A comprehensive analysis of game-based ballot privacy definitions

David Bernhard[1], Véronique Cortier[2], David Galindo[2,3], Olivier Pereira[4], and Bogdan Warinschi[1]

[1] University of Bristol
Bristol, United Kingdom
[2] CNRS, Loria, UMR 7503
Vandoeuvre-lès-Nancy, France
[3] Scytl Secure Electronic Voting
Barcelona, Spain
[4] Université Catholique de Louvain – ICTEAM
Louvain-la-Neuve, Belgium

Abstract. We critically survey game-based security definitions for the privacy of voting schemes. In addition to known limitations, we unveil several previously unnoticed shortcomings. Surprisingly, the conclusion of our study is that none of the existing definitions is satisfactory: they either provide only weak guarantees, or can be applied only to a limited class of schemes, or both.

Based on our findings, we propose a new game-based definition of privacy which we call BPRIV. We also identify a new property which we call *strong consistency*, needed to express that tallying does not leak sensitive information. We validate our security notions by showing that BPRIV, strong consistency (and an additional simple property called strong correctness) for a voting scheme imply its security in a simulation-based sense. This result also yields a proof technique for proving entropy-based notions of privacy which offer the strongest security guarantees but are hard to prove directly: first prove your scheme BPRIV, strongly consistent (and correct),then study the entropy-based privacy of the result function of the election, which is a much easier task.

Keywords: Voting, Privacy, Ballot Privacy, Cryptography

1 Introduction

Privacy of votes was the subject of major debates during the 19th century, at the time of the progressive introduction of universal suffrage. Since then, it has become a standard in all major democracies.

The introduction of electronic technologies as part of the voting process however raises new challenges and privacy concerns. Cryptographic voting protocols aim to guarantee ballot privacy in e-voting by defining security models and then constructing schemes[1]to meet these models. Generally, cryptographic voting schemes may be categorised into the purely electronic where voters may vote from the privacy of their own computers (e.g., Helios [3, 4] or Civitas [19]) and hybrid systems which use paper ballots and computers to facilitate the tally (e.g., ThreeBallot [40], Prêt-à-Voter [41] and Scantegrity [17]).

Modelling privacy The development of security models for ballot privacy started with the work of Benaloh [21, 6] and has recently started to receive more attention with new models being developed in both symbolic models [32] and computational ones [37, 38, 11]. Unlike related privacy notions such as confidential message transmission, ballot privacy is not absolute but relative to specific election bylaws and voter choices. Consider a voting system that discloses the number of votes received by each candidate: such a system essentially reveals how each voter voted in the case where all voters vote for the same candidate. Classifying such a system as insecure is clearly undesirable; ballot privacy notions require a more nuanced classification.

One generic approach is to define vote privacy through the design of an ideal functionality [35, 30, 39]: a voting scheme is declared to satisfy privacy if it securely realizes (in some formal sense) the ideal functionality. While being very powerful, these definitions are also quite difficult to prove on real cryptographic voting protocols. For

[1] The terms "scheme" and "protocol" can be read interchangeably without much loss of precision. We use the former to refer to a collection of algorithms and the latter to include the specification of who should execute these algorithms and when.

example, Helios is a purely cryptographic voting protocol (that does not rely on paper ballots) which has been used for real elections, and we are not aware of any security proofs for Helios in a simulation-based model[2].

Another, more general, approach to ballot privacy definitions focuses on entropy [22, 12], used as a measure of the amount of information that a voting system leaks about votes. Early works were based on Shannon entropy, but other entropy notions, based on min-entropy and Hartley entropy showed to be particularly informative in natural contexts. The interest of these entropy-based definitions is that they capture many possible sources of privacy leakage: the privacy leakage can be caused by the choice of the cryptographic primitives but it may also be due to the election result itself (which is out of the scope of simulation-based models) or from the distribution of the votes. A voting scheme should be considered as private under such a notion, if its privacy leakage is (computationally) close to the privacy leakage of an associated "ideal protocol" where voters send their vote on a secure channel to a trusted party that simply computes and announces the result. Furthermore, the amount of information that is leaked by this result is precisely measured, making it possible to compare different election tallying rules in various contexts.

