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

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

Lara Schmid*
schmidla@inf.ethz.ch
Department of Computer Science,
ETH Zurich
Zurich, Switzerland

David Basin
basin@inf.ethz.ch
Department of Computer Science,
ETH Zurich
Zurich, Switzerland

ABSTRACT

The results of electronic elections should be verifiable so that any 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 Helios [1], Civitas [12], and Belenios [18] 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 that is 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 be used as a replacement for existing BBs, enabling verifiability under much weaker trust assumptions.

Note: *For reproducibility, our machine-checked proofs are available at [49].*

1 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 an e-voting protocol 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 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 then perform checks on the data they read. For instance, voters may check that their ballot was recorded correctly after casting it. Moreover, any participant, *e.g.,* voter or auditor, may check that all ballots were processed correctly by the authorities.

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 exact (minimal) guarantees must be satisfied by a BB for *verifiability* to hold. All verifiability definitions surveyed in [17] and most e-voting protocols that are formally proven to provide verifiability [4, 18, 39] actually make the overly conservative assumption of an *idealized BB* such as a shared memory or a broadcast channel. Some researchers regard the realization of BBs satisfying such strong requirements as an orthogonal problem [4, 15, 18]. Thus, it is unclear how and whether these assumptions can be met in practice.

Other researchers have proposed concrete BB designs that do not aim to realize an idealized BB and provide too weak guarantees. For instance, the BB in Civitas [12] is signed, Helios [1] suggests BB contents are (re)posted by several auditors, and [24] makes use of a Byzantine Fault Tolerant (BFT) algorithm. As we shall see, these mechanisms fail to provide sufficiently strong guarantees for verifiability when considering realistic assumptions associated with medium and large scale elections. We shall also see why solutions based on distributed ledgers or blockchains are unsuitable.

As a consequence, reference implementations of the state-of-the-art systems Helios, Civitas, and Belenios [1, 12, 18], that have been extensively used, notably in academia (*e.g.,* UCLouvain, Princeton, ACM, IACR), use BBs with too weak guarantees. Therefore, the centralized entity running the BB must actually be trusted for verifiability to hold in practice. The same holds for the web-based deployments of Helios and Belenios that are currently running [5, 37]. 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, 15, 16].

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. What is the actual security impact of this mismatch? What requirements must be satisfied by a BB for verifiability to hold? Can a concrete BB protocol achieve such requirements under realistic assumptions?

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. This opens even well-designed protocols to serious attacks. In particular, we present novel attacks on the state-of-the-art voting systems Helios [1], Civitas [12], and Belenios [18]. 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

*Both authors contributed equally to this research.

only been analyzed with an idealized BB, these attacks were missed in prior formal analyses.

Second, we propose a new verifiability definition that accounts for malicious BB behaviors and thus covers more realistic scenarios and also captures our attacks. We base our definition on the generic verifiability definition of [17] that subsumes all of the definitions surveyed in [17], which all assume an idealized BB. However, in stark contrast to [17], 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 properties of the BB are 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. It does not, however, impose any relation between the writes and the reads. We 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 [23, 24], we assume that the BB is implemented using multiple peers, only some of which need to be trusted. This is a more realistic trust assumption than assuming a single trusted entity as required by some prior works. Also, our protocol requires weaker trust and system assumptions than previous approaches based on BFT (*e.g.,* [24]). 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.

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

Outline. In Section 2, we present our system model, our threat model, and the specification language we use. In Section 3, 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 4 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 final-agreement in Section 5. In Sections 6 and 7 we discuss related work, including those solutions based on distributed ledgers, and draw conclusions.

2 BULLETIN BOARD (BB) MODEL

2.1 Setup, System, and Adversary Assumptions

2.1.1 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 content is intended to reach a final state where the *final content* represents the result of the process

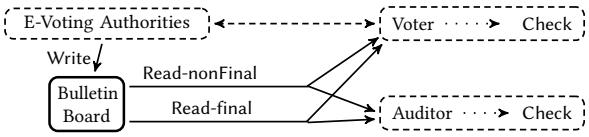

Figure 1: Typical setup of a BB in e-voting. The BB functionalities are depicted by solid lines, the verifiability checks by dotted lines, and the remaining architecture by dashed lines.

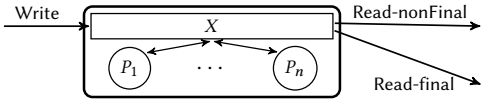

Figure 2: System setup of our BB protocol. The BB peers are depicted by circles, the proxy peer X by a rectangle, and the communication channels by arrows.

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 final content(s) and Read-nonFinal for reading any content on the BB, including non-final ones. The third and final functionality is Write.

2.1.2 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 content. A typical BB architecture for e-voting is depicted in Figure 1.

2.1.3 System Assumptions and Threat Model. A concrete BB solution must be analyzed together with appropriate system and adversary assumptions. We discuss these next.

System Assumptions. A concrete BB can be realized by a single role (*e.g.,* [35]) or by several (equal or different) roles, which we call *peers*, that run a protocol together (*e.g.,* [24]). 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 algorithms run by the readers, *e.g.,* readers cross-checking their views of the BB. However, assuming reader-reader communication makes very strong assumptions that are unrealistic for e-voting at scale, *i.e.,* medium and large scale elections, where all voters can be readers. Indeed, this would require:

(1) an infrastructure such as a Public Key Infrastructure (PKI) that voters use to identify and authenticate other genuine, eligible voters,
(2) 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 3.2.2), and
(3) the voters must trust their machines (voting platforms), which must be online and highly available.[1]

As such assumptions are unrealistic for 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, *i.e.,* the peers. 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 considered to be a single entity, it can be physically replicated on different servers to avoid a single point of failure.

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 also during the execution. All agents that are not malicious 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.

2.2 Formal Specifications in Event-B

2.2.1 Event-B Definitions. We use an event-based model based on Event-B [9] to formally describe protocols. First, we introduce standard notations and definitions for Event-B specifications. We shall use standard mathematical notations 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* value 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. $\vec{\cdot}$ denotes vectors, and $[a_i]_{i \in J}$ denotes a list of elements a_i indexed by elements in J.

Definition 2.1 (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 $\sigma = s_0.s_1.\ldots.s_n$ such that $s_0 \in \Sigma_0$ and $\forall i \in [0, n)$, $(s_i, s_{i+1}) \in \rightarrow$. $\mathrm{Be}(\mathcal{T})$ 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 $(\!|a = v_1, b = v_2, ..|\!)$. *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.

2.2.2 E-voting Protocol and BB Specification Framework. We abstractly represent e-voting and BB protocols as Event-B specifications. Our definitions are generic and focus on the interactions between e-voting entities, the BB entity, and verifiability checks, and they abstract away all other interactions.

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. Even though our results are established for the general case, one can work with a more intuitive 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 natural numbers and act as counters. Each $\mathrm{B} \in \mathcal{W}$ is thus associated with a phase, according to the function $\mathrm{ps} : \mathcal{W} \rightarrow \mathbb{N}^+$ and $\mathrm{ps}(\mathrm{B}_\perp) = 1$.

Our Framework, the Big Picture. We focus on the BB's impact on verifiability in e-voting. We therefore formalize a protocol as the combination of two specifications, modeling the BB and the rest of the e-voting system. We describe how these two specifications interact through specific state variables, as explained next; see Figure 3 for a graphical presentation.

An agent A accessing the Write functionality to write some content B_w, shown in solid, red lines in Figure 3, is modeled as B_w being added to the set in wr (*write requests*). Depending on the BB specification, the BB may then update the state variable w (*writes*) by adding B_w to the corresponding set. Hence, the state variables

[1] This is (only) problematic for e-voting protocols that assume that voters' machines can be compromised and that rely on specialized devices instead [6, 31, 32, 45].

