Fifty Shades of Ballot Privacy: Privacy against a Malicious Board

Véronique Cortier
CNRS, Loria
veronique.cortier@loria.fr

Joseph Lallemand
Inria, Loria & ETH Zürich
joseph.lallemand@inf.ethz.ch

Bogdan Warinschi
University of Bristol & Dfinity
bogdan.warinschi@dfinity.org

***Abstract*—We propose a framework for the analysis of electronic voting schemes in the presence of malicious bulletin boards. We identify a spectrum of notions where the adversary is allowed to tamper with the bulletin board in ways that reflect practical deployment and usage considerations. To clarify the security guarantees provided by the different notions we establish a relation with simulation-based security with respect to a family of ideal functionalities. The ideal functionalities make clear the set of authorised attacker capabilities which makes it easier to understand and compare the associated levels of security. As an application, we study three protocols of the literature (Helios, Belenios, and Civitas) and identify the different levels of privacy they offer.**

1. Introduction

Electronic voting aims to achieve the same properties as traditional paper based voting. Even when voters vote from their home, they should be given the same guarantees, without having to trust the election authorities, the voting infrastructure, and/or the Internet network. A key property is *vote privacy*, also called *ballot privacy*: no one should know how I voted. Many schemes have been designed to achieve vote privacy under various trust assumptions (*e.g.* Helios [1], Civitas [2], Selene [3], sElect [4], or Alethea [5] to cite a few). The typical strategy is to encrypt the votes under a key for which the corresponding decryption key is split among several authorities – at least a certain number of authorities are required to decrypt and tally. The motivation for this design is that in this setting the voting server which, among other functions, maintains the public bulletin board, does not need to be trusted.

It has recently been observed *e.g.* in [6], [7], [8] that this trust assumption is not appropriately captured by existing security definitions. In brief, existing definitions (*e.g.* [9], [10], [11]) consider a game where the adversary controls the votes cast by honest parties but cannot control the resulting ballots: these get placed on the bulletin board before it is tallied and cannot ever be modified or removed. In other words, current notions allow to prove security of schemes only under the (unrealistic) assumption that the bulletin board contains all of the submitted honest votes, in the order in which they were submitted; no honest ballot is dropped or modified. This is actually a much stronger assumption than simply requiring a public, append-only bulletin board, as often mentioned as an (informal) assumption in many papers.

This gap between security goals and security definitions has recently been confirmed by Roenne [12] in the case of Helios [1], a popular voting scheme. Roenne's attack shows that an attacker can break privacy as soon as he can tamper with the communications between the voters and the bulletin board. In short, the attacker blocks and learns Alice's first encrypted ballot, then Alice votes again (from her point of view, something went wrong, so she simply starts again) and then the attacker casts Alice's first ballot in his own name. Then looking at the result, the attacker learns information about Alice's vote. For example, in the extreme case of only two voters, the attacker learns her vote, even if Alice and Bob voted differently. This attack works even in the presence of a public, append-only bulletin board. Moreover, it seems impossible to prevent the attack even if voters and external auditors carry out additional checks (*e.g.* forbidding duplicate ballots). Even detecting the attack would require unrealistic countermeasures where every voter carefully records all of her ballots, even the ones that failed to reach the ballot box.

To fill this gap, one line of work [13], [6], [11], [14] studies how to implement a bulletin board that is strong enough to prevent the adversary from tampering with the ballots. However, whether or not such implementations are sufficient to meet the (very strong) assumptions of existing privacy proofs has not yet been formally studied. Moreover, some implementations require a heavy infrastructure, not necessarily aligned with the designers' goals.

In this work, we pursue an alternative approach where our goal is to define vote privacy under the sole assumption that all participants can see the same board; the content of the board is under the complete control of the adversary. While this consistent view assumption is not necessarily easy to achieve in practice, it is significantly less onerous than assuming that absolutely all honest ballots reach the ballot box, before the adversary even gets to see them, as assumed so far in privacy definitions.

To contextualise some of our design choices, it is useful to discuss the security of Helios against an adversary who can control the bulletin board. Recall that in Helios voters encrypt their votes and send their encrypted ballots to a bulletin board, that displays them. The tally is done through mixnets or using the homomorphic property of the encryption. Importantly, voters can and should check that their ballot

appears on the bulletin board.

How much security does Helios provide when the board which collects the votes behaves maliciously? The answer to this question strongly depends on the behaviour of the users involved in the election.

1. The strongest guarantees are offered when all of the honest voters vote and check that their vote appears on the bulletin board. In this case the result of the election contains all of the honest votes plus at most as many votes as the number of voters controlled by the adversary.

2. Assume now that not all honest voters vote, but those who vote, also check that their ballot appears on the board. Over the previous scenario, the security of Helios decreases. Indeed a malicious board may use absentee voters to place ballots of her choice. So the privacy of the election is as good as what an attacker can learn from a result formed from honest voters who did vote and any choice of votes from the remaining voters (dishonest or not).

3. Finally, the most common scenario is that not all honest voters vote and only a fraction of them actually conduct the suggested verification steps. In this case the security of Helios decreases even further. A malicious board may now selectively remove ballots that have been cast by voters who do not check. Hence a malicious board has now even more control on the result which, in turn, may leak more information about the honest votes since the board may contain fewer votes (and more of which are selected by the adversary).

The examples above make it clear that it is difficult to settle on a unique definition of privacy: each scenario is, at least in theory, possible and each corresponds to a different level of guarantees. The strongest privacy level for a voter's ballot is obtained when her ballot is *always* tallied together with all of the other honest votes. For Helios, such a level of security corresponds to the first case above but can only be provided under the unrealistic assumption that all voters verify that their vote has been cast. Alternatively, we could consider a weaker variant, where the adversary can remove some (prespecified) number of ballots. This attacker corresponds to case 3 where only some voters check that their ballot has been recorded correctly. A benefit of this relaxation is that it would allow to study the security of more schemes: while the attacker may remove votes, we would like to understand how harmful these actions are with respect to the privacy of the remaining votes.

However, if we settled only on the weak variant, we would not be able to express that security guarantees vary considerably between different usage scenarios of *the same* scheme, as our three examples above show. Moreover, while in the above examples we only identified three relevant levels of privacy for Helios, we note that other schemes may require even more different levels.

A spectrum of privacy definitions is also useful to compare *different* schemes. A "black or white" definition would declare a set of voting schemes secure while the rest would be deemed insecure. Instead, it is clearly more useful to spell out conditions under which some scheme is "more secure" than another one. For example, in Civitas [2], due to the use of credentials, a malicious server may drop ballots from honest voters who do not check but cannot replace them with arbitrary ballots, like in Helios. In that respect, Civitas is more secure since the adversary can infer less from the result. This is a more precise analysis than declaring Civitas private and Helios insecure.

1.1. Our contributions

We make four contributions which address the challenges outlined above. Throughout, unlike most existing privacy definitions, we assume a malicious voting server. Furthermore, we assume that the adversary (an arbitrary probabilistic polynomial time algorithm) fully controls the network, and a set of dishonest voters, in addition of course to the voting server. We highlight however that our privacy definition still assumes a trusted setup and a trusted tally. While verifiability notions have already been studied and formally defined against a dishonest talliers for example, this remains unexplored regarding privacy. We also leave it for future work (one issue at a time!).

Game based security. Our first contribution is *a family* of rigorous game-based definitions for ballot privacy against malicious bulletin boards. The general idea behind existing game-based definitions is that the adversary has to distinguish between two situations. The first considers the case where honest voters submit ballots containing votes selected by the adversary. This is the "real world". In the second scenario the adversary does not see the real ballots. Instead, honest voters submit "fake" ballots corresponding to other votes, also chosen by the adversary. If no adversary can tell the two worlds apart, even given the result of the election, then no information is leaked about the underlying votes. However, such a definition would immediately break if we freely tally the board seen by the adversary: the result immediately reveals whether he is in the real or the fake world. To tackle this issue, two main approaches have been proposed. One approach rooted in the seminal work of Benaloh [9] is to only compute the tally if the multiset of honest votes in the real world coincides with the multiset of honest votes in the fake world. This restricts the applicability of the definition only to elections where the result of the election is the multiset of votes. A recent approach which allows for the study of arbitrary counting functions (*e.g.* STV or Condorcet) builds upon the security notion BPRIV introduced in [15]. Here, the tally is always computed on the real board, even when the adversary gets to see the fake ballots. Additional properties identified in [10] then guarantee that the tally does not leak extra information by enforcing that the tally always corresponds to the content of the board.

The assumption that the bulletin board itself is dishonest adds significant complications which probably also explains why most existing vote privacy definitions assume an honest voting server. Both definitional approaches outlined above break immediately when considering a dishonest server and need to be "patched". To be able to apply the restriction in Benaloh's approach, we need to determine which honest

ballots have indeed reached the ballot box, which is made difficult by the fact that the adversary could have altered them. Similarly, in the BPRIV approach, we need to return to the adversary the tally of the honest votes even when he sees fake ballots. However, an adversary who fully controls the ballot box may tamper or drop all ballots submitted by the honest parties. Moreover, as discussed above, we need to distinguish between the case where removing honest ballots is an actual attack or corresponds to some actions which are in fact allowed by the scheme.

Since building on top of the Benaloh style definition is not necessarily simpler and limits the class of schemes one can analyse, we work with the more general BPRIV framework. Our solution to the technical problems outlined above is to define the security only for schemes where we can somehow detect how an adversary tampers with the bulletin board. To this end, we demand the existence of a *recovery* algorithm which, given the set of honest ballots and the board to be tallied, can detect *how* the adversary has modified the ballots issued by the honest users. The output of the recovery algorithm can be thought of as a small *tampering program*, written in a small programming language with commands that act on the bulletin board (*e.g.* delete honest votes, modify votes, re-order votes, *etc.*). This recovery algorithm is a parameter of the security definition and can be used to determine which honest votes to tally for the adversary. Of course, the more actions the recovery algorithm allows, the less security guarantees we get. We emphasise that this detection procedure is an artefact of our modelling approach, and not a procedure which could, for example, be run during the execution of the protocol to detect tampering.

Relation with simulation-based security. Next, we validate our game-based definitions of security by relating them with simulation-based notions. In this latter definitional approach, security is defined with respect to some ideal functionality that captures a small set of possible behaviours, corresponding to a very abstract model of the system. Functionalities capture security somewhat more directly, which facilitates understanding of some of their associated security guarantees.

We remark that our goal explicitly being to establish the relation with such a simulation-based definition, is an additional reason why we build on top of the BPRIV framework rather than using Benaloh's approach. Very roughly, Benaloh's definition does not seem to allow to construct a simulator which can simulate on the fly a fake board towards an adversary: the global consistency requirement between the subtally of the real votes seems to preclude an on-line simulator which can fake a board, since the simulator would learn *at a later point only* the result which corresponds to the cast votes.

The typical definition for simulation based vote privacy involves a functionality which collects the list of votes of all parties (honest and corrupt) and simply returns the result of the election determined by the list. To capture the setting where an adversary can to some extent tamper with the bulletin board (and therefore the list of votes that is tallied) we modify the ideal functionality to reflect this adversarial ability. We now give a high level (and imprecise) sketch of how we proceed. Similarly to how our game-based notion is parametrised by the recovery algorithm, the functionality is parametrised by a small tampering "programming" language P – think about P as containing commands that tamper with the board (*e.g.* delete ballot(i), insert ballot(b)). After it collects the votes, but before it returns the result, the functionality allows the adversary to tamper with the votes list via an arbitrary program f which uses the commands in P (*e.g.* delete ballot(1); delete ballot(4); delete ballot(5)) before returning the tally[1].