A third approach, initiated in the early works of Benaloh [6], considers game-based definitions of vote privacy. This approach has been used in the literature to model e.g. Helios. The literature is actually quite abundant in terms of game-based definitions of vote privacy, the differences between them are poorly understood, and there is little guidance on how to select one to analyze protocols. This work started as an attempt to understand the strength and weaknesses of the many game-based notions (as well as those of several non game-based ones). In these game-based notions, privacy is described as the negligible probability of an adversary to distinguish between two situations, precisely described as games in the definition. By only focusing on the information leakage due to the cryptography, leaving the measurement of the leakages due the tally to be measured with information theoretic techniques, this approach typically leads to simpler, more generic and more modular security proofs.

The boundaries between these three definition approaches can of course sometimes become fuzzy, with some works mixing them into a single definition. For instance, Küsters, Truderung, and Vogt [38] provide a privacy metric that focuses on the probability that an attacker notices when an (honest) voter changes her vote, while all of the other (honest) votes follow a given distribution. These authors show how to analyze the privacy offered by several paper-based voting protocols such as ThreeBallot and VAV.

Comparing existing notions of privacy Our first contribution is to systematically review, compare, and discuss existing game-based computational notions for vote privacy. In particular, we present them in a unified framework which facilitates their comparison. Our review of the literature shows that none of the existing definitions is satisfactory. Some limitations were already known but we discovered further unnoticed shortcomings in several of them. In short, based on our findings, we classify existing definitions in three categories:

- too weak [14, 13, 23, 16]: these declare protocols to be secure which intuitively do not preserve vote privacy. We give examples which we think should be considered privacy breaches, despite the examples meeting the given privacy definitions.
- too strong [15]: by contrast, this definition is so strong that no verifiable protocol can meet the privacy constraint. More precisely, we show that any protocol meeting this definition must allow the authorities to announce any result that is consistent with the number of votes on the board. Therefore this definition cannot be used for any protocol that aims at some verifiability, which is the case of most protocols of the literature.
- too limited [10, 7, 9, 11, 38, 12, 29]: while we did not identify any flaw in these definitions, they restrict the class of protocols or privacy breaches that can be considered.

  For example, they may lead to inconsistent results when applied to protocols that use some natural result functions (e.g., the majority function) or may not be applicable to protocols that output not only a result but also a proof of correct tallying (such as most cryptographic protocols do), or they may only detect specific privacy breaches (e.g. the case whether two voters vote the same is not covered).

[2] Groth [35] proved a class of protocols to be UC-secure that at a first glance might seem to include Helios. However, Groth requires that the protocols use voter's identifiers as part of the correctness proofs in ballots — Helios does not do this.

We summarise the limitations of each definition in Table 1 later in the paper. For our classification, we designed several test-case (dummy) protocols that may be used to evaluate a privacy definition; these are available in Appendix A.

A new notion for privacy Our second contribution is to propose a new game-based definition of privacy, called BPRIV, that incorporates the lessons learned from our study. Our new definition accounts for auxiliary data in the tally (such as proofs of correct decryption), is compatible with verifiability, and does not suffer from any of the flaws we uncovered. As a test of our new definition, we prove the Helios voting protocol to be BPRIV secure. Specifically, we analyze what could be considered the standard version of Helios nowadays (at least, academically speaking), that uses strong Fiat-Shamir proofs [13], implements duplicate weeding [25] and homomorphic tallying. With respect to the threat model, we consider an honest single trustee, an honest ballot box, and an adversary that can adaptively corrupt a subset of voters. Apart from the single trustee (that can be dealt of by adapting BPRIV to a multi-authority setting), the other adversarial assumptions are similar to those used in previous ballot privacy analyses of Helios.