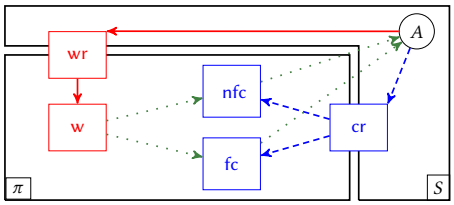

Figure 3: Abstract representation of the interactions between the BB (π) and the e-voting (S) specifications using dedicated state variables (in squares). Write actions for an agent A are depicted with red, solid arrows and read actions with blue, dashed arrows. Dotted arrows denote interactions that 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).

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 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 using Read-final to retrieve some BB content for evaluating such a 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). Accesses to Read-nonFinal are similar, except that the processed check requests are stored in the state variable nfc (*non-final-checks*) rather than in fc.

Finally, a state variable agents, from the e-voting specification, keeps track of all honest and malicious running agents.

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 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 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 $\mathcal{C}_\mathsf{nf} = \mathcal{C} \setminus \mathcal{C}_\mathsf{f}$. We describe verifiability checks further in Section 3.1. 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 2.2 (S). An *e-voting specification* S is a specification whose state variables contain: $\mathsf{cr} : \mathcal{P}(\mathcal{C} \times \mathcal{M}^* \times \mathcal{A})$, $\mathsf{wr} : \mathcal{P}(\mathcal{W})$, and $\mathsf{agents} : \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A})$. Moreover, cr and wr are initially empty and monotonically increasing; for instance, for cr, $\forall \sigma.s.\sigma'.s' \in \mathrm{Be}(S)$, $s.\mathsf{cr} \subseteq s'.\mathsf{cr}$.

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 2.3 (π). A *BB specification* π is a specification containing the state variables cr and wr (see Definition 2.2), $\mathsf{fc} : \mathcal{P}(\mathcal{C} \times \mathcal{M}^* \times \mathcal{A} \times \mathcal{W})$, $\mathsf{nfc} : \mathcal{P}(\mathcal{C}_\mathsf{nf} \times \mathcal{M}^* \times \mathcal{A} \times \mathcal{W})$, and $\mathsf{w} : \mathcal{P}(\mathcal{W})$ such that cr, wr, fc, nfc, and w are initially empty and monotonically increasing. π has neither write access to cr nor to wr, *i.e.,* their values can only be used in guards.

For a state s, $\mathsf{checksA}(s)$ 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 2.4. Consider the BB specification $\pi_\mathrm{B}^\mathrm{ide}$ defined below, which provides the functionalities depicted in Figure 2. The specification models an idealized BB that acts as a shared variable. That is, 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.

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

Next, consider $\pi_\mathrm{B}^\mathrm{tru}$, which is as $\pi_\mathrm{B}^\mathrm{ide}$ 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), that is the BB only processes checks that were requested.

Definition 2.5 (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) $\forall \sigma.s \in \mathrm{Be}(S \cup \pi)$, $\mathsf{checksA}(s) \subseteq s.\mathsf{cr}$. 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 2.6. 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, $\forall \sigma \in \mathrm{Be}(P(S, \pi))$, $\sigma \vdash T$.

Example 2.7. The *written-as-requested* predicate War denotes that all write requests were processed before reading, i.e., $\sigma \vdash \mathrm{War}$ 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, $\sigma \vdash \mathrm{RaW}$ 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*. We call it *idealized* if it also satisfies War. It is easy to see that $\pi_{\mathrm{B}}^{\mathrm{ide}}$ (respectively $\pi_{\mathrm{B}}^{\mathrm{tru}}$) from Example 2.4 is an idealized (respectively trustworthy) BB.

3 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.

3.1 Defining Verifiability for Malicious BBs

We now investigate how verifiability can be formally defined in our framework. We first explain that most prior verifiability definitions assumed a trustworthy or even an idealized BB and we formalize these in our framework. We then generalize verifiability by relaxing the restrictions on the BBs found in prior works, and allowing malicious BBs instead.

The Big Picture. Verifiability enables voters and auditors to detect any malicious behavior by the election authorities. The core property is so-called *end-to-end verifiability*, which essentially states that the election's result has been correctly computed based on all eligible voters' votes [17, 42]. It is common to divide this into sub-properties targeting individual steps of the election process [14, 41, 42]. 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.

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* [17]. 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.

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

Definition 3.1. A verifiability check is a predicate $C : \mathcal{P}(\mathcal{W} \times \mathcal{M}^*) \in \mathcal{C}$. C is said to be *final-only* when $C \in \mathcal{C}_{\mathsf{f}} \subseteq \mathcal{C}$. We require that all verifiability checks that are not final-only must be *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, if re-voting is allowed, the check that at most one ballot has been registered per voter is not monotonic, it may hold at a given time but can later be violated when a voter re-votes. Therefore, to provide meaningful guarantees such checks must be evaluated on 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 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 that can contain static information about them such as their intended choices.

Definition 3.2. A *verifiability goal* τ is a predicate over $\mathcal{W} \times \mathcal{P}(C \times \mathcal{M}^* \times \mathcal{A}) \times (\mathcal{A} \times \mathcal{A})$ 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 [17], where different verifiability notions are cast in the same framework. All the verifiability properties analyzed in [17] are only defined for protocols that assume an idealized or trustworthy BB (*e.g.,* $\pi_{\mathrm{B}}^{\mathrm{tru}}$ or $\pi_{\mathrm{B}}^{\mathrm{ide}}$ from Example 2.4), 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 3.3 (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 final outcome on 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, possibly unrelated to 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, even for malicious BBs. That is, we can only define verifiability for BBs that

	Threat model BB	Threat model other roles	Violated properties	Exploited BB weakness	Fixed by FA
C.1	none	none (all tellers hon.)	IV, UV	partial BB not on final	✓
C.2	none	none (all tellers hon.)	IV	inconsistent views	✓
C.3	none	tabulation tellers	IV, UV	no unique final	✓
C.4	none	none (all tellers hon.)	IV, UV	no unique final	✓
C.5	none	none (all tellers hon.)	EV, privacy (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	✓
B.3	voting server	none	BB auditability	–	✓
B.4	voting server	none	privacy (BP)	B.2 + [19]	✓

Each attack, except B.3, violates at least one security claim from [12, 18] under the same threat model. We denote by *basic threat model*, the weakest adversary considered in [12] and [18]. The 2nd and 3rd columns denote the threat model required for the attack. It shows, compared to the basic threat model (none), which additional entities must be malicious and which entities may additionally be honest (hon.). The 4th column denotes the properties violated by the attacks, where we distinguish two levels of privacy: ballot privacy (BP) and coercion-resistance (CR). The last column indicates whether the attack is fixed by a BB satisfying FA, as will be introduced in Section 4.

Figure 4: Summary of attacks on Civitas [12] (C.1-C.5) and Belenios [18] (B.1-B.4) (also affecting Helios [1]).

present the same content to all readers using Read-final. We thus define a predicate, called *final-consistency* (FC in short), which holds for a behavior $\sigma.s$ when $|\mathsf{B}(s.\mathsf{fc})| \leq 1$. That is, any BB specification π satisfying FC never provides two final check requests with two different BB contents. (Note that RaW strictly implies FC). For such specifications, we define the unique final content in a state s, denoted by $\mathsf{finalB}^+(s)$, as either $\mathsf{B}_\perp$ when $s.\mathsf{fc} = \emptyset$ (no one has read the final BB), or B_f, where $\mathsf{B}(s.\mathsf{fc}) = \{\mathsf{B}_f\}$, otherwise (when at least one reader has read the final BB). Note that $\mathsf{finalB}^+(s)$ may be totally unrelated to $s.\mathsf{wr}$ and $s.\mathsf{w}$. Given this, we define verifiability$^+$, which is a variant of verifiability suitable for any BB that is FC.