We can then establish a link between security *w.r.t.* mb-BPRIV parametrised by recovery algorithm f and idealised security with respect to an ideal functionality parametrised by tampering language P. Provided that f and P are compatible (roughly: they allow the same commands on the list of ballots/votes), then any scheme which is mb-BPRIV secure and strongly consistent is secure with respect to the ideal functionality. Strong consistency, a notion introduced in [10], demands that the tally reveals only the desired result function on the votes (and no additional information).

Relation with verifiability. Our definitions exhibit a subtle interplay between verifiability and privacy which is worth discussing. Our ideal functionalities reflect different adversarial capabilities and restrictions to modify honest votes. Is this a privacy or a verifiability property? Intuitively, ballot privacy says that the adversary should not learn more information about the votes than the result itself. Hence, whether or not the adversary can remove or alter honest votes, or add more votes (all of which are actions which verifiability could/should prevent), gives more control to the adversary over the result and therefore with the level of privacy offered by the voting scheme. Similarly to [8], we can show that ballot privacy entails some form of verifiability, for a spectrum of verifiability notions that echoes our spectrum of privacy levels.

Case studies. As applications of our definitional contributions we study the security of three standard protocols when the adversary can control the bulletin board: Helios [8], Belenios [16], and Civitas [2]. For each protocol in turn we identify which ideal functionalities they achieve and under which trust assumptions. In particular, we highlight that Civitas is the only scheme among the three that guarantees that ballots cannot be reordered, even by a malicious board.

1.2. Related work

The first game-based definition which considers a malicious board has been proposed by Bernhard and Smyth [7]. Their definition extends Benaloh's approach [9]. The adversary submits a board and the tally is performed only if the ballots on the board that come from honest voters

1. Actually, we capture validity of tampering functions via predicates which do more than only syntactic checks.

are such that the subtally does not differ in the "left" and "right" worlds. This somehow corresponds to one possible instance of our recovery algorithm in mb-BPRIV, where the attacker may remove any honest vote and add an arbitrary number of votes, independently of whether honest voters do check their ballot and independently of the number of dishonest voters. Note that [7] requires that ballots cannot be modified at all (*e.g.* they cannot contain a tag such as the date). Perhaps the most important technical difference from our approach is that Benaloh's definitional approach does not seem to allow a formal link with a simulation-based notion of security. Furthermore, this approach assumes that the counting function admits partial tally, which discards many modern counting functions such as Condorcet or STV. The shortcomings that stem from the use of Benaloh's approach are also shared by [8]. This definition can be again seen as an extension of Benaloh's definition but which assumes that *all* honest voters check that their ballot appears on the bulletin board.

Recently, Bursuc, Dragan and Kremer [17] have studied the security of encryption schemes where ballots can be partially modified, for example by a malicious device. They propose a variant of BPRIV that accounts for such behaviours. The case of a malicious board corresponds to the case where ballots can be fully modified. Then for malicious boards, vote privacy as defined in [17] can be seen as an instance of mb-BPRIV where the recovery algorithm lets the adversary tamper arbitrarily with the honest ballots. However, in such a case, all schemes would be declared insecure. So the model of [17] does not seem suitable to reason about malicious boards in general. Instead, it addresses a class of schemes where security is due to the part of the ballot that is securely transmitted to the (honest) board, despite the adversary tampering with the other parts of the ballot.

The approach we take in this paper is to understand the level of security offered by schemes when faced with an adversary who is allowed to tamper with the bulletin board. A different approach adopted by a series of recent works [13], [6], [11] aims to ensure that such tampering is not possible. This line of work nicely complements our approach – in particular, our mb-BPRIV definition assumes that voters all see the same board. On the other hand, [13], [6], [11] propose systems that use a distributed algorithm to provide a consistent view of the board among the voters and the auditors. [14] formalises generic conditions on such algorithms that are sufficient to guarantee this consistency, and shows how to implement a bulletin board that achieves their conditions. In particular, their board guarantees that honest ballots are not removed nor reordered. In other words, they provide sufficient conditions to realise one (strong) security level of the boards we consider, at the price of multiple independent servers. In contrast, we provide a ballot privacy definition that can cope with various bulletin boards, including weaker ones where the attacker could tamper with the ballot box (*e.g.* removing some honest ballots, reordering the ballots). We do not study how to realise these different boards.

2. Background

In this section we recall some terminology from existing literature, and fix some assumptions which we will use throughout the paper. We consider a finite set $\mathcal{I} = \mathsf{H} \cup \mathsf{D}$ of voter identities, partitioned into the sets H and D of honest and dishonest voters. H is further partitioned into sets $\mathsf{H}_{\mathsf{check}}$ and $\mathsf{H}_{\overline{\mathsf{check}}}$, meant to contain the identities of voters who verify their vote (resp. do not verify).

We study schemes for which the adversary can legally tamper with the bulletin board (NB: throughout the paper we use bulletin board and ballot box interchangeably.) For concreteness, we make a mild assumption regarding the format the bulletin board takes.

Definition 1 (Bulletin board)**.** *A* bulletin board BB *is a list of ballots of the form* (p, b) *where* p *is called a* public credential *and* b *is a ciphertext.* $\mathsf{BB}[j]$ *denotes the* j*th element of* BB*.*

We will also call extended bulletin board *a board where elements are associated to an identity, i.e. a list of elements of the form* $(id, (p, b))$*.*

Definition 2 (Voting scheme)**.** *A voting scheme consists of seven algorithms:*

$$\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Pub}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Tally}, \mathsf{Verify}).$$

- $\mathsf{Setup}(1^\lambda)$ *computes a pair of* election keys $(\mathsf{pk}, \mathsf{sk})$ *given a security parameter* λ*.*
- $\mathsf{Register}(1^\lambda, id)$ *generates a* private credential c *for voter* id *and stores the correspondence* (id, c) *in a list* U*, used for modelling purposes.*
- $\mathsf{Pub}(c)$ *returns the public credential associated with a credential* c*.*
- $\mathsf{Vote}(\mathsf{pk}, id, c, v)$ *constructs a ballot* (p, b) *for user* id *with private credential* c*, containing vote* v*, using the public election key* pk*. It also returns a state to the voter, that models what a voter should record, e.g. her ballot. One can think about this state as any information a voter would need to record, e.g. to verify if the ballot has been cast.*
- $\mathsf{Valid}(\mathsf{BB}, \mathsf{pk})$ *checks that the board* BB *is valid.*
- $\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$ *uses the board* BB *and the secret election key* sk *to compute the result* r *of the election, and potentially proofs* Π *of good tallying.*
- $\mathsf{Verify}(id, s, \mathsf{BB})$ *represents the checks a voter* id*, with local state* s*, should perform on a board* BB *to ensure her vote is counted.*

Counting functions are the functions which calculate the result of an election. For example, the result of an election can be the sum of the votes for each candidate, or the multiset of votes, or the result of more complex voting methods such as Condorcet or Single Transferable Vote.

Definition 3 (Counting function)**.** *A* counting function ρ *is a mapping that takes a sequence* S *of pairs* (id, v)*, where* $id \in \mathcal{I}$ *and* v *is a vote, and returns the result of tallying the votes in* S*. It may use the* id*s to apply a revote policy.*

We assume a special value $\perp$ *that represents the case of an invalid vote (e.g. obtained by decrypting a ballot that was*

incorrectly generated). We require that counting functions ignore this value, i.e. that for all l, l', $\rho(l||\perp||l') = \rho(l||l')$.

3. Game-based Security

In this section we present our game-based notion for vote privacy. Namely, *ballot privacy* ensures that ballots do not reveal information about the underlying votes. We will show that ballot privacy implies simulation-based security, provided that the scheme is additionally strongly consistent, that is that the tally function behaves as counting the votes extracted from the ballots.

3.1. mb-BPRIV

Ballot privacy has been proposed by Bernhard et al [10]. It captures the idea that ballots themselves do not reveal information about the underlying votes (even after tallying). That notion models an honest ballot box whereas we consider a malicious one. To distinguish between the two notions we refer to the existing one as hb-BPRIV security and to the notion we introduce here as mb-BPRIV. We start with a high level discussion of the notion which we introduce, provide a formal definition and then discuss its salient features over those of hb-BPRIV.

We consider a game which pits an adversary against a voting scheme. The adversary has partial information about honest users' votes: for each such user the adversary selects a left-or-right challenge consisting of two votes v_0 and v_1. The game computes the ballots corresponding to the votes but returns to the adversary the ballot which corresponds to v_β, for some hidden bit β which the adversary needs to determine. However, the game keeps track of two bulletin boards BB_0 and BB_1 (the ordered list of ballots calculated in response to the adversary's queries) – the adversary sees, essentially, BB_β. The adversary then creates a public bulletin board BB, by using the honest votes and arbitrary other votes it creates.

The key aspect of the definition is how the game computes the tally it returns to the adversary. When the adversary is in the world where he sees BB_0, the game simply tallies BB, the board which the adversary returned.

When the adversary is in the world where he sees BB_1 (so where BB is calculated using BB_1) we need to determine *how* the adversary manipulated the votes on BB_1 to produce the board he returned to be tallied. To define security we demand that it should be possible to (efficiently) determine which of the honest ballots have been cast, on which position on the bulletin board, and which ones have been removed. Once this transformation is determined, the game applies it to BB_0, tallies the resulting board and returns the result to the adversary. We explain a bit later in the paper how an insecure scheme would allow a distinguishing attack in the game we outlined above.

Technically, we represent the transformation which describes how the adversary constructs BB from BB_1 as a *selection function*, and we formalise the process of recovering this transformation as a *recovery* algorithm.

Definition 4 (Selection function)**.** *For $m, n \geq 1$, a* selection function for m, n *is any mapping*

$$\pi : [\![1, n]\!] \longrightarrow [\![1, m]\!] \cup (\{0,1\}^* \times \{0,1\}^*)$$

Intuitively, π represents the process used by an adversary to construct a bulletin board BB of n ballots from a given board BB_1 of m ballots. For $i \in [\![1, n]\!]$, $\pi(i)$ indicates how to construct $\mathsf{BB}[i]$:

- $\pi(i) = j \in [\![1, m]\!]$ means this element is the jth from BB_1;
- $\pi(i) = (p, b)$ means that this element is (p, b).

A bit more formally:

Definition 5 (Applying a selection function to a board)**.** *Consider a selection function π for $m, n \geq 1$. The function $\overline{\pi}$ associated to π maps an extended board BB_0 of length m to a board $\overline{\pi}(\mathsf{BB}_0)$ of length n such that for any $j \in [\![1, n]\!]$,*

$$\overline{\pi}(\mathsf{BB}_0)[j] = \begin{cases} (p, b) \text{ if } \pi(j) = i \text{ and } \mathsf{BB}_0[i] = (id, (p, b)) \\ (p, b) \text{ if } \pi(j) = (p, b) \end{cases}.$$

Note that we could consider a more general selection function that applies an arbitrary transformation to honest ballots (*e.g.* shifting a vote). We consider this version for simplicity and also because it should cover most of the reasonable needs (it is unlikely that shifting votes should be considered as acceptable).

The "recovery" algorithm which recovers the selection function used by the adversary takes as input two boards and some additional data d (intuitively, this piece of data contains the link between voter identities and public credentials).

Definition 6 (Recovery algorithm)**.** *We call* recovery algorithm *any algorithm* RECOVER *that, given a board* BB*, an extended board* BB_1*, and some additional data d as input, returns a selection function for $|\mathsf{BB}_1|$, n for some n.*

We discuss the role of RECOVER and how it can be interpreted after we provide our formal definition for mb-BPRIV.