The first of two novelties of our definition is that it accounts for tallying operations that possibly include revote policies (for example, only your last vote counts). Understanding tallying is crucial for the privacy of ballots: not only does the tallying operation usually disclose auxiliary data in addition to the result (such as proofs of correct tallying) but it also performs some cleaning operations such as removing invalid or duplicate ballots or ballots that should be erased due to re-voting, etc. For example, in Civitas [19], coercion-resistance crucially relies on the fact that ballots submitted under coercion can be (anonymously) identified and removed, without endangering the voter's identity. In Helios 2.0, removing duplicates is necessary for privacy, otherwise the protocol is subject to replay attack [26]. These operations are often considered harmless but, perhaps surprisingly, they may be crucial for the security of a voting system. Since these cleaning operations may be used to guarantee privacy, they may also damage it if performed incorrectly. Therefore, a good privacy definition should account for these operations too. In this direction, we identify a second security property, which we call *strong consistency*, that ensures that the tally phase counts the votes properly, even in the presence of an adversary. While this property is clearly desirable for verifiability, it is also crucial for privacy: otherwise the tally phase could leak information on honest votes in the result itself or, more subtly, in the choice of ballots that are removed during the cleaning operations.

A simulation-based notion of security A natural question is how to convince ourselves that the new definition BPRIV is not flawed as well. The second novelty of our definition is that we establish a tight relation between BPRIV and security in a simulation-based[3] model. Specifically, any voting scheme that is BPRIV, strongly consistent (and strongly correct) securely implements an ideal functionality. The privacy guarantees of this ideal functionality are simpler to understand compared to game-based definitions, but simulation-based security proofs are harder to carry out. We view the relation with the simulation-based notion as a validation of our new BPRIV notion. In addition, we obtain the first security proof for Helios in a simulation-based model.

A potential advantage of simulation-based models is that their security guarantees are easier to understand intuitively than game-based ones. Our ideal functionality can be described in one sentence: it simply collects all votes from the voters, then computes and announces the result. Any scheme that securely implements this functionality cannot leak any more information (in particular about people's votes) than this functionality. This result gives us a way to explain the privacy guarantees of Helios even to non-cryptographers. In addition, the result provides a bridge towards entropy-based security notions (similar to [12]), which incorporates, simultaneously, privacy loss due to cryptography in use and to leaking result functions.

A potential disadvantage however is that direct simulation-based proofs are harder to get right; an incorrect proof does not prove anything, however intuitive the functionality involved might be. Furthermore, simulation based-definitions are quite constraining, as far as security against adaptive corruption is concerned.

[3] For readers familiar with the Universal Composability (UC) notion of security: our simulation-based model can be used to show UC security for a particular functionality. We sketch this in our paper but we do not formally introduce the UC framework and do not rely on any result holding in this framework (e.g., composition.)

Summary We review existing game-based privacy notions and find them insufficient. We develop a new privacy notion BPRIV based on our insights from the reviewed notions and subject it to three tests to assure its validity: (1) we prove that Helios meets the new BPRIV notion (2) we check that the new notion does not fail under any of the counter-examples that we identified for existing notions and (3) we prove that BPRIV (together with strong consistency) implies a simulation-based notion of security. To our knowledge, BPRIV is the first game-based notion that comes with a justification in a simulation-based model too. As a corollary we obtain the first simulation-based security proof for Helios. In addition, BPRIV is the first notion to capture security requirements for two features found in real voting systems: (1) revote policies and (2) auxiliary data output by the tallying algorithm.

2 Terminology and basic properties

We first introduce some terminology. In particular, we consider here single-pass voting systems where voters only have to post a single message to the ballot box to cast their vote.

2.1 Single-Pass Voting

A voting scheme is relative to a result function $\rho : (I \times \mathbb{V})^* \to R$ where I is the set of voters identifiers, $\mathbb{V}$ is the set of possible votes, and R is the result space. Depending on the voting scheme, voters identifiers range from voters public identity (appended to ballots in Helios [8] for example) to pseudonyms as well as sets of private credentials. For example in Civitas [20], voters are supposed to use invalid credentials when under coercion and valid ones when they wish to cast their true vote.