Definition 3.4 (Verifiability$^+$). A protocol $P(S, \pi)$ with $\pi \vdash \mathsf{FC}$ provides *verifiability$^+$ for a verifiability goal* τ when

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

3.2 Practical Attacks with Malicious BBs

We now investigate the security impact of a malicious BB. We present several new, practical attacks on the existing e-voting systems Helios [1], Civitas [12], and Belenios [18], where an adversary controlling the BB can manipulate the election without being effectively detected, hence violating verifiability, even when their respective BB auditing mechanisms are used. We emphasize that our attacks can be carried out with respect to threat models under which these schemes were claimed to be secure [1, 12, 18], thus refuting these claims. We summarize our attacks, their underlying threat models, and which property they violate in Figure 4.[2] In Section 3.2.3, we discuss at greater length why the BB auditing mechanisms that have been proposed for these schemes fail to counter these attacks. Finally, we stress that our attacks do not require the BB to provide different readers with different election results, as readers may have side-channels where this is broadcast (*e.g.,* TV).

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 [1] 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 [13] are made under the assumption that the BB is honest, which is too strong for most realistic deployments. Moreover, Belenios is claimed to be secure when the entity running the BB is malicious in [18], which we also refute. Recall that verifiability may include some form of IV, UV, and EV. As our attacks fundamentally break verifiability, independently of its specific definition, we consider it informally here and refer to [12] and [18] for the formal definition of verifiability used in Civitas and Belenios.

3.2.1 Civitas. Civitas [2, 12] builds on JCJ [39] 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* 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 next the protocol, enumerating its main steps for reference.

At the protocol's setup, (s1) the supervisor publishes the election parameters on the BB and (s2) the registration tellers 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 registration tellers 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 do not satisfy well-formedness 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

[2] 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 A.3.

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 registration teller. Voters can verify that their votes were correctly recorded by reading them from the BB and checking the list of ballots 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.

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 break crucial security properties, which are more critical than availability. Note that the original formal proof [39] (for JCJ) considers an honest BB and is thus too weak to back up the aforementioned security claims.

Attack C.1. 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 registrar tellers retrieve the list of credentials. As a consequence, the TTs will (silently) discard A's ballot b_A at Step (t3) since b_A has no matching authorized public credential. Even when A performs the IV check (specified in [12]) on the final BB content, this attack cannot be detected as b_A is in the list of ballots processed by the TTs.[3] Therefore, the announced result does not take A's ballot into account; however no verifiability check is violated.

This attack breaks IV and UV, as a valid recorded ballot is not included in the tally. It can be carried out by a malicious BB (and hence a malicious supervisor) even when all other entities are honest. One could fix this by mandating that voters check IV on the final content only and also check that their public credential is still included on the BB at this point. However, this is against the so-called *Vote & Go* paradigm (*e.g.,* [8, 11, 38, 46, 50]), where after voting and performing checks, voters must no longer observe the election process. Thus, we advocate for using a more secure BB instead.

Attack C.2. The information published by the supervisor at Step (s1) includes a list of ciphertexts $C = (c_1, \ldots, c_n)$ associated with the choices v_i that voters can vote for. According to [2], 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 proves in the well-formedness proof that the underlying c_i is contained in C. At Step (t1), the TTs also read C from the BB and (silently) discard ballots that do not have a valid well-formedness proof with respect to C.

The attack C.2 exploits this as follows. 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 $C' \neq C$. As a result, A will compute and cast a ballot (v2-v3) that will be discarded by the TTs at step (t1) since the latter receive the (untampered) list C. Similar to the attack C.1, A's IV check does not detect the malicious behavior, which constitutes an attack against IV.

Attack C.3. It is claimed that verifiability is satisfied even when all TTs are malicious. Under such assumptions, there is no honest entity authenticating the set of ballots considered for tallying. Therefore, the TTs and the BB can defeat verifiability by showing different final contents to different readers as explained next. The adversary first chooses the 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 outcome of his choice when tallied, and (ii) contain the reader's ballots if any.[4] 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.

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 breaks IV and UV. Since the BB is supposed to show to the readers the tallies from all block's outcomes, it could selectively drop the results from some specific blocks in order to obtain 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. This can be fixed by using a secure BB, like ours.

Attack C.5. Public credentials are not bound to block identifiers but are delivered to voters by registrars upon checking inclusion of the voters in the block. A ballot computed by a voter contains a block identifier but no single entity is able to 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. We describe this attack in detail in Appendix A.3. While this attack leaves evidence, no specified auditing or detection mechanisms (*e.g.,* UV checks) can detect it. (We discuss why this matters in Section 3.2.3.)

This attack requires 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 use case is out of the scope of 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 5 would nevertheless prevent this attack.

3.2.2 Belenios and Helios. We now discuss Helios [1] and Belenios [18, 30], which builds upon Helios. In the following, we mainly focus on Belenios as it aims at providing strictly stronger guarantees than Helios and as (slight variants of) our attacks on Belenios also apply to Helios [1]. Belenios improves Helios by providing voters with credentials to avoid *ballot stuffing*, where the adversary adds illegitimate ballots to effect the election's outcome. The primary goal of these changes was to achieve security under weaker trust assumptions [16], *i.e.,* when the ballot box is dishonest. Our

[3] The list of discarded ballots remains secret to achieve coercion-resistance.

[4] It might be difficult to know a reader's identity. However, we believe the adversary can target specific voter platforms and track their accesses with sufficiently high probability. The attack can then be mounted by only tampering with ballots of these voters.

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.

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 then 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.

Belenios' security proofs [13] assume that the BB is a broadcast channel, but this assumption is not always explicit and not met by practical deployments. In particular, it is claimed in [18] 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 [18], we shall consider a malicious BB and see that this claim is refuted by the attacks B.2 and B.3. [18] 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. We also refute this claim with the attack B.1.

Helios [1] also realizes the BB by a centralized web-server and proposes that auditors can *repost* data from the BB so that voters can check that their ballot was considered by the auditors. This is against the *Vote & Go* paradigm discussed earlier; we will discuss why this technique fails in practice in Section 3.2.3.

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

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. 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.

This would be noticed if each voter also checked that the BB content considered in her IV check matches the BB content that is (correctly) processed by the DTs (*e.g.,* through an additional UV check). However, this is not desirable as, in practice, voters may not want to come back after the election's end (*Vote & Go*). Moreover, this is unrealistic as voters do not have the computational power, bandwidth, or 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. Finally, it seems anyway that Belenios only requires voters to check that their ballots are in the ballot box. As explained above, this does not yield IV, even when such IV checks are performed by all voters on the final BB content.

We conclude that in its current state, Belenios deployments fail to provide IV when the voting server is malicious, and so does Helios. Using a secure BB instead, as we suggest later, would fix this problem for both protocols.

Attack B.3. When discussing the BB in practice, [18] 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 a BB content. However, such BB monitoring and auditing is insufficient. If auditing tools only verify that the successive local views on the BB are append-only, this does not guarantee any agreement on the BB contents obtained by different readers. To avoid this, one could require that readers are interconnected and exchange their BB views. However, such a requirement is not specified in [18] and is also impractical, as explained in Section 2.1.3.

Attack B.4. It has recently been shown [19] that a lack of IV violates ballot privacy. That is, a malicious ballot box or BB can modify or drop all ballots except the one of Alice before tallying to learn Alice's vote. Our conclusion that an insecure BB violates IV therefore implies that it also violates privacy. Thus, a secure BB is necessary not only for verifiability but also for privacy.

3.2.3 Why do BB auditing techniques fail? Even though Civitas signs BB contents and Belenios allows each reader to check that BB contents are only monotonically increasing, readers would still have to compare all the contents they read with other readers to actually detect any BB misbehavior. However, as we argued in Section 2.1.3, such reader-reader communication is unrealistic. Regarding Helios, we argued that its re-posting mechanism violates the *Vote & Go* paradigm as it requires all voters to store their ballots, wait for the election's end, and find their ballot in the auditors' posted lists. Moreover, it essentially boils down to auditors acting as BB peers signing final contents, except that the informal presentation from [1] implies that all auditors must be trusted.