The following definition formalises ballot privacy of some scheme $\mathcal{V}$ in a setting where the adversary $\mathcal{A}$ controls the voting server hence the ballot box. Recall that we consider some fixed set of voter identities $\mathcal{I}$ partitioned between two sets H and D of honest and dishonest voters. We also assume that some subset $\mathsf{H}_{\mathsf{check}}$ of H of users perform whatever checks the scheme expects to be executed before the tally is performed. The execution described in Figure 1 starts with the generation of a public key for the election and its associated secret key (to be used for tallying). Next, a number of voters from an arbitrary set $\mathcal{I}$ are registered. We keep this aspect of the execution fairly abstract: we assume a registration algorithm/protocol is executed for each user $id \in \mathcal{I}$ and we only record the secret credential c and its associated public credential $\mathsf{Pub}(c)$. We use respectively arrays U and PU to record these. We also use array CU to record the secret credentials for some (arbitrary) set of dishonest users D.

The adversary gets as input pk, CU and PU. It also gets access to a left-right voting oracle. On input an identity id and two potential votes v_0 and v_1, the oracle computes two

ballots for id, one for each adversarially selected vote. It records the first ballot in list BB_0 and the second in list BB_1. We remark that we model a voting algorithm which is stateful. This is a necessary feature if one wants, as we do, to consider voters who perform additional actions after they have voted (*e.g.* checking that their ballot has been cast). For each user, we store the resulting state (for both worlds) in arrays V_0 and V_1, respectively. This phase corresponds to the voting phase where the users submit their ballots. Then, the adversary prepares a bulletin board BB which it would like to be tallied. If the bulletin board does not pass the validity test then tally does not occur and the adversary needs to output his guess at this point.

Otherwise, the adversary gets control over the users who check via the oracle $\mathcal{O}\mathsf{verify}$, to which it submits arbitrary identities. The oracle records the set of users who have checked in variable $\mathsf{Checked}$ and the set of users for which the check was successful in variable Happy.

If all of the voters who should check do check successfully, then the adversary gets to see the tally of the election. Otherwise the adversary must produce his guess without seeing the tally.

Finally, one of the salient aspects of our definition is how the experiment calculates the tally. In the real execution (*i.e.* $\beta = 0$) the tally is simply executed on BB. In the fake execution (*i.e.* $\beta = 1$) the tally first employs the RECOVER algorithm which parametrises the game to determine how the adversary has tampered with the votes it has seen (*i.e.* BB_1) to produce the board it asks to be tallied. Then the game applies the transformation obtained this way to BB_0. The resulting board is tallied and the result, together with a simulated proof, is returned to the adversary.

Definition 7 (mb-BPRIV *w.r.t.* a recovery algorithm)**.** *Let $\mathcal{V}$ be a voting scheme, and* RECOVER *a recovery algorithm. Consider game $\mathsf{Exp}^{\textit{mb-BPRIV},\textsc{Recover},\beta}_{\mathcal{A},\mathcal{V},\mathsf{SimProof}}$ defined in Figure 1. $\mathcal{V}$ satisfies mb-BPRIV w.r.t.* RECOVER *if there exists a simulator* SimProof *such that for any polynomial adversary $\mathcal{A}$,*

$$|\mathbb{P}(\mathsf{Exp}^{\textit{mb-BPRIV},\textsc{Recover},0}_{\mathcal{A},\mathcal{V},\mathsf{SimProof}}(\lambda)=1) - \mathbb{P}(\mathsf{Exp}^{\textit{mb-BPRIV},\textsc{Recover},1}_{\mathcal{A},\mathcal{V},\mathsf{SimProof}}(\lambda)=1)|$$

is negligible in λ.

Our definition is parametrised by a recovery algorithm, which is a rather non-standard feature. We explain the role it plays through an example. One way to think about the recovery algorithm is that it aims to detect the (legal) actions which the adversary took when tampering with the board. For a secure scheme, it should be possible to understand how each ballot on the bulletin board to be tallied has been created, *i.e.* was it submitted by an honest user, was it created by the adversary, was it submitted by an honest user but modified by the adversary, according to what is considered to be acceptable.

The following example sheds some light on the role played by the recovery algorithm in our security definition. Similarly to the Helios scheme, assume a scheme where copying a ballot and submitting it in the name of another voter is possible. There are already two possibilities. Either this is completely allowed by the scheme (scheme A, no weeding is performed) or such duplicate ballots should be weeded out and therefore a board with duplicate ballots would be rejected (scheme B). Now, even when no weeding is performed, such a weakness (letting the adversary copy votes) may be well identified and accepted by the users (case 1) or not accepted (case 2). To win mb-BPRIV, the adversary can proceed as follows. First, it submits $(id, 1, 0)$ to the voting oracle. The game calculates b_0 and b_1 and returns b_β to the adversary. The adversary returns for tallying a board containing b_β, b'_β, where the adversary turned b_β into an equivalent ballot b'_β, which contains the same vote as b_β. In the left world, the tally returns 2. What happens in the right world depends on which recovery algorithm we consider. If we pick a recovery algorithm that does not detect that b'_β is a duplicate of b_β, then it interprets b'_β as a fresh adversarial ballot. The board which will be tallied is then b_0, b'_1, so the result would be 1. The scheme would thus be insecure for this recovery algorithm. This corresponds to case 2: ballot copying has not been identified as an acceptable behaviour and therefore this is an attack. Conversely, if we pick a recovery that detects that b'_1 is a copy of b_1, then the board submitted to tally would be b_0, b'_0, where b'_0 is obtained from b_0 using the same action the adversary used on b_1, so the result would also be 2, and it would not help the adversary to distinguish. The scheme would then be mb-BPRIV for this recovery algorithm which detects the copying action. This corresponds to case 1: the users are aware that ballot copying is possible and this is not considered an issue. Yet, the users would like to perform a privacy analysis: are there other behaviours that may leak some information on the votes?

We see here that the RECOVER we choose determines the security level provided by mb-BPRIV. Our example where ballots can be copied would be declared insecure for a RECOVER that does not detect copied ballots, as this RECOVER is unable to detect what the adversary did, but secure for a RECOVER that does detect copies. The second RECOVER detects more possible actions from the adversary, and hence allows the adversary to do more without breaking mb-BPRIV: so this variant of recovery algorithm yields weaker security guarantees. More generally, the transformations which RECOVER detects limit what an adversary is allowed to do without breaking mb-BPRIV. An adversary that can perform some actions that RECOVER does not detect will break mb-BPRIV, as in the example. Thus, proving a given voting scheme mb-BPRIV for a RECOVER that detects less behaviours from the attacker gives stronger security guarantees: it means that no adversary can have such behaviours, as otherwise mb-BPRIV would break.

Finally, it is instructive to consider the case where the scheme itself detects and discards such copies (to offer better privacy). This is scheme B. Then, even if we pick the first recovery algorithm, the one that does not detect that b'_β is a duplicate of b_β, the scheme would satisfy mb-BPRIV. Indeed, as the scheme itself detects and prevents the adversary from copying ballots, there is no need for RECOVER to detect

$\mathsf{Exp}^{\mathsf{mb\text{-}BPRIV},\textsc{Recover},\beta}_{\mathcal{A},\mathcal{V},\mathsf{SimProof}}(\lambda)$

$\mathsf{V}_0, \mathsf{V}_1, \mathsf{Checked}, \mathsf{Happy} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
for all $id \in \mathcal{I}$ **do**
  $c \leftarrow \mathsf{Register}(1^\lambda, id)$
  $\mathsf{U}[id] \leftarrow c, \mathsf{PU}[id] \leftarrow \mathsf{Pub}(c)$
for all $id \in \mathsf{D}$ **do** $\mathsf{CU}[id] \leftarrow \mathsf{U}[id]$
$\mathsf{BB} \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{voteLR}}(\mathsf{pk}, \mathsf{CU}, \mathsf{PU})$
if $\mathsf{H}_{\mathsf{check}} \not\subseteq \mathsf{V}_0, \mathsf{V}_1$ **then return** $\bot$;
if $\mathsf{Valid}(\mathsf{BB}, \mathsf{pk}) = \bot$ **then**
  $d \leftarrow \mathcal{A}()$; **output** d
$\mathcal{A}^{\mathcal{O}\mathsf{verify}_{\mathsf{BB}}}()$
if $\mathsf{H}_{\mathsf{check}} \not\subseteq \mathsf{Checked}$ **then return** $\bot$
if $\mathsf{H}_{\mathsf{check}} \not\subseteq \mathsf{Happy}$ **then** $d \leftarrow \mathcal{A}()$;
if $\mathsf{H}_{\mathsf{check}} \subseteq \mathsf{Happy}$ **then**
  $d \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{tally}_{\mathsf{BB},\mathsf{BB}_0,\mathsf{BB}_1}}()$
output d.

$\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$

if $id \notin \mathsf{H}$ **then return** $\bot$
$(p_0, b_0, s_0) \leftarrow \mathsf{Vote}(\mathsf{pk}, id, \mathsf{U}[id], v_0)$
$(p_1, b_1, s_1) \leftarrow \mathsf{Vote}(\mathsf{pk}, id, \mathsf{U}[id], v_1)$
$\mathsf{V}_0[id] \leftarrow s_0, \mathsf{V}_1[id] \leftarrow s_1$
$\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \parallel (id, (p_0, b_0))$
$\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \parallel (id, (p_1, b_1))$
return (p_β, b_β).

$\mathcal{O}\mathsf{verify}_{\mathsf{BB}}(id)$ for $id \in \mathsf{H}_{\mathsf{check}}$

$\mathsf{Checked} \leftarrow \mathsf{Checked} \cup \{id\}$
if $\mathsf{Verify}(id, \mathsf{V}_\beta[id], \mathsf{BB}) = \top$ **then**
  $\mathsf{Happy} \leftarrow \mathsf{Happy} \cup \{id\}$

$\mathcal{O}\mathsf{tally}_{\mathsf{BB},\mathsf{BB}_0,\mathsf{BB}_1}()$ for $\beta = 0$

$(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$
return (r, Π)

$\mathcal{O}\mathsf{tally}_{\mathsf{BB},\mathsf{BB}_0,\mathsf{BB}_1}()$ for $\beta = 1$

$\pi \leftarrow \textsc{Recover}_{\mathsf{U}}(\mathsf{BB}_1, \mathsf{BB})$
$\mathsf{BB}' \leftarrow \overline{\pi}(\mathsf{BB}_0)$
$(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}', \mathsf{sk})$
$\Pi' \leftarrow \mathsf{SimProof}(\mathsf{BB}, r)$
return (r, Π')

Figure 1. The mb-BPRIV game.

this behaviour. Just as intuition should say, such a scheme would be mb-BPRIV with a recovery algorithm that detects less, which ensures a stronger level of security.

3.2. Instantiations of mb-BPRIV

In this section we describe three instantiations of mb-BPRIV with recovery algorithms relevant for the schemes we study in this paper. Recall that the recovery algorithm aims to determine how the adversary tampered with the board. For clarity, in our examples we indicate in the indices of the recovery algorithms the actions which we expect each recovery algorithm to be able to detect. For example, $\textsc{Recover}^{\mathsf{del},\mathsf{reorder}}$ would be expected to detect, for each vote in turn, if the adversary has blocked it from appearing in the final tally, or if it has changed the order in which it was cast. In that case, such actions are deemed acceptable. We detail this recovery algorithm and discuss how it works. We then provide two variations: one which adds an additional capability to the adversary (and thus makes the associated mb-BPRIV variant weaker) and one which restricts the power of the recovery algorithm (and thus makes the associated variant stronger).

For each instantiation we informally describe the power we expect the adversary to have and then give a matching recovery algorithm. Our instantiations assume an efficient algorithm that can extract an identity and a vote from each ballot. Formally, we assume two functions $\mathsf{extract}_{\mathsf{id}}$ and $\mathsf{extract}_{\mathsf{v}}$ such that for any (p, b) generated by $\mathsf{Vote}(\mathsf{pk}, id, \mathsf{U}[id], v)$, then $\mathsf{extract}_{\mathsf{id}}(\mathsf{U}, p) = id$ and $\mathsf{extract}_{\mathsf{v}}(\mathsf{sk}, b) = v$ with overwhelming probability. We then write $\mathsf{extract}(\mathsf{sk}, \mathsf{U}, p, b) = (\mathsf{extract}_{\mathsf{id}}(\mathsf{U}, p), \mathsf{extract}_{\mathsf{v}}(\mathsf{sk}, b))$ the extraction function that extracts (id, v) from (p, b). Typically, the extraction of the vote is simply the decryption of the ballot. The extraction of the identity may consist in reading the first component of the ballot (*e.g.* in Helios) and is based on the credential (*e.g.* in Belenios). This extract function will also be used in other parts of the paper.