The result function explains how the election system should behave. When voters cast their votes, possibly several times, ρ is in charge of specifying which votes shall be counted and how. This involves typically two main tasks. First, usually a *revote policy* specifies which votes shall be retained. A typical revote policy consists in keeping the last cast vote from each voter. However, more complex revote policies can be considered such as computing the average of the votes or counting a vote at a polling station in preference to all online votes[4] (for the same voter). Then, once the revote policy has been applied, votes are counted with some appropriate counting function. Typical counting functions are:

- the *multiset function* multiset that discloses the sequence of all the casted votes, in a random order;
- the *counting function* counting that tells how many votes each candidate received;
- the *majority function* majority that only discloses the winner.

We consider a set I of voters, a subset $\mathcal{H} \subseteq I$ of honest voters, a ballot box BB and a public bulletin board PBB. A single-pass protocol π_ρ executes in three phases.

1. In the *setup* phase, the ballot box expects one message from an administrator after which it may either transition to the *voting* phase or abort.
2. In the *voting* phase, all parties may post messages to the ballot box at any time; the ballot box decides to accept or reject a message based on its current state and a public algorithm. The ballot box stores all accepted messages. Any party may ask to read the bulletin board at any time; the ballot box replies by running a public filtering[5]algorithm (that we will call Publish) on its current state and returns the result. Once the voting phase is closed, the ballot box transitions to the *result* phase.
3. In the *result* phase, the ballot box is closed: it accepts no more messages but can still be read. The administrator computes the final outcome r and a proof of valid tabulation Π via a tallying procedure that may operate on the ballot box.

[4] This is the policy in Estonian elections, for example.

[5] Filtering serves several purposes. In an election where only the last vote from each voter counts, the filter may return only last ballot from each voter. In Helios, there is a “short” board consisting only of hashes of the ballots; this is sufficient to check whether one’s ballot has been included and could again be modelled as a filter. Such a filter could even be used as a defense against ballot-copying, by not revealing the full ballots until the voting phase is closed.

Formally, a voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Publish}, \mathsf{Valid}, \mathsf{Tally}, \mathsf{Verify})$, for a list of voters I and a result function ρ, consists of six algorithms with the syntax given below. The new features of our formalisation are (1) the Publish algorithm, which allows ballot boxes to store more information than they display to the public (i.e. they may keep invalid ballots internally but not display them) and (2) we let each ballot have an explicit identity, which seems to be required to model revoting policies (i.e. "the last ballot from each voter counts"). Our model matches how e.g. Helios handles identities in practice.

- $\mathsf{Setup}(\lambda)$ on input a security parameter λ outputs an election public key $\mathbf{pk}$ and a secret tallying key $\mathbf{sk}$. We assume $\mathbf{pk}$ to be an implicit input of the remaining algorithms.
- $\mathsf{Vote}(id, v)$ is used by voter id to cast his vote $v \in \mathbb{V}$ for the election, as a ballot $b \leftarrow \mathsf{Vote}(id, v)$[6].
- $\mathsf{Valid}(\mathsf{BB}, b)$ takes as input the ballot box BB and a ballot b and checks it validity. It returns $\top$ for valid ballots and $\perp$ for invalid ones (ill-formed, contains duplicated ciphertext from the ballot box, etc.).
- $\mathsf{Publish}(\mathsf{BB})$ takes as input the ballot box BB and outputs the public view PBB of BB, called public bulletin board.
- $\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ takes as input the ballot box BB and the secret key $\mathbf{sk}$. It outputs the tally r, together with a proof of correct tabulation Π. Possibly, $r = \perp$. This algorithm might be (partially) in charge of implementing the revote policy.
- $\mathsf{Verify}(\mathsf{PBB}, r, \Pi)$ takes as input a public bulletin board PBB, and a result/proof pair (r, Π) and checks whether Π is a valid proof of correct tallying for r, it returns $\top$ if so, otherwise it returns $\perp$.