For designing systems with realistically workable end-to-end verifiability, the assumptions regarding the voters and their actions must also be realistic (*e.g.,* no reader-reader communication, Vote & Go, download of a reasonable amount of data). Indeed, relying on the voters to perform non user-friendly, time-consuming, or complex IV checks results in insufficiently many voters actually performing these checks correctly, which compromises the election's verifiability in practice.[5] Moreover, the protocol itself must include a precise specification of these actions, in particular the verifiability checks and their timing. (This is, for example, not the case in both Helios and Belenios UI, where it is proposed to the voters to perform IV checks right after casting the ballot, even though, in theory, they could also re-check after the election's end.) Finally, even though BB misbehavior may leave some evidence, it could remain undetected if dedicated and explicit auditing mechanisms are not specified and actually carried out. This is analogous

[5] See footnote in Section 6.2 for a concrete example.

to vulnerabilities that can remain unnoticed and exploitable, even though their exploitation may leave evidence.

4 FINAL-AGREEMENT (FA)

Our attacks demonstrate that it is crucial to consider a realistic BB model when making security claims. In particular, instead of considering honest BBs, system designers should assume BBs providing requirements that can be met in practice under realistic trust assumptions. We next introduce such a BB requirement, which is achievable in practice under weak trust assumptions (as shown in Section 5), but nevertheless is sufficient for verifiability.

4.1 Definition

We have already explained in Section 3.1 that verifiability can only be meaningfully defined if there is one well-defined final BB, i.e., the BB must satisfy FC. Requiring FC also prevents the attacks C.3, C.4, and B.1 from Section 3.2. Additionally, the attacks C.1 and B.2 show that a BB that can drop items between non-final reads and final reads has dramatic security consequences. For example, if a voter checks that his ballot is recorded on the BB right after voting (*Vote & Go*), she must be sure that her ballot is still included in the final BB content. Otherwise, such checks on intermediate BB contents provide no guarantee (see *e.g.,* attack B.2). We thus 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 4.1. Final-agreement (FA) holds for $\sigma.s$ when (i) $\sigma.s \vdash$ FC and (ii) $s.\mathrm{fc} = \emptyset \vee \mathrm{B}(s.\mathrm{nfc}) \sqsubseteq_b \mathrm{B}(s.\mathrm{fc})$.

Note that 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 Read-nonFinal, nor relates the writes and the reads. Nevertheless, if a BB with FA was used by Helios, Civitas, and Belenios, then all the attacks in Figure 4 except C.5 would be prevented, as shown in the Figure's last column.

We discuss next FA in e-voting and refer to Appendix A.4 for a presentation of other scenarios for which FA is also suitable.

4.2 FA in E-voting

We show next that any protocol satisfying Definition 3.3 (verifiability) with an idealized BB, as proven in many prior works, also satisfies Definition 3.4 (verifiability$^+$ under a malicious BB) provided that the BB satisfies FA. Additionally, we prove the converse for a large class of checks and goals: it is impossible to achieve Definition 3.4 with a BB that does not satisfy FA.

4.2.1 FA is Sufficient for Verifiability. Recall $\pi_{\mathrm{B}}^{\mathrm{ide}}$ and $\pi_{\mathrm{B}}^{\mathrm{tru}}$ from Example 2.4. 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 $\pi_{\mathrm{B}}^{\mathrm{ide}}$ or $\pi_{\mathrm{B}}^{\mathrm{tru}}$ also satisfies verifiability with any, possibly malicious BB π, provided that $\pi \vdash$ FA.

First, we relate verifiability under $\pi_{\mathrm{B}}^{\mathrm{tru}}$ with verifiability$^+$ under π. Second, we relate verifiability under $\pi_{\mathrm{B}}^{\mathrm{ide}}$ 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 insensitive to malicious write requests. We formally define these assumptions and prove the following theorem in Appendix A.

Theorem 4.2. *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 $P(S, \pi_{\mathrm{B}}^{\mathrm{ide}})$, $P(S, \pi_{\mathrm{B}}^{\mathrm{tru}})$, and $P(S, \pi)$ are protocols.*

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

(2) *When $P(S, \pi_{\mathrm{B}}^{\mathrm{ide}})$ provides* verifiability for $\tau, \pi \vdash$ 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 an idealized 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 5). For instance, Belenios [13] assumed an idealized BB and Alethea [4] assumed a trustworthy BB. The formal definition and security claim of verifiability for Civitas originate from [39] and also assume an idealized BB. Hence, our results allow weaker, more realistic trust assumptions than those currently used for existing e-voting schemes.

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

There are protocols, such as CHvote [33], that do not utilize a BB and for which verifiability relies instead on so-called *verification codes* (see Appendix A.2). 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. Loosely speaking, we show in Appendix A.2 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 any time after vote casting.

5 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.

5.1 Design Rationale

Recall our system assumptions from Section 2.1.3: 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 $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$.

Our protocol works as follows. Each peer locally stores its current BB view. When peers receive write requests with new content (forwarded by X), they update their BB views accordingly and sign the updated content. These signed contents are collected again 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.

5.1.1 Trust Assumptions vs. Availability Trade-off. We could require that readers only accept BB contents when they obtain *all* peers' signatures on this content. 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. Therefore, we consider a fixed *threshold* $\gamma \in [1, n]$ and state that a reader can read a BB content when he receives valid signatures on this content from at least γ peers. We will later give a tight lower bound on γ for FA to be satisfied, which depends on the number n_h of honest peers. 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. This honest peer ensures that the two readers obtained consistent BB contents due to the signing policies that the honest peers follow, which we specify next.

5.1.2 BB Peers' Policies. We require that peers only update their BB view with new contents that extend their previous views. That is, they only update 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.

To achieve FA (i), the peers must sign at most one final BB. Otherwise, two readers obtaining a signature from the same honest BB peer, may disagree on the final, signed content. To enforce this policy, we use the notion of phases from Section 2.2.2, which denote the different stages when the BB is updated. In particular, we assume that there is a pre-defined, agreed upon *final phase* $p_f \in \mathbb{N}^+$. We then 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, *i.e.,* a part of the current BB's 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.,* $\mathrm{B}_j \sqsubseteq_b \mathrm{B}$. Due to the monotonicity (w.r.t. $\sqsubseteq_b$) of the successive BB updates, this locally implies FA (ii).

Based on these policies, honest peers locally enforce FA, *i.e.,* FA holds from each peer's perspective. This and the correct choice of γ, which entails that two readers always have an honest peer in common for each read, ensures FA globally.

5.1.3 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 5.1 (Partial BB Contents). Let $\mathrm{partial} : \mathcal{W} \to (\mathsf{L} \mapsto \mathcal{W})$ 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 $\mathrm{partial}(\mathrm{B}) = [\mathrm{B}_l]_{l \in L}$ for some $L \subseteq \mathsf{L}$. We sometimes consider partial(B) as a subset of $\mathcal{W}$. We assume given a distinguished label $\mathsf{f} \in \mathsf{L}$ corresponding to final contents and we assume that $\forall \mathrm{B} \in \mathcal{W}$, $\mathrm{partial}(\mathrm{B})(\mathsf{f}) = \mathrm{B}$. We additionally assume that $\forall \mathrm{B} \in \mathcal{W}, \mathrm{B}' \in \mathrm{partial}(\mathrm{B}),\ \mathrm{B}' \sqsubseteq_b \mathrm{B}$.

In e-voting, L typically includes the voters' identities and, given an identity, partial returns the content associated with this identity (*e.g.,* the 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.