3.2.1. del + reorder. We start with adversaries who are allowed to arbitrarily change the order of the votes in the ballot box, and remove the votes of the voters who do not run the verification algorithm, but who cannot replace these votes with other votes of their own choosing. In other words, if an adversary succeeds in replacing a vote, he will win the game. Conversely, if she simply blocks a ballot, this will not form an attack. Below, we informally describe the transformation the recovery algorithm recovers (as it is applied to the board in the left world).

Given BB_1 and BB, when applied to BB_0, $\textsc{Recover}^{\mathsf{del},\mathsf{reorder}}$ will construct a board BB' where

- Each ballot in BB that comes from BB_1 is replaced with the ballot at the same position in BB_0.
- The other ballots in BB are considered to be cast, provided they do not belong to a honest voter, *i.e.* if they do not extract to a honest identity. They are added to BB' as is.
- In addition, all ballots in BB_0 created by honest voters who check their votes are added to BB', regardless of whether these voters' ballots actually occur in BB.

The details of the $\textsc{Recover}^{\mathsf{del},\mathsf{reorder}}$ algorithm are in Figure 2.

3.2.2. del + change + reorder. Here, we assume that the adversary is allowed to change the order of the votes, and remove or change the votes of voters who do not verify.

Intuitively, given BB_1 and BB, when applied to BB_0, recovery algorithm $\textsc{Recover}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}$ will construct a board BB' where

- Each ballot in BB that comes from BB_1 is replaced with the corresponding ballot from BB_0.
- The other ballots in BB are considered to be cast, even if they belong to an honest voter. They are added to BB' as is.
- All the ballots registered for voters who check in BB_0 are added to BB', regardless of whether these voters' ballots actually occur in BB.

Formally, $\textsc{Recover}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}$ is defined in Figure 2.

3.2.3. Ideal. Assume now the adversary is not allowed to remove, change, or reorder any honest vote, even for voters who do not verify. Any adversary that can produce a valid board with such an alteration of the ballots will win the game.

Intuitively, given BB_1 and BB, when applied to BB_0, $\textsc{Recover}^{\emptyset}$ will construct a board BB' where

- Each ballot in BB_0 is added to BB', in the same order, regardless of whether the corresponding ballot from BB_1 actually occurs in BB.
- The other ballots in BB that belong to a dishonest voter are considered to be cast, and are added to BB' as is.

Formally, the $\textsc{Recover}^{\emptyset}$ algorithm is displayed in Figure 2.

4. Simulation-based security

In this section we introduce simulation based definitions for the security of voting systems. As usual, we describe a real and an ideal execution scenario for the protocol. The definitions are fairly standard in terms of the underlying communication models, and to a large extent in terms of the ideal functionalities we consider. A major departure from functionalities used in the literature, *e.g.* [18], [10], is that our ideal functionalities explicitly allow the adversary to influence the list of votes to be tallied.

4.1. Real execution

We describe the real execution of the protocol in a hybrid model where the protocol is implemented using ideal functionalities for registration and for tallying. As for our game based approach, some parameters are fixed. These include the sets H and D of honest and corrupt voters and the set $\mathsf{H}_{\mathsf{check}}$ of voters who check. All these parameters are assumed hardwired in the algorithms defining the execution. We illustrate in Figure 3 the execution setting. It comprises:

- The environment $\mathcal{E}$ is in charge of deciding on the execution phases of the protocol (setup, vote, tally); the environment also decides on the votes of the honest users.
- The adversary $\mathcal{A}$: it receives the ballots of the honest users, controls the corrupt users and produces the bulletin board to be tallied. The adversary is controlled by the environment via a direct communication channel.
- An entity $\mathcal{H}$ models the honest parties in the system. In particular it models the honest voters (for simplicity we do not consider separate entities for each individual voter in the system) and the generation of keys for the election.
- The functionality for registration $\mathcal{R}$, in charge of generating and distributing credentials to voters.
- The functionality for tallying $\mathcal{T}$.

The execution consists of three phases (a setup phase, a voting phase, and a tallying phase). The environment $\mathcal{E}$ sends commands to $\mathcal{H}$ to trigger these phases. $\mathcal{H}$ then informs the other entities of the phase change.

Setup phase. During the setup phase, $\mathcal{H}$ runs the Setup algorithm to generate the election keys $(\mathsf{pk}, \mathsf{sk})$, sends pk to the other entities, and sk to $\mathcal{T}$. $\mathcal{H}$ also asks $\mathcal{R}$ to generate credentials. $\mathcal{R}$ runs the $\mathsf{Register}$ algorithm to generate credentials for each voter. It returns the secret credentials of voters in H to $\mathcal{H}$ and the secret credentials of voters in D to $\mathcal{A}$. It also sends the list of public credentials (computed with Pub) to all other entities. Finally, $\mathcal{H}$ returns the control to $\mathcal{E}$.

Voting and checking phase. During the voting phase, $\mathcal{E}$ may send any number of $\mathtt{vote}(id, v)$ to $\mathcal{H}$, for honest voters $id \in \mathsf{H}$. When receiving such a command, $\mathcal{H}$ runs the Vote algorithm to obtain a ballot for id containing v, it records the state returned by the voting algorithm and sends id and the ballot to $\mathcal{A}$. At some point, $\mathcal{E}$ notifies $\mathcal{H}$ that the voting phase is done. At that point, $\mathcal{H}$ asks $\mathcal{A}$ to provide a board BB. $\mathcal{H}$ checks that $\mathsf{Valid}(\mathsf{BB}, \mathsf{pk}) = \top$, and continues the execution. If the check fails, it informs $\mathcal{E}$ that no result will be published. $\mathcal{H}$ then performs the verifications of honest voters. It asks $\mathcal{A}$ in which order the voters should verify. $\mathcal{H}$ then runs the Verify algorithm on BB for each voter in $\mathsf{H}_{\mathsf{check}}$, in the order specified by $\mathcal{A}$. If all of these checks succeed, it continues the execution. Otherwise, it informs $\mathcal{E}$ that no result will be published.

Tallying phase. During the tallying phase, $\mathcal{H}$ sends BB to $\mathcal{T}$ and asks for the result. $\mathcal{T}$ runs the Tally algorithm, to compute a result (r, Π), and sends it back to $\mathcal{H}$. $\mathcal{H}$ forwards this result to $\mathcal{A}$, asking if the result should be published. Depending on $\mathcal{A}$'s decision, $\mathcal{H}$ sends $\mathcal{E}$ either r or a message informing $\mathcal{E}$ that no result is published. Finally, the environment $\mathcal{E}$ outputs a bit β which serves as the output of $\mathsf{realexec}(\mathcal{E}||\mathcal{A}||\mathcal{V})$.

4.2. Ideal voting functionality and ideal execution

The ideal execution replaces the honest participants and the functionalities for registration and tallying with a single idealised functionality $\mathcal{F}_{\mathsf{v}}$. The resulting structure of the system is illustrated in Figure 4. It comprises

- The environment $\mathcal{E}$: as in the real execution the environment decides changes between the different phases of the execution, decides on the votes of the honest parties, and communicates with the ideal world adversary. As in the real case, we will only consider environments that choose to make each voter in $\mathsf{H}_{\mathsf{check}}$ vote at least once.
- The ideal world adversary $\mathcal{S}$, a.k.a. the simulator;

$\text{RECOVER}_{\mathsf{U}}^{\mathsf{del,reorder}}(\mathsf{BB}_1, \mathsf{BB})$

$L \leftarrow []$;
for $(p, b) \in \mathsf{BB}$ **do**
  if $\exists j, id.\ \mathsf{BB}_1[j] = (id, (p, b))$ **then**
    $L \leftarrow L \parallel j$
    (if several such j exist,
     pick the first one)
  else if $\mathsf{extract}_{\mathsf{id}}(\mathsf{U}, p) \notin \mathsf{H}$ **then**
    $L \leftarrow L \parallel (p, b)$
$L' \leftarrow [i | \mathsf{BB}_1[i] = (id, (p, b)) \wedge$
      $id \in \mathsf{H}_{\mathsf{check}} \ \wedge \ (p, b) \notin \mathsf{BB}]$
$L'' \leftarrow L \parallel L'$
return $(\lambda i.\ L''[i])$

$\text{RECOVER}_{\mathsf{U}}^{\emptyset}(\mathsf{BB}_1, \mathsf{BB})$

$L \leftarrow [1, \ldots, |\mathsf{BB}_1|]$;
for $(p, b) \in \mathsf{BB}$ **do**
  if $\mathsf{extract}_{\mathsf{id}}(\mathsf{U}, p) \notin \mathsf{H}$ **then**
    $L \leftarrow L \parallel (p, b)$
return $(\lambda i.\ L[i])$

$\text{RECOVER}_{\mathsf{U}}^{\mathsf{del,reorder,change}}(\mathsf{BB}_1, \mathsf{BB})$

$L \leftarrow []$;
for $(p, b) \in \mathsf{BB}$ **do**
  if $\exists j, id.\ \mathsf{BB}_1[j] = (id, (p, b))$ **then**
    $L \leftarrow L \parallel j$
    (if several such j exist,
     pick the first one)
  else if $\mathsf{extract}_{\mathsf{id}}(\mathsf{U}, p) \notin \mathsf{H}_{\mathsf{check}}$ **then**
  $L \leftarrow L \parallel (p, b)$
$L' \leftarrow [i | \mathsf{BB}_1[i] = (id, (p, b)) \wedge$
      $id \in \mathsf{H}_{\mathsf{check}} \ \wedge \ (p, b) \notin \mathsf{BB}]$
$L'' \leftarrow L \parallel L'$
return $(\lambda i.\ L''[i])$

Figure 2. The $\text{RECOVER}^{\mathsf{del,reorder}}$, $\text{RECOVER}^{\mathsf{del,reorder,change}}$, and $\text{RECOVER}^{\emptyset}$ algorithms.

