (3)(a,d,e), (4), and $\mathrm{B}_f \cap_b (\cup_b s.\mathsf{wr}) = \mathrm{B}_f^i$ (2) imply $\tau(\mathsf{finalB}^+(s) \cap (\cup_b s.\mathsf{wr}), \mathsf{checksA}(s), s.\mathsf{agents})$. Since $P(S,\pi)$ and τ authenticate all writes, it then holds that $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ concluding the proof. □

Lemmas 1 and 2 imply Theorem 1.

2) FA is Necessary for Verifiability: In Section IV-B2, we informally argued in which cases FA is *necessary* for verifiability$^+$. We now elaborate on this. Recall that we distinguished *critical* checks, which must be satisfied for the goal to hold, and noncritical checks.

First, we have argued that some e-voting protocols achieve verifiability independently of any BB content. For example, CHvote [37] only relies on *verification codes* to achieve verifiability. Such codes are initially only known to the voters and can be computed for a ballot only if sufficiently many authorities collaborate. Therefore, if a voter receives a valid code for her ballot, she knows that her ballot has been received by the required number of authorities. Thus, when at least one of the involved authorities is trusted, she also knows that her ballot will be counted in the tally. This way, CHvote provides IV without relying on a BB. We thus concentrate in the following on protocols that use a BB to achieve verifiability.

We have argued that FA(i) is necessary, as otherwise the final BB relevant for the stated goal is not well-defined. In practice, without this guarantee, attacks such as C.3, C.4, or B.1 from Section III-B are possible, where each reader requesting a final content is provided with an adversary-chosen content, tailored to this reader to satisfy his checks.

We now turn to the FA(ii) requirement. While the verifiability attacks C.2 and B.2 rely on violations of this requirement, the requirement is not always necessary to achieve verifiability. For instance, if all checks are final-only, this requirement is of no use. However, we argue next that under reasonable assumptions about the verifiability checks and goal, the requirement (ii) is necessary for verifiability to hold. IV checks are typical examples satisfying these assumptions. (Independently of this, note that one can always consider some C and τ that, together, exactly check FA(ii), in which case verifiability implies FA(ii).)

We say that a check is *minimal* in its BB contents, when the totality of the BB content that has been read to evaluate the verifiability check is actually needed for the check to hold. Also, we say that a protocol can *postpone reads* when it can wait until reaching the final phase before the critical checks are performed. We show that FA(ii) is necessary for checks that are critical, minimal, and used in protocols that can postpone reads. The result implies that FA is necessary for many realistic scenarios, for example for all protocols that specify critical IV checks that can be performed at any time after vote casting. We first formalize these assumptions.

We say that B is *minimal* for $C, \overline{x}$ when: $C(\mathrm{B},\overline{x})$ holds but for any $\mathrm{B}' \in \mathcal{W}$, if $\mathrm{B} \not\sqsubseteq_b \mathrm{B}'$, then $C(\mathrm{B}',\overline{x})$ does not hold. For a protocol $P(S,\pi)$, we say that a verifiability check C is *critical for a verifiability goal* τ when there is no execution $(\sigma.s) \in \mathsf{Be}(P(S,\pi))$ and $(C,\overline{x},a,\mathrm{B}) \in (s.\mathsf{nfc} \cup s.\mathsf{fc})$ such that $C(\mathrm{B},\overline{x})$ does not hold but $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ does hold. Finally, we say that a protocol $P(S,\pi)$, such that π satisfies FC, *can postpone reads* when any processed check can always be processed later using the final BB content[5]; formally when for any $(\sigma.s) \in \mathsf{Be}(P(S,\pi))$, $(C,\overline{x},a,\mathrm{B}) \in s.\mathsf{nfc}$, and $\mathrm{B}_f \in \mathrm{B}(s.\mathsf{fc})$, there exists a $\sigma'.s' \in \mathsf{Be}(P(S,\pi))$ such that: $(s'.\mathsf{nfc} \cup s'.\mathsf{fc}) = (s.\mathsf{nfc} \cup s.\mathsf{fc}) \setminus \{(C,\overline{x},a,\mathrm{B})\} \cup \{(C,\overline{x},a,\mathrm{B}_f)\}$, $\mathsf{finalB}^+(s') = \mathsf{finalB}^+(s)$, and $s'.\mathsf{agents} = s.\mathsf{agents}$.