5.2 Generic Protocol

As explained above, 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 $\mathrm{getSig}(\mathcal{DB}, p, l)$ 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 5, 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 5.1.2. All signatures produced are then sent to X, who stores them in $\mathcal{DB}$.

Figure 6 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 this is the case, 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 $\mathrm{getSig}(\mathcal{DB}, p_f, \mathsf{f})$ to retrieve all signatures on the full, final BB contents.

5.2.1 Threshold γ. For our protocol to satisfy FA, γ must be chosen such that $\gamma > n - \frac{n_h}{2}$. Indeed, this bound ensures that any two readers obtained at least one signature from the same honest peer. We formally prove this and that the bound is tight (FA is violated otherwise) in Appendix B.1.

5.2.2 Availability. If too few peers agree on the content to sign, for example when X sends them different contents B_w, then there

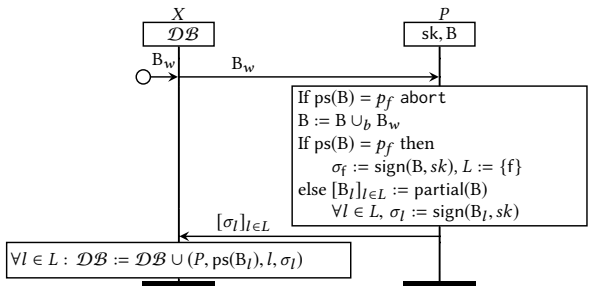

Figure 5: The sub-protocol BB-Write. When the proxy X receives a write request with B_w, it forwards it to all peers, which we only describe for one peer P. Here, sk and B respectively denote P's signing key and P's current view of the BB content. Initially $B := B_\perp$. sign(m, sk) denotes the signature of m under the signing key sk and $\mathcal{DB}$ stores all signatures.

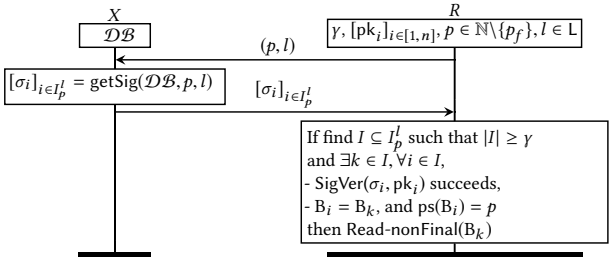

Figure 6: The sub-protocol BB-Read-NF for reading non-final BB contents. SigVer(σ, pk) **denotes the verification of the signature σ with the verification key** pk**. Upon receiving a request for a content labeled l at phase $p \neq p_f$, the proxy X retrieves all peers' signatures for this label and phase using** getSig**. From these, the reader then tries to find sufficiently many matching signatures.**

is no hope for the readers to obtain sufficiently many matching signatures and they cannot read any content. Note that this is an availability issue and not a violation of verifiability. We stress that this is a general problem: to provide a certain level of availability, any protocol relies on some system assumptions that might not be relevant for security, such as reliable communication channels.

In practice, a BB should provide both security and availability. To enhance availability, one could combine our protocol with the ideas of prior BB protocols, which provide better availability but insufficient guarantees for verifiability (see Section 6): *e.g.,* we would omit X and, similarly to [24], connect the readers with all peers who could run a BFT algorithm at each writing request to agree on a BB content before signing it.

5.3 Using our BB for E-voting

For e-voting, our protocol could, for example, be instantiated 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 voters' identities, and of the final BB, labeled by the distinguished label f. Authorities could publish the received ballots right after their reception using Write and voters could use BB-Read-NF to read the partial BB containing their ballot and perform an IV check. Also, at the election's end, auditors can use BB-Read-F to read the final BB and perform all required UV checks.

5.3.1 Consequences for Verifiability. As FA requires trusting some peers and verifiability requires FA (Section 4.2.2), we can conclude that *verifiability requires trust.* Looking at the bigger picture, this is at odds with some prior works claiming that verifiability can be achieved with no trust assumptions at all [28] or no trust assumptions with respect to the BB [16, 18].

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. This does not require more trust assumptions than those needed anyway. The same holds for EV checks. Our analysis allows to draw 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. This enhances flexibility in how the voters access the BB, for example it enables the use of specialized trusted devices as discussed next.

5.3.2 Practical Considerations. In practice, voters must trust their machines to achieve any guarantees. We explain in Appendix B.2 that our protocol can be used in settings where this trust is put in *specialized devices* [4, 6, 31, 32, 45], which have limited capabilities and connectivity. Moreover, we address in Appendix B.2 that 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 discuss the resulting trade-offs.

5.4 Security Analysis

We use TLA+ [43] to specify our protocol and FA and formally establish that the former satisfies the latter. TLA+ has a rich specification language [9] 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 partial than presented in this paper. Using the embedded proof system TLAPS [21], 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 5.2.1 and establish that FA is an invariant of our protocol. All our specifications and machine-checked proofs (ca. 800 L.o.C. in total) can be found at [49].

6 RELATED WORK

There has been extensive prior work on both voting and on BB-like designs in the broad sense (consensus algorithms, distributed ledger technology, etc.). Here we focus on the most relevant related work and we refer the interested reader to Appendix C for further details.

6.1 BB Realizations

6.1.1 Realizations Based on BFT Algorithms. [24] designed for the poll-site voting scheme *vVote* [22, 23] a distributed BB protocol, later improved in [40]. The main differences to our work are (1) the security properties that are aimed for, which do not explicitly include an agreement among the readers, and (2) the consensus mechanism that is leveraged, namely a BFT algorithm that requires strictly stronger trust and system assumptions than our protocol. The emphasis of the algorithm of [24, 40] is on (i) reaching an agreement among sufficiently many BB peers before they jointly sign a content and on (ii) providing writers with a "receipt" that should give assurance that the written item will be part of the next content. (i) is achieved through a lightweight BFT algorithm run among peers and a threshold signature scheme, which is also used to sign the written items for producing the receipts (ii). 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. As is standard for BFT, the protocol meets the specified properties when at least $\frac{2n}{3}$ out of the n peers are honest and available. We show next that this is a strictly stronger trust assumption than ours.

A 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. Our protocol requires (i) $n_h > 2(n-\gamma)$ while BFT consensus protocols run by the BB peers, *e.g.*, the one from [24], require (ii) $\gamma > \frac{2n}{3}$ (by construction) and $n_{h,a} > \frac{2n}{3}$ (trust assumption). We formally establish in Appendix C.2 that (i) is strictly weaker[6] than (ii). Our protocol requires weaker assumptions as it achieves a weaker consensus property between the BB peers.

6.1.2 Distributed Ledger-based Realizations. Some researchers have proposed BBs and even e-voting realizations embedded in distributed ledgers [27, 44, 48, 51, 52]. Most proposals utilize permissionless ledgers (*e.g.*, Bitcoin or Ethereum). Thus readers, including voters, must either run a full node or trust external full nodes (third parties). The former involves too strong system and threat assumptions regarding the voters, concerning their availability, bandwidth, storage, and trust in their voting platform (see Section 2.1.3), which is also supported by [34]. Moreover, as is standard with distributed ledgers, both types of solutions crucially rely on economic incentives, which are hard to quantify for elections. Finally, although such technologies are supposed to provide decentralization, they are often centralized in practice due to *pools* [29]. [36] surveys such shortcomings and draws similar conclusions. Other solutions leverage permissioned ledgers where a few distinguished nodes establish a consensus on data that can be publicly accessed by all other nodes. This boils down to the BFT solutions discussed in Section 6.1.1. Finally, one may use permissioned ledgers but with all e-voting participants acting as full nodes. Readers would then have to run full nodes, which is impractical as argued above.