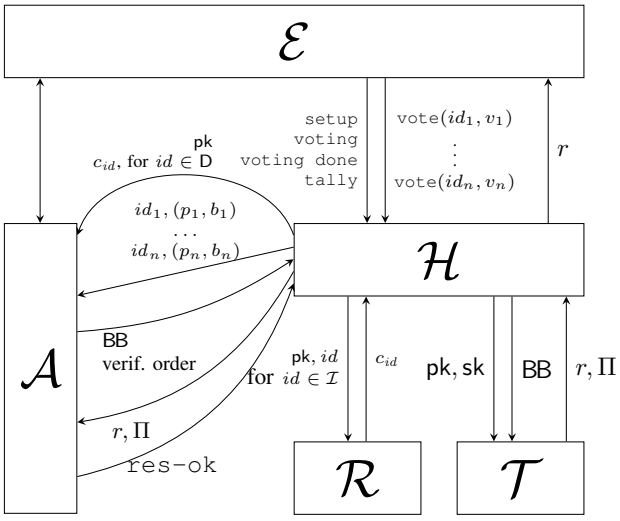

Figure 3. The real execution, with $id_1, \ldots, id_n \in \mathsf{H}$

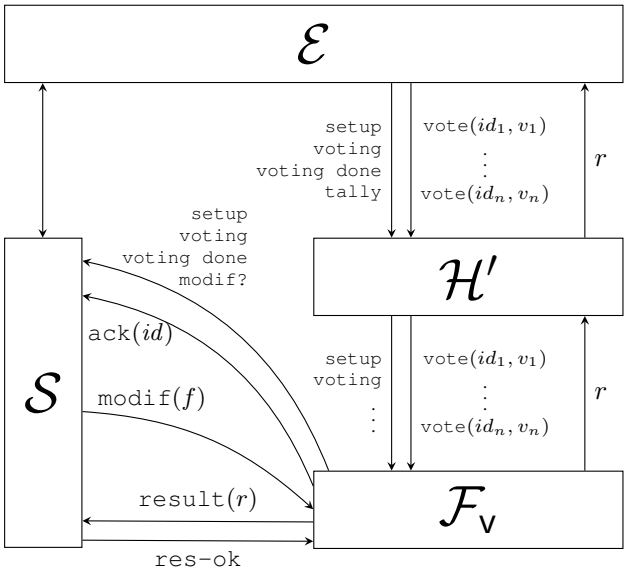

Figure 4. The ideal executionwith $id_1, \ldots, id_n \in \mathsf{H}$

- The ideal voting functionality $\mathcal{F}_{\mathsf{V}}$: this component captures the idealised voting scheme. Very roughly, it receives the votes from the honest parties and, when queried, it returns the result of the election. We give a precise description in the next section.
- The entity $\mathcal{H}'$ is a dummy interface between the environment and the voting functionality (*i.e.* it only forwards the messages between $\mathcal{E}$ and $\mathcal{F}_{\mathsf{V}}$).

As in the real execution the environment decides when to switch between the three phases of the execution (setup, vote and check, tally) and decides on the votes of the honest parties via messages it sends to $\mathcal{H}'$. In this world $\mathcal{H}'$ is simply a forwarding channel between the environment and the ideal functionality (we explain below how the functionality operates). At some point the environment outputs a bit β which is also the output of $\mathsf{idealexec}(\mathcal{E}||\mathcal{S}||\mathcal{F}_{\mathsf{V}})$.

Next we describe the ideal functionality which is the key component of the execution, and which encapsulates the level of security guaranteed.

Ideal voting functionality. We consider several ideal functionalities which share the same basic design idea: they collect the votes of the honest parties (in a way which hides them from the adversary). Nonetheless, since we treat the setting where the adversary controls the bulletin board, and can therefore influence what is being tallied, our functionalities reflect this ability. The difference between the functionalities we consider is reflected in how permissive they are with respect to this step.

The functionality $\mathcal{F}_{\mathsf{V}}^{\mathsf{del,reorder}}(\rho)$ is in Figure 5. In brief, i) it ensures that an adversary only learns who voted, and learns the result of the election, computed using ρ, but not what the votes were; ii) it ensures that the votes of honest voters who verify are not removed (though they may be reordered); iii) it allows an adversary to delete the votes of

$\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}}(\rho)$ accepts the following commands:

- on `setup` from $\mathcal{H}'$: send `setup` to $\mathcal{S}$.
- on `voting` from $\mathcal{H}'$: send `voting` to $\mathcal{S}$.
- on `voting done` from $\mathcal{H}'$: send `voting done` to $\mathcal{S}$.
- on `vote`(id, v) (for $id \in \mathsf{H}$) from $\mathcal{H}'$: $L \leftarrow L \parallel (id, v)$; send `ack`$(id)$ to $\mathcal{S}$.
- on `tally` from $\mathcal{H}'$: send `modif?` to $\mathcal{S}$.
- on `modif`(f) from $\mathcal{S}$: (only once, after `tally`)
  - if f keeps the votes of all voters who check:
$$\forall i.\ \forall id \in \mathsf{H}_{\mathsf{check}}.\ \forall v.\ L[i] = (id, v) \implies \exists j.\ f(j) = i.$$
  - and if no honest votes are modified by f:
$$\forall i, id, v.\ f(i) = (id, v) \implies id \in D.$$
  Then let $r = \rho(\overline{f}(L))$ else let $r =$ `no tally`. Send `result`(r) to $\mathcal{S}$.
- on `res-ok` from $\mathcal{S}$: (only once, after `modif`) send r to $\mathcal{H}'$.
- on `res-block` from $\mathcal{S}$: (only once, after `modif`) send `no tally` to $\mathcal{H}'$.

Figure 5. The ideal functionality $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}}(\rho)$.

voters who do not verify, but not to change them.

Technically, the functionality maintains a list L of votes submitted by honest voters. Once the voting phase is over, it allows the ideal world adversary $\mathcal{S}$ to submit a *vote modification function*, *i.e.* a function f with domain $[\![1, n]\!]$ for some n, that describes how $\mathcal{S}$ wishes to manipulate the votes in L. The function provided by $\mathcal{S}$ needs to satisfy a couple of restrictions. Specifically, for any i, $f(i)$ can either be some index j or some pair (id, v). Applying f to the list L of honest votes results in list $\overline{f}(L)$ of length n defined by

$$\forall i \in [\![1, n]\!].\ \overline{f}(L)[i] = \begin{cases} L[j] & \text{if } f(i) = j \text{ is an index} \\ (id, v) & \text{if } \exists id, v.\ f(i) = (id, v) \end{cases}$$

NB: the function f is applied only if it satisfies the requirements outlined above on how it affects the votes corresponding to the honest voters who check.

Next, we define $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}(\rho)$, a more permissive functionality for voting schemes. This functionality is similar to the previous $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}}(\rho)$ but it allows an adversary to change (and not only delete) the votes of voters who do not verify. Technically, $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}(\rho)$ accepts the same commands as $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}}(\rho)$, and answers them identically, except for the `modif`(f) command. In that case, in $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}(\rho)$, the checks performed before computing the result are:

- f keeps the votes of all voters who check:
$$\forall i.\ \forall id \in \mathsf{H}_{\mathsf{check}}.\ \forall v.\ L[i] = (id, v) \implies \exists j.\ f(j) = i.$$
- no votes from voters who verify are modified by f:
$$\forall i, id, v.\ f(i) = (id, v) \implies id \in D \cup \mathsf{H}_{\overline{\mathsf{check}}}.$$

Finally we define a functionality $\mathcal{F}_{\mathsf{v}}^{\emptyset}(\rho)$, that gives the strongest security guarantees, as it does not allow the adversary to delete, change, nor reorder the votes of honest voters, even if they do not verify. The adversary may only cast votes in the name of dishonest voters. This functionality is similar to the previous two, except it checks a stronger condition before computing the result. More precisely, $\mathcal{F}_{\mathsf{v}}^{\emptyset}(\rho)$ is identical to $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}}(\rho)$, except that the test performed before computing $\rho(\overline{f}(L))$ on command `modif`(f) is that:

- f keeps the votes of all honest voters in the same order:
$$[f(j), j = 1 \ldots |\mathsf{dom}(f)| \mid f(j) \in \mathbb{N}] = [1, \ldots, |L|]$$
- and no honest votes are modified by f:
$$\forall i, id, v.\ f(i) = (id, v) \implies id \in \mathsf{D}.$$

This functionality enforces that no honest votes can be deleted or even reordered. Looking ahead, in the presence of a malicious ballot box, this level of security can be guaranteed only if all honest voters verify their votes.

4.3. Security

As usual, we define simulation-based security by demanding that environments cannot distinguish between the interaction with the real protocol, or with the ideal functionality (together with some simulator). However, as the motivating examples from the introduction show, the level of security guaranteed depends on the fact that (some) voters (*i.e.* those in $\mathsf{H}_{\mathsf{check}}$) check that their vote had been cast. Our security definition captures this by considering certain restrictions. Specifically, we will only consider environments who direct all voters in $\mathsf{H}_{\mathsf{check}}$ to cast at least one vote, so that it makes sense for this vote to be verified. We call such an environment *well-behaved*.

Definition 8. *We say that a voting scheme $\mathcal{V}$ securely implements an ideal functionality $\mathcal{F}_{\mathsf{v}}$ if for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any well-behaved environment $\mathcal{E}$ the outputs of* $\mathsf{realexec}(\mathcal{E}||\mathcal{A}||\mathcal{V})$ *and* $\mathsf{idealexec}(\mathcal{E}||\mathcal{S}||\mathcal{F}_{\mathsf{v}})$ *are indistinguishable.*

5. mb-BPRIV entails simulation-based security

Our main technical result, detailed in this section, is that game-based ballot privacy implies simulation security with respect to a suitable ideal functionality. This holds for *strongly consistent* voting schemes, Strong consistency demands that the tallying process behaves as expected, *i.e.*

it returns the result of tallying the votes which underly the ballots on any given board, even a *dishonestly produced* one. In particular, strong consistency excludes insecure tally functions that would *e.g.* remove the first ballot if it corresponds to a vote for candidate A, hence breaking privacy of the first voter. The notion is a direct extension of the analogous notion introduced by Bernhard et al [10]. It considers an adversary who is given the public key for some election as well as public information for a set $\mathcal{I}$ of registered users. The adversary returns an arbitrary bulletin board. Strong consistency requires that tallying the board returns the same result as running the desired counting function on the votes underlying the ballots on the board. A formal definition can be found in Appendix A.

Both our game-based security notion and the ideal functionalities we consider are parametrised. The former is parametrised by a recovery algorithm which aims to "detect" how the adversary has tampered with the bulletin board. The latter allows the adversary to submit a tampering function, but only allows certain tampering functions. We show that the two parameters are closely related: mb-BPRIV with respect to some specific recovery algorithm implies simulation-based security, if the tampering function recovered by the recovery algorithm is permitted by the ideal functionality.

To make this relatively complex statement more palatable, we start with a warm-up theorem which establishes this type of relation between the three instantiations of mb-BPRIV from Section 3.2 and simulation based security which uses the three ideal functionalities from Section 4.2, respectively. Then, we provide a powerful generalisation which links mb-BPRIV with simulation based security under an abstract assumption on their parameters.

Before we provide our warm-up theorem, we motivate and introduce a mild assumption required by the scheme. All of the recover algorithms considered earlier in this paper identify the ballots on the board by matching them with the specific calls to the voting oracle which produced them. For this reason, a precondition for the recovery algorithms to work as intended is that distinct calls to the Vote algorithm produce two different ballots (except with negligible probability). We say that a scheme with this property does not produce duplicate ballots (see formal definition in Appendix C).

Theorem 1. *Consider a strongly consistent voting scheme $\mathcal{V}$ for counting function ρ which does not produce duplicate ballots. Let* $\mathsf{power} \in \{\emptyset, (\mathsf{del}, \mathsf{reorder}), (\mathsf{del}, \mathsf{reorder}, \mathsf{change})\}$. *If $\mathcal{V}$ satisfies mb-BPRIV with* $\textsc{Recover}^{\mathsf{power}}$*, then $\mathcal{V}$ securely implements* $\mathcal{F}_{\mathsf{v}}^{\mathsf{power}}(\rho)$.

This theorem is proved in a companion technical report [**?**] as a particular case of our general theorem, that we explain next.

Parametric ideal functionality. The three functionalities from Theorem 1, $\mathcal{F}_{\mathsf{v}}^{\emptyset}(\rho)$, $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}}(\rho)$ and $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}(\rho)$ differ only by the check they perform on the modification function provided by the simulator. They can be seen as instances of a more general functionality, that is parametrised by a predicate P that expresses this check. In other words, P characterises the ability of the simulator to manipulate the votes. The predicate P takes as inputs a list L of pairs (id, v), where id is a voter identity and v is a vote, and a modification function f. It returns $\top$ or $\bot$, indicating whether the modifications specified by f are allowed on L or not.

The generalisation is then straightforward: we consider the functionality $\mathcal{F}_{\mathsf{v}}^{P}(\rho)$ with the same interface and (mostly the same) internal behaviour as $\mathcal{F}_{\mathsf{v}}^{\mathsf{power}}(\rho)$. The only distinction is that the checks performed on f before applying to L are replaced with a single check that $P(L, f) = \top$.