Under these assumptions on some protocol $P(S,\pi)$, verifiability goal τ and check C, we formally prove that if there is an execution $(\sigma.s) \in \mathsf{Be}(P(S,\pi))$ whose processed checks hold and a check $(C,\overline{x},a,\mathrm{B}) \in s.\mathsf{nfc}$ such that FA (ii) is violated for B; *i.e.,* $\mathrm{B} \not\sqsubseteq_b \mathrm{B}_f$ where $\mathrm{B}_f \in \mathrm{B}(s.\mathsf{fc})$), then $P(S,\pi)$ does not provide verifiability for τ.

Theorem 2. *Let $P(S,\pi)$ be a protocol that* can postpone reads *and such that $\pi \vdash \mathsf{FC}$. Let $(\sigma.s) \in \mathsf{Be}(P(s,\pi))$ and $(C,\overline{x},a,\mathrm{B}) \in s.\mathsf{nfc}$ such that $\mathrm{B}(s.\mathsf{fc}) = \{\mathrm{B}_f\}$ and*

[5]In practice, this may be because the initial check request (in cr) originating from an entity (specified in S) has been postponed.

$\mathrm{B} \not\sqsubseteq_b \mathrm{B}_f$*,* B *is minimal for* $C, \overline{x}$*,* C *is critical for* τ*, and* $\bigwedge_{(C',\overline{x'},a',\mathrm{B}') \in (s.\mathsf{fc} \cup s.\mathsf{nfc})} C'(\mathrm{B}', \overline{x'})$ *holds. Then,* $P(S, \pi)$ *does not provide verifiability*$^+$ *for* τ*.*

Proof sketch. Assume given $\sigma.s$, $(C, \overline{x}, a, \mathrm{B})$, and B_f as above. We assume that $P(S, \pi)$ provides verifiability$^+$ for τ; thus by hypothesis (0) $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ holds, and derive a contradiction. Since $\mathrm{B}_f \in \mathrm{B}(s.\mathsf{fc})$ and $P(S, \pi)$ can postpone reads, $(C, \overline{x}, a, \mathrm{B})$ can be postponed, which yields an execution $\sigma'.s' \in \mathsf{Be}(P(S, \pi))$ such that (i) $(s'.\mathsf{nfc} \cup s'.\mathsf{fc}) = (s.\mathsf{nfc} \cup s.\mathsf{fc}) \setminus \{(C, \overline{x}, a, \mathrm{B})\} \cup \{(C, \overline{x}, a, \mathrm{B}_f)\}$, (ii) $\mathsf{finalB}^+(s') = \mathsf{finalB}^+(s)$, and (iii) $s'.\mathsf{agents} = s.\mathsf{agents}$. We now prove that the execution $\sigma'.s'$ contradicts the fact that C is critical for τ. By the minimality of B for $C, \overline{x}$ and by $\mathrm{B} \not\sqsubseteq_b \mathrm{B}_f$, we have that (iv) $C(\mathrm{B}_f, \overline{x})$ does not hold. However, (v) τ holds for $\sigma'.s'$ since we have by (0) that $\tau(\mathsf{finalB}^+(s), \mathsf{checksA}(s), s.\mathsf{agents})$ holds and $\mathsf{finalB}^+(s') = \mathrm{B}_f = \mathsf{finalB}^+(s)$ by (ii), $\mathsf{checksA}(s') = \mathsf{checksA}(s)$ by (i), and $s'.\mathsf{agents} = s.\mathsf{agents}$ by (iii). Therefore, (iv) and (v) contradict the fact that C is critical for τ. □

3) Attacks:

a) Our Attacks on Helios: We explain why our attacks on Belenios from Section III-B2 also apply to Helios [13]. Similarly to Belenios, Helios [13] also realizes the BB by a centralized web-server. Unlike Belenios, Helios additionally proposes that auditors can *re-post* data from the BB so that voters can check that their ballot was considered by the auditors. We next argue why this does not prevent our attacks, in particular Attack B.2, when considering the specification in [13] and taking into account practical considerations.