6.1.3 [35]. Heather and Lundin present another BB protocol that is often cited as a potential BB solution for e-voting schemes [25, 46, 47]. The stated properties do not include any agreement among readers and thus do not exclude the attacks from Section 3.2. Also, the protocol's decentralized variant is only proposed for improving robustness and still requires that all peers are trusted.

6.2 BBs in E-Voting Protocols

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

For example, the *JCJ* e-voting protocol [39] does not mention how the BB is realized and just assumes a *universally accessible memory* that all agents can write to in an append-only manner. *Alethea* [4] assumes that the BB is realized by a single trusted entity that sends the content correctly and consistently to all readers. See Section 3.2 for Civitas [12], Belenios [18], and Helios [1]. *Prêt à Voter* [25], which is not a remote but a poll-site scheme, does not specify an explicit BB and refers to [35] instead (see Section 6.1.3). Building on [25], *vVote* [22, 23] (also poll-site) makes use of [24] for its *private BB*, to which the voters have no access, and assumes a *public BB*, corresponding to our 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 violates *Vote & Go* and requests the voters to cross-check information from different media.[7] *Scantegrity* [10] 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. The authors of [16] identify that a malicious BB could perform ballot stuffing before tallying and break verifiability this way and claim that an honest registrar suffices to prevent this misbehavior. Formally, they consider verifiability under malicious ballot boxes but still consider a BB that provides a broadcast channel whose content is partially under adversarial control. In particular, their "dishonest bulletin board" does not cover the attacks described in Section 3.2. Finally, [28] 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 the claim that no trust is required.

7 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 by practical BB realizations under weak trust assumptions. We also provide a concrete BB protocol and formally prove that it satisfies FA. Our protocol could be deployed in existing e-voting schemes to replace the current insecure BBs that constitute e-voting's Achilles

[6] *E.g.*, for $n = 12$, $\gamma = 10$, and $n_h = 5$, our solution only requires 5 out of 12 peers to be honest as opposed to 9 out of 12 for BFT. Note that γ can be greater than n_h since n_h only counts the amount of peers that we *must* trust to be honest, and not the peers that actually behave honestly, which can be higher and can depend on the reader.

[7] [7] reports on a user study about vVote in the Victorian state election that reveals that only 13% of the voters have accessed the BB to perform the IV check. In that particular case, they claim that this yields a security margin of 95% (chances of cheating detection). Considering that only a fraction of these voters would additional cross-check the hash (say 25%), this results in worrying security margins (50%).

heel. 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 adapted the definition of verifiability. Whereas we focused on possibilistic verifiability notions, our modifications appear generic and we speculate that one can similarly adapt probabilistic definitions. 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 versus weaker system assumptions in future work.

ACKNOWLEDGMENTS

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

A VERIFIABILITY, FA, ATTACKS

A.1 FA is Sufficient for Verifiability

This section is dedicated to the proof of Theorem 4.2.

Lemma A.1. *Let S be an e-voting specification, τ a verifiability goal, and π be an arbitrary BB specification, which can notably 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 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 2.4) that adds $(C, \overline{x}, a, \cup_b s.\mathrm{w})$ to the state variable $s'.\mathsf{fc}$, where s' is the current state. Note that for any such state s', $s'.\mathrm{w} = s_w.\mathrm{w} = \{\mathrm{B}_f\}$ and $\cup_b s_w.\mathrm{w} = \mathrm{B}_f$. 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 (†) : $s_f.\mathsf{fc} = \{(C, \overline{x}, a, \mathrm{B}_f) \mid (C, \overline{x}, a) \in \mathsf{checksA}(s)\}$. 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, † follows from the construction of σ_r.

(†) entails (b) as $s_f.\mathsf{nfc}$ is empty by construction, and (d). We have that $s_f.\mathrm{w} = s_w.\mathrm{w} = \{\mathrm{B}_f\}$ 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 A.5, corresponding to the second part of Theorem 4.2, 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 A.2. WrF is such that $\sigma.s \vdash \mathsf{WrF}$ if, and only if, ($\sigma.s \vdash \mathsf{FC}$ and $s.\mathsf{fc} = \emptyset \vee (\forall \mathrm{B}_x \in s.\mathrm{wr}, \mathrm{B}_x \sqsubseteq_b \mathsf{finalB}^+(s))$).

Definition A.3. Let $P(S, \pi)$ be a protocol. Let $C_{\mathrm{in}}(\mathrm{B}, \mathrm{B}_x)$ 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) $\forall \sigma.s \in \mathsf{Be}(P(S, \pi)), \mathrm{B}_w \in s.\mathrm{wr}, \exists a \in \mathcal{A}$, $(C_{\mathrm{in}}, \mathrm{B}_w, a) \in s.\mathsf{cr}$ and
(2) $\forall \sigma.s \in \mathsf{Be}(P(S, \pi)), s_0 \in \sigma, (C_{\mathrm{in}}, \mathrm{B}_w, a) \in s_0.\mathsf{cr}$, $s.\mathsf{fc} \neq \emptyset \implies \exists \mathrm{B} \in \mathcal{W}$, $(C_{\mathrm{in}}, \mathrm{B}_w, a, \mathrm{B}) \in (s.\mathsf{nfc} \cup s.\mathsf{fc})$.

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 $\sigma.s \vdash \mathsf{FA}$, $\sigma.s \vdash \mathsf{FC}$. Thus, if $s.\mathsf{fc} = \emptyset$, then $\sigma.s \vdash \mathsf{WrF}$ holds. Otherwise, as $\sigma.s \vdash \mathsf{FC}$, 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 $(C_{\mathrm{in}}, \mathrm{B}_w, a) \in s.\mathsf{cr}$. Since $\mathsf{fc} \neq \emptyset$, there exists some $\mathrm{B} \in \mathcal{W}$ such that $(C_{\mathrm{in}}, \mathrm{B}_w, a, \mathrm{B}) \in (s.\mathsf{nfc} \cup s.\mathsf{fc})$. Since $\bigwedge_{(C,\overline{x},a,\mathrm{B})\in(s.\mathsf{fc}\cup s.\mathsf{nfc})} C(\mathrm{B},\overline{x})$, then $C_{\mathrm{in}}(\mathrm{B},\mathrm{B}_w)$ 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 A.4. 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 by addition of unauthenticated data (*i.e.,* not in $\cup_b s.\mathsf{wr}$) and verifiability checks are stable by removal of unauthenticated data.

For instance, for an UV check, we should ensure that the ballots that are 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 [16, 18] over Helios.

Lemma A.5. *Let S be an e-voting specification, τ a verifiability goal, and π be an arbitrary BB specification, which can notably specify a malicious BB. We assume that $P(S,\pi^{\mathrm{ide}}_{\mathrm{B}})$ and $P(S,\pi)$ are protocols. When $P(S,\pi^{\mathrm{ide}}_{\mathrm{B}})$ provides* verifiability for τ, $\pi \vdash \mathsf{FA}$, *$P(S,\pi)$ 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 $\pi \vdash \mathsf{FA}$ 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 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 the one of Lemma A.1, except that $\pi^{\mathrm{ide}}_{\mathrm{B}}$ has more guards than $\pi^{\mathrm{tru}}_{\mathrm{B}}$ and the behavior of $P(S,\pi^{\mathrm{ide}}_{\mathrm{B}})$ we shall build from $\sigma.s$ in **(3)** will be different.

(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, 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 $\mathrm{B}^i_f := \cup_b s.\mathsf{wr}$ and we shall establish that $\mathrm{B}^i_f \sqsubseteq_b \mathrm{B}_f$. 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, $\mathrm{B}^i_f \sqsubseteq_b \mathrm{B}_f$, which yields $\mathrm{B}_f \cap_b \mathrm{B}^i_f = \mathrm{B}^i_f$. Since $P(S,\pi)$ and τ authenticate write requests, we have that (1) implies $\bigwedge_{(C,\overline{x},a,\mathrm{B})\in(s.\mathsf{fc}\cup s.\mathsf{nfc})} C(\mathrm{B}^i_f,\overline{x})$.