Our generic theorem links mb-BPRIV *w.r.t.* some recovery algorithm RECOVER with an ideal functionality which allows tampering satisfying some predicate P whenever the RECOVER algorithm returns (with overwhelming probability) only tampering functions which satisfy P. This condition, which we call *compatibility*, is formally defined in Appendix B.

Our main technical result establishes a relation between game-based and simulation-based security for voting.

Theorem 2 (mb-BPRIV implies simulation)**.** *Let P be a predicate, and* RECOVER *a recovery algorithm compatible with P. Let $\mathcal{V}$ be a strongly consistent voting scheme for counting function ρ.*

If $\mathcal{V}$ satisfies mb-BPRIV w.r.t. RECOVER*, then $\mathcal{V}$ securely implements $\mathcal{F}_{\mathsf{v}}^{P}(\rho)$.*

We give the full proof of this theorem in the technical report [**?**]. We also provide a variant which establishes a similar link for schemes where revote is not allowed.

Proof sketch. Basically, we construct a simulator $\mathcal{S}$ that, given black-box access to a real adversary $\mathcal{A}$, ensures that for any well-behaved environment $\mathcal{E}$, the outputs of $\mathsf{realexec}(\mathcal{E}||\mathcal{A}||\mathcal{V})$ and $\mathsf{idealexec}(\mathcal{E}||\mathcal{S}||\mathcal{F}_{\mathsf{v}}^{P}(\rho))$ are indistinguishable.

$\mathcal{S}$ runs $\mathcal{A}$ internally, letting $\mathcal{A}$ communicate with $\mathcal{E}$, and simulates a real execution of the voting scheme to $\mathcal{A}$. To do this, $\mathcal{S}$ generates "fake" election material to show $\mathcal{A}$. $\mathcal{S}$ must then provide him with honest ballots, to obtain a board BB from $\mathcal{A}$. However $\mathcal{S}$ cannot know to the actual votes: $\mathcal{F}_{\mathsf{v}}^{P}$ only shows who voted, but not the votes. Hence $\mathcal{S}$ cannot construct the real ballots. Instead, $\mathcal{S}$ will construct fake ballots, containing some arbitrary fixed fake vote v^*. Finally, $\mathcal{S}$ must choose which modifications it should submit to $\mathcal{F}_{\mathsf{v}}^{P}$ to get the tally of BB, that $\mathcal{A}$ expects. To do this, $\mathcal{S}$ applies RECOVER to BB to determine how $\mathcal{A}$ manipulated the ballots, and submits the corresponding modifications to $\mathcal{F}_{\mathsf{v}}^{P}$.

Intuitively, the view of $\mathcal{A}$ when placed in the real execution, is as in $\mathsf{Exp}_{\mathcal{V},\mathsf{SimProof}}^{\mathsf{mb\text{-}BPRIV},0}$: the ballots are real, and BB is tallied. Conversely, its view when run by $\mathcal{S}$ in the ideal execution is as in $\mathsf{Exp}_{\mathcal{V},\mathsf{SimProof}}^{\mathsf{mb\text{-}BPRIV},1}$: the ballots contain "fake" votes, and the tally is computed using RECOVER. mb-BPRIV guarantees that $\mathcal{A}$ cannot tell the two situations apart, hence $\mathcal{E}$, communicating with $\mathcal{A}$ only, cannot distinguish between the real and ideal executions.

A bit more formally, we proceed to construct $\mathcal{S}$ by a succession of game hops, progressively going from $\mathsf{realexec}(\mathcal{E}||\mathcal{A}||\mathcal{V})$ to the ideal execution against $\mathcal{S}$.

Game 0 is just the real execution $\mathsf{realexec}(\mathcal{E}||\mathcal{A}||\mathcal{V})$.

Game 1 is a variant of $\mathsf{realexec}(\mathcal{E}||\mathcal{A}||\mathcal{V})$, where $\mathcal{A}$ does not see ballots for the actual votes chosen by $\mathcal{E}$, but rather fake ballots containing an arbitrary fixed vote v^*. The execution of Game 1 follows that of Game 0, except that on $\texttt{vote}(id, v)$ from $\mathcal{E}$, $\mathcal{H}$ generates two ballots: a real one b_0 for v, and a fake one b_1 for v^*. $\mathcal{H}$ stores these respectively in two boards BB_0, BB_1, and only sends b_1 to $\mathcal{A}$ (instead of b_0 in Game 0). Then, on $\texttt{tally}$ from $\mathcal{E}$, $\mathcal{H}$ does not send BB to $\mathcal{T}$ for tallying, but rather $\mathsf{BB}' = \overline{\pi}(\mathsf{BB}_0)$, where $\pi = \textsc{Recover}_{\mathsf{U}}(\mathsf{BB}_1, \mathsf{BB})$. When $\mathcal{H}$ gets a result $(r, \Pi) = \mathsf{Tally}(\mathsf{BB}', \mathsf{sk})$ from $\mathcal{T}$, it simulates a proof $\Pi' = \mathsf{SimProof}(\mathsf{BB}, r)$, and sends (r, Π') to $\mathcal{A}$.

As explained above, Game 0 corresponds to the execution of $\mathsf{Exp}^{\mathsf{mb\text{-}BPRIV},\textsc{Recover},\beta}_{\mathcal{A},\mathcal{V}}$ when $\beta = 0$, and Game 1 to the execution when $\beta = 1$. Since $\mathcal{V}$ is $\mathsf{mb\text{-}BPRIV}$, we are able to show that the outputs of $\mathcal{E}$ in Games 0 and 1 are indistinguishable.

Game 2 establishes an invariant on BB'. It is as Game 1, but has $\mathcal{H}$ store, in addition to BB_0 and BB_1, the list of votes submitted in clear in BB_2. Once $\mathcal{H}$ has determined π, it additionally computes the associated modification function $f = \mathsf{mod}_{\mathsf{sk},\mathsf{U}}(\pi)$, and applies it to BB_2: $\mathsf{BB}'_2 = \overline{f}(\mathsf{BB}_2)$. It then checks that BB'_2 correctly contains the votes in BB' in clear, *i.e.* $\mathsf{BB}'_2 = \overline{\mathsf{extract}}(\mathsf{sk}, \mathsf{U}, \mathsf{BB}')$. Only in that case $\mathcal{H}$ lets the execution go through.

Game 1 and 2 can only be distinguished when this check fails: we can show using strong consistency that this has only negligible probability.

Game 3 then makes use of this invariant: it is as Game 2 except $\mathcal{H}$ directly computes the result as $\rho(\mathsf{BB}'_2)$, instead of calling $\mathcal{T}$. By the previous invariant, and the strong consistency property, this yields the correct result except with negligible probability, and the two games are indistinguishable.

Game 4 removes the invariant introduced in Game 2; it is indistinguishable from Game 3, by the same reasoning as for the hop between Games 1 and 2.

Game 5 removes entirely the ballots containing the real votes, as these are now no longer used to compute the result. $\mathcal{H}$ now only keeps track of BB_1 and BB_2, but not BB_0. As these ballots were unused, this game is indistinguishable from Game 4. To sum up, in Game 5, $\mathcal{H}$ performs the setup, receives and stores honest votes from $\mathcal{E}$ in BB_2, and computes fake ballots for v^* (stored in BB_1) which are sent to $\mathcal{A}$. When $\mathcal{A}$ submits BB, $\mathcal{H}$ lets $\pi = \textsc{Recover}_{\mathsf{U}}(\mathsf{BB}_1, \mathsf{BB})$, $f = \mathsf{mod}_{\mathsf{sk},\mathsf{U}}(\pi)$, $\mathsf{BB}'_2 = \overline{f}(\mathsf{BB}_2)$, and sends $r = \rho(\mathsf{BB}'_2)$ and a simulated proof to $\mathcal{A}$.

Ideal adversary. We can now define the simulator $\mathcal{S}$, that runs $\mathcal{A}$ internally, taking care by itself of most of the actions that were performed by $\mathcal{H}$ and $\mathcal{R}$ in Game 5, *i.e.* the cryptographic setup of keys and credentials, generation of fake ballots, computing of π and f. The only thing $\mathcal{S}$ cannot do on its own is store BB'_2 and apply f to it, as it does not know the real votes. Instead $\mathcal{S}$ sends the modification function f to $\mathcal{F}^P_{\mathsf{v}}(\rho)$, who accepts it as $\textsc{Recover}$ is allowed by P. $\mathcal{F}^P_{\mathsf{v}}(\rho)$ will then apply f to the list of honest votes, and compute the result, that $\mathcal{S}$ will show $\mathcal{A}$ together with a simulated proof.

It is clear that in the ideal execution with this simulator $\mathcal{S}$, both $\mathcal{E}$ and $\mathcal{A}$ have the same view as in Game 5, making them indistinguishable, which concludes the proof. □

Relation with individual verifiability. Interestingly, the simulation-based notion of privacy guarantees more than just privacy: the ideal functionality $\mathcal{F}^P_{\mathsf{v}}(\rho)$ ensures that votes are properly collected and counted, to an extent that depends on the parameter P. As one could expect, the more permissive P (allowing *e.g.* deletions and vote changes), the weaker verifiability guarantees we obtain. Actually, we can formally relate simulation-based security to a rather standard [16] game-based notion of individual verifiability. Basically, individual verifiability requires that some relation holds between the election result and the votes. One typical relation would be that the result must properly account for at least all votes from voters who perform whatever verification checks are prescribed – plus some additional votes, that can come from honest voters who did not check or from corrupted voters. However, the same concerns we exposed earlier for privacy also apply to verifiability: depending on the scheme, threat model and use case considered, different levels of verifiability are achievable and desirable. So we show that we can generically relate our family of privacy notions to various levels of individual verifiability that echo the variants of privacy we defined earlier. For space reasons, the formal definitions and theorems are not presented here but can be found in the technical report [**?**].

6. Application to voting schemes

We use the general framework developed in previous sections to study the resilience of three protocols of the literature, namely Helios [8], Belenios [16], and Civitas [2], in the presence of malicious boards. For each of them, we identify which ideal functionalities they achieve. Interestingly, the guarantees differ depending on the revote policy in place and the counting function ρ. We consider security *w.r.t.* the three functionalities introduced in Section 4.2 plus a new functionality we introduce here. $\mathcal{F}^{\emptyset}_{\mathsf{v}}$ is the very ideal functionality where honest votes are registered and processed exactly as they are received, $\mathcal{F}^{\mathsf{del}}_{\mathsf{v}}$ (defined formally in the technical report [**?**], together with the associated verifiability relation) lets the adversary remove some honest votes (from voters that do not check), but requires that the votes of voters who check are kept and not reordered, while $\mathcal{F}^{\mathsf{del},\mathsf{reorder}}_{\mathsf{v}}$ further lets the adversary reorder the votes. Finally, $\mathcal{F}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}_{\mathsf{v}}$ even lets the adversary modify honest votes, from voters who do not verify.

6.1. Overview of the protocols

Helios. Helios is a simple voting protocol that guarantees privacy and verifiability in a low-coercion environment. It has been used in several elections, *e.g.* to elect the president of the university of Louvain-la-Neuve [19], or for student elections at Princeton [20]. In Helios, voters cast a vote by computing a ballot $\mathsf{Vote}(id, \mathsf{pk}, c, v) = (s, id, (\mathsf{enc}(v, \mathsf{pk}), \pi))$, where the state s records the ciphertext $(\mathsf{enc}(v, \mathsf{pk}), \pi)$ and id is the identity of the voter (or possibly a pseudonym). If the voter id votes again then the state is updated with the new ciphertext. The ciphertext is simply formed of an ElGamal encryption $\mathsf{enc}(v, \mathsf{pk})$ of v under the public key of the election pk, together with π, a zero-knowledge proof that guarantees that v is a valid vote. Helios does not use credentials, hence c is not used. The check $\mathsf{Verify}(id, s, \mathsf{BB})$ done by a voter id consists in verifying that the ciphertext recorded in s appears on BB.