First, [13] does not specify how many and which auditors a voter must contact, whether auditors also compare their lists among each other, and which parties must be trusted. If, for example, a voter only contacts malicious auditors, she does not get any guarantees about whether her ballot was actually considered in the tally. Also, it is unrealistic to assume that voters contact all auditors. Thus, the specification in [13] is insufficient, i.e., too imprecise, for avoiding our attacks and seems to rely either on unpractical communication efforts (a voter contacts all auditors) or strong trust assumptions (all auditors are trusted).

Moreover, Helios' re-posting violates the *Vote&Go* paradigm (see Section III-A) as it requires all voters to store their ballots, wait for the election's end, and find their ballot in the auditors' posted lists.

b) Continuing attack C.5 from Section III-B1: We assume that all entities proceed on a block-basis and that a malicious BB may redirect public credentials intended for a certain block to another block. Indeed, the specification only specifies that the registration tellers should sign such write requests to the BB, but not in the context of a specific block. A malicious BB and a coerced voter A could thus attack coercion-resistance and EV as follows: when computing her ballot b_A, A binds b_A to an adversary-chosen block identifier b', instead of the block identifier b that is bound to her credential (b is the block where A is registered, according to the registration tellers). A then casts her vote to the ballot boxes in the context of the block b'. The BB then provides the TTs for the block b' with a content where A's public credential has been added. Therefore, A's vote will be counted in block b', although she was registered for b, violating some form of EV. Moreover, by choosing b' appropriately, the adversary can introduce a bias in the result in this block and thus learn A's vote by adapting the attack of [55]. This way, the adversary can determine if coerced voters behaved as requested.

Although this attack leaves some traces, they are currently neither detected by the protocol's verifiability checks nor by the auditing mechanisms. Indeed, because the malicious BB has added A's public credential to the list of credentials for the block b' and because the TTs will sign this list and post it on the BB, this attack leaves evidence. Thus, as a countermeasure, one could mandate the registrars to check whether the list of credentials published for the block b' contains credentials that they have not seen in the context of the block b'. However, we stress that our attack would remain unnoticed in the current protocol and that this countermeasure is only effective if the protocol specification is modified such that it explicitely defines and mandates the use of the corresponding additional verifiability checks.

c) Problems at Setup: We describe an additional attack that is enabled by an insecure BB. In Civitas, a malicious supervisor could, at Step (s1), publish public keys that belong to the adversary rather than to the legitimate tellers. The adversary could then impersonate all tellers and violate verifiability and privacy. A similar attack can also be mounted on Belenios. These attacks boil down to the problem of bootstrapping a PKI, which we consider orthogonal to the problem we address in this paper. In particular, a solution to this problem does not directly solve the BB problem, which is illustrated by the other attacks that are possible even when all public keys are authenticated correctly. In other words, the former needs a static agreement (on the keys) while the BB needs a dynamic agreement (on the evolving contents).

4) Other Scenarios where Final-Agreement is Sufficient: Although our work is primarily motivated by e-voting, there are other scenarios where FA is sufficient and sometimes necessary. First, BBs providing FA can be safely used in *sealed-bid auctions*, where all bidders submit bids without knowing the other bids. For example, the bidders can submit a bid by writing a commitment thereof on the BB and, after a known deadline, write the information to open their commitment, too. By non-final checks, the bidders can verify that their bid was included. FA ensures that such checks still hold on the final BB and that all bidders agree on the auction's final outcome.