Any voting protocol should ensure that if everyone acts correctly, the protocol indeed computes ρ on the votes submitted. Formally, a voting scheme is *correct* if the following properties hold with overwhelming probability. Let $v_1, \ldots, v_n \in \mathbb{V}$ be valid votes and $id_1, \ldots, id_n \in I$ be voter identities. We consider an honest execution. Let $(\mathbf{pk}, \mathbf{sk}) \leftarrow \mathsf{Setup}(\lambda)$. Let $b_i \leftarrow \mathsf{Vote}(id_i, v_i)$ and $\mathsf{BB}_i := [b_1, \ldots, b_i]$ for all i. Then honest ballots are valid, that is, $\mathsf{Valid}(\mathsf{BB}_{i-1}, b_i) = \top$ for all i; and the protocol computes the correct election result, that is, let $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_n, \mathbf{sk})$ then $r = \rho((id_1, v_1), \ldots, (id_n, v_n))$ and $\mathsf{Verify}(\mathsf{Publish}(\mathsf{BB}_n), r, \Pi) = \top$.

An important contribution of our work is an explicit formalisation of revote policies. Intuitively, there are two main and distinct reasons for removing a ballot:

- *Cryptographic cleaning*: This deletion might be due to the cryptographic implementation. In Helios for example, a ballot may be removed because it is ill-formed (e.g. invalid zero-knowledge proofs) or because it contains a duplicated ciphertext, which may yield a privacy breach [25]. In our formalism, this cleaning operation can be taken care of either by the Valid predicate or by the Tally function itself.
- *Mandatory revote policy*: This deletion might correspond to the implementation of the "ideal" revote policy specified by the result function ρ. A typical revote policy is that, for each voter, only the last ballot shall be retained.

2.2 Tally uniqueness

We define tally uniqueness, a "minimal" property that any verifiable system should satisfy. This property is of course not mandatory for ballot privacy but we will use it in the next sections to illustrate that some ballot privacy definitions are incompatible with any verifiable system.

Intuitively, tally uniqueness of a voting scheme ensures that two different tallies $r \neq r'$ for the same board cannot be accepted by the verification algorithm, even if all the players in the system are malicious. The goal of the adversary against tally uniqueness is to output a public key $\mathbf{pk}$, a list of legitimate public identities, a ballot box BB, and two tallies $r \neq r'$, and corresponding proofs of valid tabulation Π and Π', such that both pass verification. We define tally uniqueness by the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{uniq}}$ in Figure 1. A voting scheme $\mathcal{V}$ has *tally uniqueness* if $\mathsf{Succ}^{\mathsf{uniq}}(\mathcal{A})$ is a negligible function for any PPT adversary $\mathcal{A}$, where

$$\mathsf{Succ}^{\mathsf{uniq}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{uniq}}(\lambda) = 1\right]$$

Tally uniqueness is often considered as a requirement for *election verifiability* [31, 36].

[6] We do not include in our syntax, as most of the related work in the area, the necessary algorithms that allow an election administrator to distribute credentials among users, that will be in turn used to authenticate the voter to the ballot box and cast a ballot.

```
Experiment $\mathsf{Exp}^{\mathsf{uniq}}_{\mathcal{A}}(\lambda)$
(1) $(\mathbf{pk}, \mathsf{BB}, r, \Pi, r', \Pi') \leftarrow \mathcal{A}(1^\lambda)$
(2) $\mathsf{PBB} \leftarrow \mathsf{Publish}(\mathsf{BB})$
(3) if $r' \neq r$ and
(4) $\mathsf{Verify}(\mathsf{PBB}, r, \Pi) = \mathsf{Verify}(\mathsf{PBB}, r', \Pi') = \top$
      return $1$ else return $0$
```