Belenios. Belenios enhances Helios with credentials, so that a compromised voting server cannot add votes. It has been launched in 2016 and used in more than 200 elections [21]. At registration, each voter id receives a signing key k_{id}, with an associated verification key pk_{id}. The voting procedure $\mathsf{Vote}(id, \mathsf{pk}, (k_{id}, pk_{id}), v)$ produces the state and ballot $(s, pk_{id}, (\mathsf{signElGamal}(v, \mathsf{pk}, k_{id}), \pi))$, where we denote $\mathsf{signElGamal}(v, \mathsf{pk}, k_{id})$ the ElGamal encryption of v, signed with k_{id}. The other algorithms are modified as expected. To ease the verification step made by voters, in Helios and Belenios, only the last ballot for each voter is presented in the bulletin board. This can be modelled by a $\mathsf{Verify}(id, s, \mathsf{BB})$ algorithm that checks that the last ballot recorded in s is the last ballot appearing in BB for voter id.

Civitas. Civitas has been designed to protect voters against coercion. Each voter receives a credential but can produce a fake credential when she is under coercion. Ballots cast with invalid credentials are removed thanks to plaintext equality tests (PET), after some mixing phase, to avoid a coercer noticing that his ballots have been excluded. The voting procedure $\mathsf{Vote}(id, \mathsf{pk}, c, v)$ yields $(s, \mathsf{enc}(c, \mathsf{pk}), (\mathsf{enc}(v, \mathsf{pk}), \pi_1, \pi_2))$, where s again records the ballot. π_1 is a zero-knowledge proof that v is a valid vote, and π_2 is a zero-knowledge proof that the agent generating the ballot knows both c and v.

In Civitas, the voting server can no longer select the "last" ballot for each voter since ballots cannot be properly linked to an identity. So when a voter revotes (if this is allowed), she should additionally link her new ballot to the previous one, proving that she knows the credential and the choices used in both ballots. Then the $\mathsf{Valid}(\mathsf{BB}, \mathsf{pk})$ algorithm checks consistency of all the proofs.

6.2. Our findings

The results of our study are gathered in Figure 6. For each protocol, we distinguish the case where revote is allowed from the case where it is not. When revote is not allowed then we assume that (honest) voters do *not* revote. As we will discuss in this section, this is not equivalent, security-wise, to the case where voters may revote but only the first vote is counted.

(in)security of Helios. As mentioned already in introduction, Helios is subject to an attack [12] if the attacker controls the bulletin board (or simply the communication channel between the voter and the server). Indeed, an attacker may block and copy the first ballot b_A sent by Alice, say for candidate 0. The attacker can then pretend that the communication was lost, so that Alice starts over the procedure and sends again a ballot, b'_A, still for candidate 0 (there is no reason that she changes her mind). Then since ballots are not cryptographically authenticated in Helios, the attacker may submit b_A as his own ballot, introducing a bias in the result. This attack cannot be prevented, even if the auditors check for duplicates before the tally. Therefore Helios with revote does not satisfy any of the four functionalities.

Assume now that there is no revote, in the strong sense that voters do not ever construct two ballots for their vote. Since in Helios the identity of a voter is not strongly linked to the ciphertext containing her vote, as soon as two voters A and B do not verify, an adversary is able to swap their ciphertexts, *e.g.* replacing $[(A, b_A), (B, b_B)]$ with $[(A, b_B), (B, b_A)]$. This is fine as long as the counting function processes the votes independently of the actual identity of a voter, so that attributing Alice's vote to Bob and vice versa does not change the result. We call this property *id-blindness*. Most voting functions enjoy this property but not all of them. For example, for elections with weighted votes, each vote is associated to a weight depending on the status of the voter. Then, it could be the case that Alice's vote is counted 10 times while Bob's vote is counted only 3 times. For id-blind counting functions, Helios satisfies $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder},\mathsf{change}}$, the weakest functionality, since an attacker may reorder votes and remove and even modify the ballots of voters that do not check.

No revote vs $\rho = first$. Interestingly, the Helios example illustrates why it is not possible to properly emulate the "no revote" policy by letting voters revote and considering a function ρ where only the first ballot is counted for each voter. In fact, if voters may revote then the adversary has more power. In particular, Helios with revote is still subject to Roenne's attack, even if only the first ballot is counted.

No reordering in Civitas. Our findings highlight that Civitas is the only scheme that prevents an adversary from reordering the votes, thanks to the link made by voters between their ballots. Note that if the attacker controls the board, then he can always permute Alice and Bob's ballots without anyone noticing. However, this does not influence the result for all the result functions ρ we know, namely *stable functions*, that if swapping any two votes of *distinct* voters does not influence the result.

So now the question is: which schemes can prevent an adversary from reordering the votes of a given voter? This would be a priori a real attack since it does change the result.

Voting scheme	$\mathcal{F}_{\mathsf{v}}^{\emptyset}$		$\mathcal{F}_{\mathsf{v}}^{\mathsf{del}}$	$\mathcal{F}_{\mathsf{v}}^{\mathsf{del,reorder}}$	$\mathcal{F}_{\mathsf{v}}^{\mathsf{del,reorder,change}}$
	General case	$\mathsf{H}_{\mathsf{check}} = \mathsf{H}$			
Helios - no revote	✗	✓ if ρ stable $\wedge$ id-blind	✗	✗	✓ if ρ id-blind
Helios - revote	✗	✗	✗	✗	✗
Belenios - no revote	✗	✓ if ρ stable	✓	✓	✓
Belenios - revote, $\rho = last$	✗	✓ if ρ stable	✓	✓	✓
Belenios - revote, arbitrary rev. policy	✗	✗	✗	✗	✗
Civitas - no revote	✗	✓ if ρ stable	✓	✓	✓
Civitas - revote	✗	✓ if ρ stable	✓	✓	✓

Figure 6. Case study.

Our results show that only Civitas protects again such a re-ordering, thanks to the chain between ballots cast by the same voter. Belenios provides weaker security guarantees since the adversary may reorder the votes of Alice. However, this does not affect the result as long as only the last ballot is counted, since Alice checks that her last ballot appears in the final board. To render Belenios suitable for *arbitrary* (stable) counting functions, we would need to require that each voter records her ballots in order, and checks that they appear *in the same order* in the final board. This would of course not be realistic. Alternatively, the most reasonable approach is probably to chain ballots thanks to an additional zero-knowledge proof, like in Civitas. Note however that this chain is only briefly sketched in [2] and no proper definition is provided.

"Perfect" functionality $\mathcal{F}_{\mathsf{v}}^{\emptyset}$. Perhaps surprisingly, none of the three schemes satisfy the strongest ideal functionality, where the attacker cannot tamper at all with honest votes. This is due to the fact that an adversary can always drop the ballots of voters that do not check. This limitation applies to many other schemes as well. If we assume now that *all* honest voters actually vote and conduct all required checks, the three schemes (except Helios with revote) satisfy $\mathcal{F}_{\mathsf{v}}^{\emptyset}$. This requirement is however not realistic in practice.

Proofs. To establish security with respect to ideal functionalities in Figure 6 we leverage the framework developed in this paper in two distinct ways. First, for each scheme in turn we prove game based security with respect to appropriately chosen recovery algorithms and then employ Theorem 2 to conclude simulation-based security. Interestingly, we also employ reasoning about ideal functionalities directly. Specifically, we show that for stable ρ functions, $\mathcal{F}_{\mathsf{v}}^{\mathsf{del}} \wedge \mathsf{H} = \mathsf{H}_{\mathsf{check}} \Rightarrow \mathcal{F}_{\mathsf{v}}^{\emptyset}$ and the desired results in the column $\mathsf{H} = \mathsf{H}_{\mathsf{check}}$ (under the $\mathcal{F}_{\mathsf{v}}^{\emptyset}$ heading) follow from those in column with heading $\mathcal{F}_{\mathsf{v}}^{\mathsf{del}}$.

7. Discussion and conclusion

We have proposed a flexible definition for ballot privacy against a dishonest board, that captures several adversarial capabilities. This is not only important to analyse different schemes with various security levels but also useful when analysing a given scheme under different trust assumptions, as exemplified by our case analysis. We formally relate mb-BPRIV (with strong consistency) with a family of ideal functionalities, for which it is easier to understand the associated security guarantees. As a result, we are able to formally spell out for example that Belenios and Civitas offer more privacy than Helios thanks to the fact that ballots are authenticated. This does not mean that Helios does not offer any privacy at all. A nice feature of our approach is precisely that it allows to compare the security of several voting schemes and even compare the security offered by a single scheme depending on the threat model.

Note that while our definition now reflects that the adversary controls the ballots cast by honest voters (to the extent permitted by the checks in place), we still implicitly assume that the voters and the auditors running the Valid algorithm all have access to the *same* board, right before the tally. In practice, voters check their ballot whenever they wish to, typically right after they voted. So not only voters should always see the same board but it should grow consistently during the election. Most voting schemes assume such a setting while eluding on how to achieve this. Recent work [13], [6], [11], [14] study this issue and propose to distribute the board. We plan to investigate whether a given voting scheme, implemented with some distributed board, does achieve mb-BPRIV.

As future work, we also plan to extend our definition to more complex models of voting protocols. For example, it would be interesting to model possible interactions between a voter and the board or the server while casting a vote. Furthermore, the tally process is currently represented as one single, honest block. A refined model should reflect the fact that the tally is typically distributed among several parties.

Acknowledgements

This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement No 645865-SPOOC).

References

[1] B. Adida, "Helios: Web-based open-audit voting," in *17th USENIX Security Symposium (Usenix'08)*, 2008, pp. 335–348.

[2] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," in *IEEE Symposium on Security and Privacy (S&P'08)*, 2008, pp. 354–368.

[3] P. Y. A. Ryan, P. B. Roenne, and V. Iovino, "Selene: Voting with transparent verifiability and coercion-mitigation," in *1st Workshop on Secure Voting Systems (VOTING'16)*, 2016, pp. 176–192.

[4] R. Küsters, J. Müller, E. Scapin, and T. Truderung, "sElect: A lightweight verifiable remote voting system," in *29th IEEE Computer Security Foundations Symposium (CSF'16)*, 2016, pp. 341–354.

[5] D. A. Basin, S. Radomirovic, and L. Schmid, "Alethea: A provably secure random sample voting protocol," in *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, 2018, pp. 283–297. [Online]. Available: https://doi.org/10.1109/CSF.2018.00028

[6] C. Culnane and S. Schneider, "A peered bulletin board for robust use in verifiable voting systems," in *27th IEEE Computer Security Foundations Symposium (CSF 2014)*, 2014, p. 169–183.

[7] D. Bernhard and B. Smyth, "Ballot secrecy with malicious bulletin boards," Cryptology ePrint Archive, Report 2014/822, 2014.

[8] V. Cortier and J. Lallemand, "Voting: You can't have privacy without individual verifiability," in *25th ACM Conference on Computer and Communications Security (CCS'18)*. ACM, 2018, pp. 53–66.

[9] J. C. Benaloh and M. Yung, "Distributing the power of a government to enhance the privacy of voters," in *5th ACM Symposium on Principles of Distributed Computing, (PODC'86*, 1986, pp. 52—62.

[10] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "A comprehensive analysis of game-based ballot privacy definitions," in *36th IEEE Symposium on Security and Privacy (S&P'15)*, May 2015, pp. 499–516.

[11] N. Chondros, B. Zhang, T. Zacharias, P. Diamantopoulos, S. Maneas, C. Patsonakis, A. Delis, A. Kiayias, and M. Roussopoulos, "D-demos: A distributed, end-to-end verifiable, internet voting system," in *ICDCS 2016*, 2016, pp. 711–720.