Another use case where BBs satisfying FA are useful is for collecting signatures for an (online) *petition* [18]. Usually, some number of signatures must be collected by a deadline, which defines when the BB should be considered final. Everyone can use Write to send their signature to the BB, check using Read-nonFinal that their signature is considered, and check the final result using Read-final. By FA, it is guaranteed that all checked signatures are still included in the final BB and that all readers agree on the final content.

However, in general, FA does not enforce any ordering between the writes and the reads or between successive reads. This excludes using a BB that only satisfies FA for system logs, general auctions, discussion boards, *etc.* Note that even if time stamps were used, the BB could still globally drop selected messages, which is critical in some scenarios, such as for secure logs. In such cases, stronger requirements and thus more costly BBs are needed.

B. Our Bulletin Board Protocol

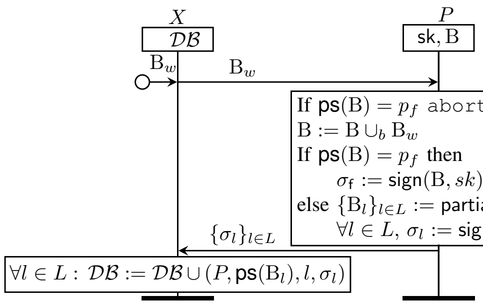
1) Protocol Specification: As explained earlier, the peers sign different (partial) BB contents. We assume that the proxy X stores these signatures in a database $\mathcal{DB}$. Entries in $\mathcal{DB}$ have the form (P, p, l, σ), where σ is peer P's signature on a BB content labeled with $l \in \mathsf{L}$, with phase $p \in \mathbb{N}^+$. The function $\mathsf{getSig}(\mathcal{DB}, p, l)$ retrieves from $\mathcal{DB}$ all σ_i from entries (P_i, p, l, σ_i), i.e., all the stored signatures for the phase p and the label l. Using these functions, we next describe our protocol in terms of three sub-protocols for Write, Read-nonFinal, and Read-final.

BB-Write, depicted in Figure 7, is the sub-protocol for writing to the BB. As the specification is uniform for all peers, we only describe it for one peer P. Upon receiving a new written content, P updates its local view B, computes B's partial contents (B_l) using partial, and signs each of them. How such a batch of signatures can be efficiently computed (*e.g.,* using hash-trees) is left to more concrete system designs. Note that when the local view B is final, P only signs the full content B, labeled $\mathsf{f} \in \mathsf{L}$, thereby enforcing the policy presented in Section V-A2. All signatures produced are then sent to X, who stores them in $\mathcal{DB}$.

Figure 8 depicts the sub-protocol BB-Read-NF for reading a non-final BB content. To request the partial content labeled by $l \in \mathsf{L}$ at phase p, a reader sends the pair (p, l) to the proxy X. X retrieves from $\mathcal{DB}$ all existing signatures for this phase and label, where $I_p^l \subseteq [1, n]$ denotes the peer indices for which such a signatures exists, and X sends them back to the reader. The reader then tries to find a sufficiently large set (of size at least γ) of peers' signatures that are all valid and that all sign the same content B_k whose phase is p. If such a set is found, the BB content B_k is considered to be successfully read and can be used to evaluate some verifiability check.

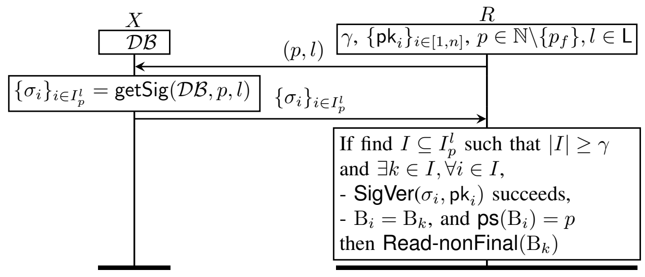
The sub-protocol BB-Read-F for reading a final BB is similar to BB-Read-NF, except that no label l need be sent by the reader in the first message, as only the full BB is signed for the final phase, and X uses $\mathsf{getSig}(\mathcal{DB}, p_f, \mathsf{f})$ to retrieve all signatures on the full, final BB contents.