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

- (a) $\mathsf{finalB}(s_f) = \mathrm{B}^i_f$,
- (b) $\forall(C,\overline{x},a,\mathrm{B}) \in (s_f.\mathsf{fc} \cup s_f.\mathsf{nfc})$, $\mathrm{B} = \mathrm{B}^i_f$,
- (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 $\sigma^0_f.s^0_f.\sigma_w.s_w.\sigma_r \in \mathsf{Be}(P(S,\pi^{\mathrm{ide}}_{\mathrm{B}}))$ from $\sigma.s \in \mathsf{Be}(P(S,\pi))$ as follows. First of all, any action from S is left unchanged and any action from π is replaced by Skip(). We obtain this way a behavior $\sigma^0_f.s^0_f \in \mathsf{Be}(P(S,\pi^{\mathrm{ide}}_{\mathrm{B}}))$. Next, we complete this behavior as follows. First, for any $\mathrm{B} \in s^0_f.\mathsf{wr} = s.\mathsf{wr}$, 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 $s_w.\mathsf{wr} = s^0_f.\mathsf{wr} = s.\mathsf{wr}$. Second, for any $(C,\overline{x},a) \in s_w.\mathsf{cr} \cap \mathsf{checksA}(s)$, we trigger an RF action (see Example 2.4) 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, $\cup_b s'.\mathsf{w} = \mathrm{B}^i_f$. Since all guards of $\pi^{\mathrm{ide}}_{\mathrm{B}}$ hold, we obtain $\sigma^0_f.s^0_f.\sigma_w.s_w.\sigma_r \in \mathsf{Be}(P(S,\pi^{\mathrm{ide}}_{\mathrm{B}}))$. Let s_f be the last state of σ_r. Note that $\mathsf{finalB}(s_f) = \cup_b s_f.\mathsf{w} = \mathrm{B}^i_f$ and hence (a) holds. We now establish (†) : $s_f.\mathsf{fc} = \{(C,\overline{x},a,\mathrm{B}^i_f) \mid (C,\overline{x},a) \in \mathsf{checksA}(s)\}$. 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, † follows from the construction of σ_r and the aforementioned invariants. (†) entails (b) and (d). (c) follows from (†) 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 $\mathrm{B}_f \cap_b (\cup_b s.\mathsf{wr}) = \mathrm{B}^i_f$ (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 A.1 and A.5 imply Theorem 4.2.

A.2 FA is Necessary for Verifiability

In Section 4.2.2, 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 [33] 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 amount 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 3.2 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.1 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 next argue 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. Now, 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 $\mathrm{B} \not\subseteq_b \mathrm{B}'$, 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[8]; 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: $(s'.\mathsf{nfc} \cup s'.\mathsf{fc}) = (s.\mathsf{nfc} \cup s.\mathsf{fc}) \backslash \{(C, \overline{x}, a, \mathrm{B})\} \cup \{(C, \overline{x}, a, \mathrm{B}_f)\}$, $\mathsf{finalB}^+(s') = \mathsf{finalB}^+(s)$, 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.,* $\mathrm{B} \not\subseteq_b \mathrm{B}_f$ where $\mathrm{B}_f \in \mathrm{B}(s.\mathsf{fc})$, then $P(S, \pi)$ does not provide verifiability for τ.

Theorem A.6. *Let $P(S, \pi)$ be a protocol that* can postpone reads *and such that $\pi \vdash$ 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 $\mathrm{B} \not\subseteq_b \mathrm{B}_f$, 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) $(s'.\mathsf{nfc} \cup s'.\mathsf{fc}) = (s.\mathsf{nfc} \cup s.\mathsf{fc}) \backslash \{(C, \overline{x}, a, \mathrm{B})\} \cup \{(C, \overline{x}, a, \mathrm{B}_f)\}$, (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 $\mathrm{B} \not\subseteq_b \mathrm{B}_f$, 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. □

A.3 Attacks

Continuing attack C.5 from Section 3.2.1. 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 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, breaking 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 [20]. 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 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

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

the TTs will sign this list and post it on the BB, the registrars could *theoretically* detect this attack by finding in the list a public credential they have not seen in the context of the block b'. As discussed in Section 3.2.3, we believe this detection remains purely theoretical until dedicated verifiability checks are specified that can detect such an attack in practice. Hence we suggest to mandate the registrars to check the final BB content and check that the list of public credentials for each block corresponds to voters eligible in the corresponding block.

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).

A.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* [35]. 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

B.1 Threshold γ

We claimed in Section 5.2 that the threshold γ in our protocol ensures that for any two readers, the intersection of the set 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 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 $|I_1 \cap I_2| > n - n_h$.

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.

B.2 Practical Considerations

We provide details for the practical considerations mentioned in Section 5.3.2.

B.2.1 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 [3]. Thus, FA, verifiability, and also privacy only hold when the platform is honest. Instead of trusting general purpose platforms, some e-voting protocols [4, 6, 31, 32, 45] 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 [26] (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 5.3.1, the UV and EV checks can be outsourced to external auditors and thus voters must only 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.2.2 Online BB Peers. We have argued 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 [18] 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 versus less costly architectures as future work.

B.2.3 Privacy. In the above solution, when voters retrieve a partial content, the proxy peer learns their ballot. However, we stress that this does not mean that the proxy peer learn its content, *i.e.,* 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

C.1 Existing BB Requirements in E-voting

As explained in Section 6, 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 $B_1 \sqsubseteq_b B_2$, 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 4.2.2). Moreover, all of the attacks presented in Section 3.2 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 5.2.2). 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.

C.2 Threshold Comparison to BFT algorithms

Our protocol requires (i) $n_h > 2(n - \gamma)$ while BFT consensus protocols run by the BB peers such (*e.g.,* [24]), 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 BFTs require trusting at least 9 out of 12 peers to be honest.

REFERENCES

[1] Ben Adida. 2008. Helios: Web-based Open-audit Voting. In *Conference on Security Symposium*. USENIX Association, 335–348. http://dl.acm.org/citation.cfm?id=1496711.1496734

[2] Michael R Clarkson Stephen Chong Andrew and C Myers. 2007. Civitas: Toward a Secure Voting System. *Computing and Information Science Technical Report TR* 2081 (2007).

[3] David Basin, Sasa Radomirovic, and Michael Schlaepfer. 2015. A Complete Characterization of Secure Human-Server Communication. In *Computer Security Foundations Symposium (CSF)*. IEEE, 199–213.

[4] David Basin, Saša Radomirovic, and Lara Schmid. 2018. Alethea: A Provably Secure Random Sample Voting Protocol. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 283–297.

[5] Belenios website 2020. https://belenios.loria.fr/admin. (2020). Accessed: 2020-01-18.

[6] Sergiu Bursuc, Constantin-Cătălin Drăgan, and Steve Kremer. 2019. 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)*. IEEE, Stockholm, Sweden.

[7] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Vanessa Teague, Roland Wen, Zhe Xia, and Sriramkrishnan Srinivasan. 2012. Using Prêt à Voter in Victorian State Elections. In *Electronic Voting Technology Workshop*.

[8] Craig Burton, Chris Culnane, and Steve Schneider. 2015. Secure and verifiable electronic voting in practice: the use of vvote in the victorian state election. *arXiv preprint arXiv:1504.07098* (2015).

[9] Dominique Cansell and Dominique Mery. 2006. Tutorial on the event-based B method. https://cel.archives-ouvertes.fr/inria-00092846. (2006).

[10] David Chaum, Aleks Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan Sherman, and Poorvi Vora. 2008. Scantegrity: End-to-End Voter-Verifiable

Optical-Scan Voting. *IEEE Security Privacy* 6, 3 (May 2008), 40–46. https://doi.org/10.1109/MSP.2008.70

[11] Sherman SM Chow, Joseph K Liu, and Duncan S Wong. 2008. Robust Receipt-Free Election System with Ballot Secrecy and Verifiability.. In *NDSS*, Vol. 8. 81–94.