[12] P. Roenne, "Private communication."

[13] A. Kiayias, A. Kuldmaa, H. Lipmaa, J. Siim, and T. Zacharias, "On the security properties of e-voting bulletin boards," in *11th International Conference on Security and Cryptography for Networks (SCN 2018)*, 2018, pp. 505–523.

[14] L. Hirschi, L. Schmid, and D. Basin, "Fixing the achilles heel of e-voting: The bulletin board," Cryptology ePrint Archive, Report 2020/109, 2020, https://eprint.iacr.org/2020/109.

[15] D. Bernhard, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi, "Adapting helios for provable ballot secrecy," in *Proceedings of the 16th European Symposium on Research in Computer Security (ESORICS'11)*, ser. Lecture Notes in Computer Science, Springer, Ed., vol. 6879, 2011.

[16] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, "Election verifiability for Helios under weaker trust assumptions," in *19th European Symposium on Research in Computer Security (ESORICS'14)*, ser. LNCS, vol. 8713. Springer, 2014, pp. 327–344.

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

[18] J. Groth, "Evaluating security of oting schemes in the universal composability framework," in *International Conference on Applied Cryptography and Network Securtiy*, 2004, pp. 46–60.

[19] B. Adida, O. D. Marneffe, O. Pereira, and J.-J. Quisquater, "Electing a university president using open-audit voting: Analysis of real-world use of helios," in *International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, 2009.

[20] O. Pereira, *Real-World Electronic Voting: Design, Analysis and Deployment*. CRC Press, 2016, ch. Internet Voting with Helios.

[21] V. Cortier, P. Gaudry, and S. Glondu, *Belenios: A Simple Private and Verifiable Electronic Voting System*. Springer International Publishing, 2019, pp. 214–238.

Appendix A. Strong consistency

In this section we formally define the notion of strong consistency, evoked in Section 5.

Definition 9 (Strong consistency)**.** *A voting scheme $\mathcal{V}$ is strongly consistent if there exist two algorithms* $\mathsf{extract}_{\mathsf{id}}$, $\mathsf{extract}_{\mathsf{v}}$ *such that:*

- *For any $id \in \mathcal{I}$, and any vote v, if* $(\mathsf{pk}, \mathsf{sk})$ *are generated by* Setup, U *by* Register, *and* (p, b) *by* $\mathsf{Vote}(\mathsf{pk}, id, \mathsf{U}[id], v)$, *then* $\mathsf{extract}_{\mathsf{id}}(\mathsf{U}, p) = id$ *and* $\mathsf{extract}_{\mathsf{v}}(\mathsf{sk}, b) = v$ *with overwhelming probability. We then write* $\mathsf{extract}(\mathsf{sk}, \mathsf{U}, p, b) = (\mathsf{extract}_{\mathsf{id}}(\mathsf{U}, p), \mathsf{extract}_{\mathsf{v}}(\mathsf{sk}, b))$ *the extraction function that extracts (id, v) from (p, b).*
- *For any adversary $\mathcal{A}$, the advantage $\mathbb{P}(\mathsf{Exp}^{\mathsf{SC}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1)$ is negligible, where $\mathsf{Exp}^{\mathsf{SC}}_{\mathcal{A},\mathcal{V}}$ is defined as the following game:*

$\mathsf{Exp}^{\mathsf{SC}}_{\mathcal{A},\mathcal{V}}(\lambda)$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
for all $id \in \mathcal{I}$ **do**
  $\mathsf{U}[id] \leftarrow \mathsf{Register}(id)$
$\mathsf{BB} \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{U})$
$(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$
if $r \neq \rho(\mathsf{extract}(\mathsf{sk}, \mathsf{U}, p_1, b_1), \ldots, \mathsf{extract}(\mathsf{sk}, \mathsf{U}, p_n, b_n))$
  where $\mathsf{BB} = [(p_1, b_1), \ldots, (p_n, b_n)]$
then return 1
else return 0

Notation: for any board $\mathsf{BB} = [(p_1, b_1), \ldots, (p_n, b_n)]$, any sk, U and extract, we denote $\mathsf{extract}(\mathsf{sk}, \mathsf{U}, \mathsf{BB})$ the list of extractions of the ballots in BB, *i.e.* $[\mathsf{extract}(\mathsf{sk}, \mathsf{U}, p_1, b_1), \ldots, \mathsf{extract}(\mathsf{sk}, \mathsf{U}, p_n, b_n)]$. We also denote $\overline{\mathsf{extract}}(\mathsf{sk}, \mathsf{U}, \mathsf{BB})$ the list obtained by removing all $\perp$ elements from $\mathsf{extract}(\mathsf{sk}, \mathsf{U}, \mathsf{BB})$. Note that, by definition of a counting function, $\rho(\overline{\mathsf{extract}}(\mathsf{sk}, \mathsf{U}, \mathsf{BB})) = \rho(\mathsf{extract}(\mathsf{sk}, \mathsf{U}, \mathsf{BB}))$.

Appendix B. Compatibility of a predicate and a recovery algorithm

In the informal description from Section 5, we overloaded the semantics of the recovery algorithm. Strictly speaking this algorithm returns a selection function, which in turn defines a tampering function. Below, we develop the technical machinery that captures these ideas.

First we relate selection functions (which are the type of functions returned by recovery algorithms) with tampering functions (the functions the simulators provide to the ideal functionalities). This definition can be seen as the analogous definition of applying a selection function to a bulletin board, except that now we operate at vote level (rather than ballot level). In particular, this requires that we recover the votes underlying ballots in the selection function.

Definition 10 (Vote modification associated to a selection function)**.** *Assume a strongly consistent voting scheme $\mathcal{V}$. Let*

$\mathsf{Exp}^{\mathsf{comp},P,\textsc{Recover}}_{\mathcal{A},\mathcal{V}}(\lambda)$

$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
for all $id \in \mathcal{I}$ **do**
  $c \leftarrow \mathsf{Register}(1^\lambda, id)$
  $\mathsf{U}[id] \leftarrow c$,
  $\mathsf{PU}[id] \leftarrow \mathsf{Pub}(c)$
for all $id \in \mathsf{D}$ **do**
  $\mathsf{CU}[id] \leftarrow \mathsf{U}[id]$
$\mathsf{BB} \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{vote}}(\mathsf{pk},\mathsf{CU},\mathsf{PU})$
if $\mathsf{Valid}(\mathsf{BB},\mathsf{pk}) = \bot \vee \mathsf{H}_{\mathsf{check}} \not\subseteq \mathsf{L}_{id}$ **then return** 0
$\pi \leftarrow \textsc{Recover}_{\mathsf{U}}(\mathsf{BB}_0,\mathsf{BB})$
if π is not compatible with P *w.r.t.* $\mathsf{sk},\mathsf{U},\mathsf{L}_{id}$
then return 1 **else return** 0.

$\mathcal{O}\mathsf{vote}(id,v)$ for $id \in \mathsf{H}$

$(p,b,s) \leftarrow \mathsf{Vote}(\mathsf{pk},id,\mathsf{U}[id],v)$
$\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \parallel (id,(p,b))$;
$\mathsf{L}_{id} \leftarrow \mathsf{L}_{id} \parallel id$;
return (p,b).

Figure 7. The compatibility game.

$(\mathsf{pk},\mathsf{sk})$ *be a pair of keys generated by the* Setup *algorithm, and* U *be a list of credentials issued by* Register. *Let* π *be a selection function for two integers* m, n. *Let then* L *be the list of length* n *defined by*

$$\forall i \in [\![1,n]\!].\ L[i] = \begin{cases} \pi(i) & \text{if } \pi(i) \in [\![1,m]\!] \\ \mathsf{extract}(\mathsf{sk},\mathsf{U},p,b) & \text{if } \exists p,b.\ \pi(i)=(p,b) \end{cases}$$

The vote modification $\mathsf{mod}_{\mathsf{sk},\mathsf{U}}(\pi)$ *associated to* π *for* sk *and* U *is the function* $\lambda i.\ L'[i]$ *(*L' *is* L *where* $\bot$ *elements have been removed).*

As explained informally in Section 5, we want to consider ideal functionalities which put restrictions on how the adversary (the simulator) can tamper with the list of votes collected by the functionality. We capture this intuition by requiring that the selection function is *compatible* with the testing predicate associated to the functionality.

Definition 11 (Selection function compatible with a testing predicate)**.** *Let* $(\mathsf{pk},\mathsf{sk})$ *be a pair of keys generated by the* Setup *algorithm, and* U *be a list of credentials issued by* Register. *Let* P *be a predicate, and* π *be a selection function for* m, n. *Let* L_{id} *be a list of ids of length* m. *We say that* π *is compatible with* P *w.r.t.* sk, U, *and* L_{id} *if for any list* L *of pairs of the form* (id,v) *such that* $[id \mid (id,v) \in L] = L_{id}$, *we have* $P(L, \mathsf{mod}_{\mathsf{sk},\mathsf{U}}(\pi)) = \top$.

The notion of compatibility can then be extended from individual selection functions to recovery algorithms which return selection functions. The general intuition is that $\textsc{Recover}$ is compatible with P if $\textsc{Recover}$ (almost) always returns selection functions compatible with P in normal executions of the scheme (*i.e.* where parameters and ballots are generated honestly).

Definition 12 (Recovery algorithm compatible with a testing predicate)**.** *Let* P *be a predicate, and* $\textsc{Recover}$ *be a recovery algorithm. We say that* $\textsc{Recover}$ *is compatible with* P *if for any adversary* $\mathcal{A}$, *the advantage* $\mathbb{P}(\mathsf{Exp}^{\mathsf{comp},P,\textsc{Recover}}_{\mathcal{A},\mathcal{V}}(\lambda) = 1)$ *is negligible, where* $\mathsf{Exp}^{\mathsf{comp},P,\textsc{Recover}}_{\mathcal{A},\mathcal{V}}$ *is defined in Figure 7.*

For example, $\textsc{Recover}^{\mathsf{del},\mathsf{reorder}}$ is compatible with predicate $P^{\mathsf{del},\mathsf{reorder}}$, where $P^{\mathsf{del},\mathsf{reorder}}(L,f)$ holds if f keeps the votes of all voters who check (that is, $\forall id \in \mathsf{H}_{\mathsf{check}}.\ L[i] = (id,v) \implies \exists j.\ f(j) = i$) and no honest votes are modified by f (*i.e.* $f(i) = (id,v) \implies id \in \mathsf{D}$).

This precisely corresponds to the checks made by $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}}$ or, in other words, $\mathcal{F}_{\mathsf{v}}^{\mathsf{del},\mathsf{reorder}} = \mathcal{F}_{\mathsf{v}}^{P^{\mathsf{del},\mathsf{reorder}}}$.

Appendix C. No duplicate ballots

We give here the formal definition of the "no duplicate ballots" assumption mentioned in Section 5.

Definition 13 (No duplicate ballots)**.** *A voting scheme* $(\mathsf{Setup},\mathsf{Register},\mathsf{Pub},\mathsf{Vote},\mathsf{Valid},\mathsf{Tally},\mathsf{Verify})$ does not produce duplicate ballots *if for all adversary* $\mathcal{A}$, *the following probability is negligible in* λ.

$$\mathbb{P}(\ (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda);\ \mathsf{U} \leftarrow \mathsf{Register}(1^\lambda,\mathcal{I});\ (id,v,id',v') \leftarrow \mathcal{A}(\mathsf{pk},\mathsf{U});\ (p,b,s) \leftarrow \mathsf{Vote}(\mathsf{pk},id,\mathsf{U}[id],v);\ (p',b',s') \leftarrow \mathsf{Vote}(\mathsf{pk},id',\mathsf{U}[id'],v');\ (p,b) = (p',b'))$$