2) Threshold γ: We claimed in Section V-A4 that the threshold γ in our protocol ensures that for any two readers, the intersection of the sets of signatures that were verified contains one honest peer. We now prove this claim. Let $I = [1, n]$. Let $I_1 \subseteq I$ and $I_2 \subseteq I$ be the sets of peers whose signatures matched and were verified in a read by reader R_1 and R_2, respectively. By our protocol design, it



X
$\mathcal{DB}$
P
sk, B
B_w
B_w
If $\mathsf{ps}(\mathrm{B}) = p_f$ abort
$\mathrm{B} := \mathrm{B} \cup_b \mathrm{B}_w$
If $\mathsf{ps}(\mathrm{B}) = p_f$ then
$\sigma_{\mathsf{f}} := \mathsf{sign}(\mathrm{B}, sk)$, $L := \{\mathsf{f}\}$
else $\{\mathrm{B}_l\}_{l \in L} := \mathsf{partial}(\mathrm{B})$
$\forall l \in L$, $\sigma_l := \mathsf{sign}(\mathrm{B}_l, sk)$
$\{\sigma_l\}_{l \in L}$
$\forall l \in L : \mathcal{DB} := \mathcal{DB} \cup (P, \mathsf{ps}(\mathrm{B}_l), l, \sigma_l)$

Fig. 7. The sub-protocol BB-Write. When the proxy X receives a write request with B_w, it forwards it to all peers, which we only describe for one peer P. Here, sk and B respectively denote P's signing key and P's current view of the BB content. Initially $\mathrm{B} := \mathrm{B}_\perp$. $\mathsf{sign}(m, \mathsf{sk})$ denotes the signature of m under the signing key sk and $\mathcal{DB}$ stores all signatures.



X
$\mathcal{DB}$
R
γ, $\{\mathsf{pk}_i\}_{i \in [1,n]}$, $p \in \mathbb{N} \backslash \{p_f\}$, $l \in \mathsf{L}$
(p, l)
$\{\sigma_i\}_{i \in I_p^l} = \mathsf{getSig}(\mathcal{DB}, p, l)$
$\{\sigma_i\}_{i \in I_p^l}$
If find $I \subseteq I_p^l$ such that $|I| \geq \gamma$
and $\exists k \in I, \forall i \in I$,
- $\mathsf{SigVer}(\sigma_i, \mathsf{pk}_i)$ succeeds,
- $\mathrm{B}_i = \mathrm{B}_k$, and $\mathsf{ps}(\mathrm{B}_i) = p$
then Read-nonFinal(B_k)

Fig. 8. The sub-protocol BB-Read-NF for reading non-final BB contents. $\mathsf{SigVer}(\sigma, \mathsf{pk})$ denotes the verification of the signature σ with the verification key pk. Upon receiving a request for a content labeled l at phase $p \neq p_f$, the proxy X retrieves all peers' signatures for this label and phase using getSig. From these, the reader then tries to find sufficiently many matching signatures.

holds that $|I_1|, |I_2| \geq \gamma$. Assuming γ satisfies $\gamma > n - \frac{n_h}{2}$, we establish that $|I_1 \cap I_2| > n - n_h$; *i.e.,* the intersection is strictly larger than the number of dishonest peers and thus contains an honest peer. It holds that $|I_1| + |I_2| - |I_1 \cap I_2| = |I_1 \cup I_2|$. Since $I_1 \cup I_2 \subseteq I$, one has $|I_1 \cup I_2| \leq n$ and thus $|I_1 \cap I_2| \geq |I_1| + |I_2| - n \geq 2\gamma - n$. The hypothesis $\gamma > n - \frac{n_h}{2}$ then yields $2\gamma - n > n - n_h$. Hence $|I_1 \cap I_2| > n - n_h$. This has been machine-checked as part of our security proof for our protocol in [1].