[12] Michael R Clarkson, Stephen Chong, and Andrew C Myers. 2008. Civitas: Toward a Secure Voting System. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 354–368.

[13] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. 2018. Machine-checked proofs for electronic voting: privacy and verifiability for Belenios. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 298–312.

[14] Véronique Cortier, Fabienne Eigner, Steve Kremer, Matteo Maffei, and Cyrille Wiedling. 2015. Type-Based Verification of Electronic Voting Protocols. In *Principles of Security and Trust*. Springer Berlin Heidelberg, Berlin, Heidelberg, 303–323.

[15] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. 2019. BeleniosVS: Secrecy and Verifiability against a Corrupted Voting Device. In *32nd Computer Security Foundations Symposium (CSF)*. IEEE.

[16] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. 2014. Election Verifiability for Helios under Weaker Trust Assumptions. In *European Symposium on Research in Computer Security*. Springer, 327–344.

[17] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Mueller, and Tomasz Truderung. 2016. Sok: Verifiability notions for e-voting protocols. In *Symposium on Security and Privacy (SP)*. IEEE, 779–798.

[18] Véronique Cortier, Pierrick Gaudry, and Stephane Glondu. 2019. Belenios: a simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*. Springer, 214–238.

[19] Véronique Cortier and Joseph Lallemand. 2018. Voting: You Can't Have Privacy without Individual Verifiability. In *Conference on Computer and Communications Security*. ACM, 53–66.

[20] Véronique Cortier and Ben Smyth. 2013. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security* 21, 1 (2013), 89–148.

[21] Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. 2012. TLA+ proofs. In *International Symposium on Formal Methods*. Springer, 147–154.

[22] Chris Culnane, Peter YA Ryan, Steve A Schneider, and Vanessa Teague. 2015. vVote: A Verifiable Voting System. *ACM Transactions on Information and System Security (TISSEC)* 18, 1 (2015), 3:1–3:30.

[23] Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. 2014. vVote: a Verifiable Voting System (version 4.0). *CoRR* abs/1404.6822 (2014). arXiv:1404.6822 http://arxiv.org/abs/1404.6822

[24] Chris Culnane and Steve Schneider. 2014. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. *CoRR* abs/1401.4151 (2014). arXiv:1401.4151 http://arxiv.org/abs/1401.4151

[25] Denise Demirel, Maria Henning, Jeroen van de Graaf, Peter YA Ryan, and Johannes Buchmann. 2013. Prêt à Voter Providing Everlasting Privacy. In *E-Voting and Identify*. Springer Berlin Heidelberg, Berlin, Heidelberg, 156–175.

[26] Riccardo Focardi, Flaminia L Luccio, and Heider AM Wahsheh. 2018. Usable cryptographic QR codes. In *2018 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 1664–1669.

[27] Follow my vote company: Blockchain voting 2020. https://followmyvote.com/blockchain-voting-the-end-to-end-process/. (2020). Accessed: 2020-01-18.

[28] Gina Gallegos-Garcia, Vincenzo Iovino, Alfredo Rial, Peter B Roenne, and Peter YA Ryan. 2016. (Universal) Unconditional Verifiability in E-Voting without Trusted Parties. *arXiv preprint arXiv:1610.06343* (2016).

[29] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. 2014. Is Bitcoin a Decentralized Currency? *IEEE Security Privacy* 12, 3 (2014), 54–60. https://doi.org/10.1109/MSP.2014.49

[30] Stéphane Glondu. 2019. Belenios specification. *Version 1.6. http://www.belenios.org/specification.pdf* (2019).

[31] Gurchetan S Grewal, Mark D Ryan, Liqun Chen, and Michael R Clarkson. 2015. Du-vote: Remote electronic voting with untrusted computers. In *Computer Security Foundations Symposium*. IEEE, 155–169.

[32] Rolf Haenni and Reto E Koenig. 2014. Voting over the Internet on an Insecure Platform. In *Design, Development, and Use of Secure Electronic Voting Systems*. IGI Global, 62–75.

[33] Rolf Haenni, Reto E Koenig, Philipp Locher, and Eric Dubuis. 2017. CHVote System Specification (Version 3.0). *IACR Cryptology ePrint Archive* 2017 (2017), 325.

[34] Severin Hauser and Rolf Haenni. 2019. Modeling a Bulletin Board Service Based on Broadcast Channels with Memory. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 232–246.

[35] James Heather and David Lundin. 2009. The Append-Only Web Bulletin Board. In *Formal Aspects in Security and Trust*. Springer Berlin Heidelberg, Berlin, Heidelberg, 242–256.

[36] Sven Heiberg, Ivo Kubjas, Janno Siim, and Jan Willemson. 2018. On trade-offs of applying block chains for electronic voting bulletin boards. *E-Vote-ID 2018* (2018), 259.

[37] Helios website 2020. https://vote.heliosvoting.org/. (2020). Accessed: 2020-01-18.

[38] Martin Hirt and Kazue Sako. 2000. Efficient receipt-free voting based on homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 539–556.

[39] Ari Juels, Dario Catalano, and Markus Jakobsson. 2010. Coercion-resistant Electronic Elections. In *Towards Trustworthy Elections*. Springer-Verlag, 37–63.

[40] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. 2018. On the Security Properties of e-Voting Bulletin Boards. In *Security and Cryptography for Networks*. 505–523.

[41] Steve Kremer, Mark Ryan, and Ben Smyth. 2010. Election Verifiability in Electronic Voting Protocols. In *Computer Security – ESORICS 2010*. Springer Berlin Heidelberg, Berlin, Heidelberg, 389–404.

[42] Ralf Kusters, Tomasz Truderung, and Andreas Vogt. 2012. Clash attacks on the verifiability of e-voting systems. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 395–409.

[43] Leslie Lamport. 2015. The TLA+ hyperbook. (2015). https://lamport.azurewebsites.net/tla/hyperbook.html.

[44] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. 2017. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*. Springer, 357–375.

[45] Stephan Neumann and Melanie Volkamer. 2012. Civitas and the real world: problems and solutions from a practical point of view. In *International Conference on Availability, Reliability and Security*. IEEE, 180–185.

[46] Peter YA Ryan, David Bismark, James A Heather, Steve A Schneider, and Zhe Xia. 2009. Prêt à Voter: a Voter-Verifiable Voting System. *Transactions on Information Forensics and Security* 4, 4 (2009), 662–673.

[47] Peter YA Ryan, Peter B Rønne, and Vincenzo Iovino. 2016. Selene: Voting with transparent verifiability and coercion-mitigation. In *International Conference on Financial Cryptography and Data Security*. Springer, 176–192.

[48] Pavel Tarasov and Hitesh Tewari. 2017. Internet Voting Using Zcash. *IACR Cryptology ePrint Archive* 2017 (2017), 585.

[49] TLA+ proofs 2020. https://drive.google.com/drive/folders/1SDTTDCUd1p11UwBYHrn4nGTd-TGTQcp7. (2020). Accessed: 2020-01-18.

[50] Nan Yang and Jeremy Clark. 2017. Practical governmental voting with unconditional integrity and privacy. In *International Conference on Financial Cryptography and Data Security*. Springer, 434–449.

[51] Bin Yu, Joseph K Liu, Amin Sakzad, Surya Nepal, Ron Steinfeld, Paul Rimba, and Man Ho Au. 2018. Platform-independent secure blockchain-based voting system. In *International Conference on Information Security*. Springer, 369–386.

[52] Zhichao Zhao and T-H Hubert Chan. 2015. How to vote privately using bitcoin. In *International Conference on Information and Communications Security*. Springer, 82–96.