We now prove that this bound is tight. That is, if $\gamma \leq n - \frac{n_h}{2}$, then we can build I_1, I_2 as above, whose intersection does not include any honest peer. If $n \geq 2\gamma$ then one can choose I_1, I_2 with an empty intersection, which concludes the proof. Otherwise, $n < 2\gamma$. Since $\frac{n_h}{2} \leq n - \gamma$, it follows that $\frac{n_h}{2} < \gamma$ and thus $\lceil \frac{n_h}{2} \rceil \leq \gamma$. Thus, one can choose I_1 of size γ, containing exactly $\lceil \frac{n_h}{2} \rceil$ honest peers. Let $I_h \subseteq I$ be the set of all honest peers. We note that $|I_1 \cap I_h| = \lceil \frac{n_h}{2} \rceil \leq n - \gamma$. Therefore $|I \backslash (I_1 \cap I_h)| \geq \gamma$. One can thus choose $I_2 := I \backslash (I_1 \cap I_h)$, which satisfies $|I_2| \geq \gamma$ and $I_1 \cap I_2 \cap I_h = \emptyset$.

If such I_1, I_2 exist, our protocol does not satisfy FA as X (or the malicious network) can send different contents to the peers in I_1 and I_2. When two readers R_1 and R_2 read the BB, X can send the signatures from the peers in I_i to the reader R_i, for $i \in \{1, 2\}$, which results in R_1 and R_2 accepting inconsistent BB contents, *e.g.,* distinct final contents hence violating FA.

3) Practical Considerations: We provide details for the practical considerations mentioned in Section V-C2.

a) Trusted Devices: In practice, voters are humans who rely on their platforms (*e.g.,* laptops) to read the BB and maybe even to perform some cryptographic checks [59]. Thus, FA, verifiability, and also privacy only hold when the platform is honest. Instead of trusting general purpose platforms, some e-voting protocols [5], [21], [26]–[28] prefer to trust *specialized devices*. As such devices have limited capabilities and network connectivity, their attack surface is smaller and thus trusting them is more realistic than trusting the platforms.

In such a setting, our BB protocol can still be used. The read requests can be entered by the voter on the platform and the proxy's answers can be relayed by the platform to the device. The trusted device can then perform all the required checks, including those from the BB protocol and possibly IV checks, and display the result to the voter.

As the communication channels to the device are limited by assumption, their realization must be considered carefully in terms of usability and feasibility. For instance, requiring voters to copy all BB peer's signatures manually from the platform to the device is not usable. Alternatively, messages can be relayed on specific digital channels, such as by Bluetooth, USB cables, or by the device scanning some QR codes on the platform's display. Whereas these options are more usable than manual copying, the bandwidth is still limited. For example, a QR code can only contain roughly 9 signatures [60] (for 256-bits payloads, *e.g.,* hashed ballots). More generally, fetching the full final BB content as needed for performing UV or EV checks is very likely impractical. However, as we have explained in Section V-C1, the UV and EV checks can be outsourced to external auditors and thus voters only have to use their devices to check IV. Furthermore, our protocol allows readers to read only partial BB contents. The combination of these mechanisms makes the use of semi-online specialized devices practical in our protocol.

b) Online BB Peers: We have argued in Section V-C2 that assuming distributed online servers, the BB peers, might be too strong a system assumption for small-scale elections.

We point out that some e-voting protocols already rely on distributed online servers for realizing other entities in the system. For instance in Civitas [12], the ballot boxes are distributed entities that must be online during the election. For such e-voting protocols, no additional system assumptions and thus extra costs are required to realize our BB protocol.

Other protocols do not currently use such an architecture. For instance, Belenios [4] makes use of distributed peers for tallying and possibly for the *Registrar* which only have to be online and perform computations at specific times. As it takes less effort for the administrators to run such entities than running our BB peers which need to be (almost) always online, our protocol could negatively impact the usability and deployment cost of Belenios and other similar protocols.

While we argued that our assumptions are suitable for large scale, high stake elections, for low stake elections, *e.g.,* the election of board members in a company, strictly weaker adversaries could cover all realistic threat scenarios. In such settings, one may want to prioritize weakening the system assumptions over strengthening the adversary model, as the available or feasible infrastructure is limited. For instance, a malicious but cautious BB could meet FA without relying on distributed online entities. We leave the investigation of such trade-offs between security under strong threat models and less costly architectures as future work.

c) Privacy: In our BB protocol, when voters retrieve a partial content, the proxy peer learns their ballot. However, we stress that this does not mean that the proxy peer learns the ballot's content, *i.e.,* the intended vote. Therefore, *ballot privacy* is not impacted. Moreover, in our adversary model the adversary controls the network and learns this (non critical) information anyway when the ballot is cast. In use cases where this could still be a concern, alternative setups must be used together with an instantiation of our protocol where partial contents contain the information associated with several voters (thus increasing the size of anonymity sets) or where our protocol is used together with techniques such as private information retrieval.

C. Related Work

1) Existing BB Requirements in E-voting: As explained in Section VI, many prior works require the BB to provide the properties of *authenticity*, *append-only*, and/or *availability*. Append-only denotes that items cannot be erased from the BB content over time, which can be defined with respect to one reader's BB view or with respect to many readers' views. In the latter case, append-only enforces some form of agreement on BB contents among readers. This is not the case, however, when append-only is defined with respect to one reader, that is $\mathrm{B}_1 \sqsubseteq_b \mathrm{B}_2$, when a reader reads B_1 and later B_2. Even the combination of *authenticity*, *availability*, and this notion of *append-only* does not entail FA and is thus insufficient to achieve verifiability in many cases (see Section IV-B2). Moreover, all of the attacks presented in Section III-B can still be carried out by a BB satisfying all three of these properties.

In contrast, FA provides weaker guarantees to the readers than a *broadcast channel*. First, FA does not require the readers to agree on *non-final* contents and does not guarantee any order of the data on the BB. Second, most broadcast channel definitions entail *termination*, which is a form of availability requiring that *eventually every BB reader decides on some value for the BB content*. In contrast, FA solely focuses on security (see Section V-B1). We believe that it is important to identify the minimal requirements for security in order to better understand the trade-off between security and availability.

2) Threshold Comparison to BFT algorithms: We next prove the claim from Section VI-A that our protocol requires strictly weaker bounds than those by previous BBs based on BFT algorithms [14], [25].

A BB peer may be available or not, and orthogonally, may be honest or not. We denote by n_a the number of available peers and by $n_{h,a}$ the number of honest and available peers. Also, let γ be the number of matching signatures needed for a reader to accept a new content. Note that γ can be greater than n_h since n_h only counts the amount of peers that *must* be honest (for all readers), and not the peers that actually behave honestly, which can be more numerous and can depend on the reader. Our protocol requires (i) $n_h > 2(n-\gamma)$ to achieve its security properties, while [14], [25] relying on BFT protocols run by the BB peers require (ii) $\gamma > \frac{2n}{3}$ (by construction) and $n_{h,a} > \frac{2n}{3}$ (trust assumption). We now show that our trust assumptions (i) are strictly weaker than (ii).

First, we show that (ii) implies (i). From $\gamma > \frac{2n}{3}$, it follows that $2\gamma > \frac{4n}{3}$. Therefore,

$$2n - 2\gamma < 2n - \frac{4n}{3} = \frac{2n}{3} < n_{h,a} \leq n_h.$$

Second, we show by a counterexample that in general the converse is false. Let $n = 12, \gamma = 10$, and $n_h = 5$. Then, (i) holds as $5 > 2(12-10) = 4$. However, (ii) does not hold as the statement $5 > \frac{2*12}{3} = 8$ is false. In this context, our solution only requires 5 out of 12 peers to be honest, while the BFT-based BBs require trusting at least 9 out of 12 peers to be honest.

3) Unconditional Verifiability [39]: [39] claims to achieve unconditional verifiability using an append-only BB. Their proof implicitly relies on readers having direct access to a *trustworthy*, centralized BB via an oracle call, which contradicts their claim that no trust is required.