Fig. 1. Tally Uniqueness

3 Survey and analysis of previous game-based computational vote privacy definitions

Game-based privacy [10, 7, 9, 11, 14, 13, 29, 15, 23, 16] (our terminology) definitions require that it should be hard for an adversary to win a game with a challenger behaving in a fully specified manner, in the spirit of traditional indistinguishability definitions for encryption. In this section we review the most relevant ballot privacy definitions in the literature. We unveil that some previous definitions present shortcomings that have gone unnoticed: either ballot secrecy and election verifiability are incompatible [15], or ballot secrecy does not guarantee that the choices of the voters remain private once the election result is published [13, 16]. We also discuss known limitations of the approaches in [10, 9, 7, 31, 33]. A summary of our findings is shown in Table 1, at the end of this section.

Most existing definitions do not distinguish between the ballot box and what is actually published, i.e. the bulletin board. Therefore, in this section, we implicitly assume $\mathsf{Publish}(\mathsf{BB}) = \mathsf{BB}$. Moreover, some definitions do not model voters identifiers. By a slight abuse of notation, we may write $\mathsf{Vote}(v)$ instead of $\mathsf{Vote}(id, v)$ when id is ignored.

We formalize ballot privacy definitions by using two ballot boxes BB_0 and BB_1, from which only one box will be visible to the adversary. Of course, the adversary's goal is to learn which one of them is visible. For the uniformity of the presentation, we write explicitly both ballot boxes in each of the definitions that follow, even if some of them can be defined using only one ballot box (such as Definitions 3.1 and 3.2).

We use the notation and some terminology that we fix above to discuss the existing game-based definitions. We proceed following, roughly, the chronological order.

3.1 Ballot privacy for permutations of honest votes - PODC 1986 [10], STOC 1994 [9]

One first definition of ballot privacy follows a simple idea: an attacker should not notice if the votes of two voters are swapped. More precisely, a coalition of voters should not be able to distinguish when two honest voters id_0, id_1 vote respectively v_0 and v_1, from the case where they vote respectively v_1 and v_0.

Definition 1 (IND-BB). *I is a list of voters.* $\mathsf{BB}_0, \mathsf{BB}_1$ *are lists initialized at empty. The challenger starts by picking a random bit* β*, and the adversary* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ *is given access to lists* I *and* BB_β*. The challenger runs the setup algorithm and the keys* $(\mathbf{pk}, \mathbf{sk})$ *are created. The adversary* $\mathcal{B}_1$ *can repeatedly query the oracle* $\mathcal{O}\mathsf{cast}$ *as follows:*

- $\mathcal{O}\mathsf{cast}(id, b)$*: runs* $bb \mapsto bb \| b$ *on ballot boxes* BB_0 *and* BB_1*. (The expression* $bb \| b$ *appends* b *to* bb*.)*

The adversary can also query once *an oracle* $\mathcal{O}\mathsf{VoteIND}(\cdot, \cdot)$ *as follows:*

- $\mathcal{O}\mathsf{VoteIND}(id_0, id_1, v_0, v_1)$ *: if* $v_\delta \notin \mathbb{V}$ *for* $\delta = 0, 1$*, halt. Else, runs* $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| \{\mathsf{Vote}(id_0, v_0), \mathsf{Vote}(id_1, v_1)\}$ *as well as* $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| \{\mathsf{Vote}(id_0, v_1), \mathsf{Vote}(id_1, v_0)\}$.

At some point, the adversary $\mathcal{B}_1$ *asks to see the result. The challenger computes* $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_\beta, \mathbf{sk})$. *Finally the* IND-BB *adversary* $\mathcal{B}_2$ *outputs* β' *as the guess for* β*.*

Formally, we say that a voting scheme $\mathcal{V}$ *is* IND-BB *secure if no PPT algorithm* $\mathcal{B}$ *can distinguish between the outputs in the experiment just described for* $\beta = 0$ *and* $\beta = 1$*, i.e. for any PPT adversary* $\mathcal{B}$*,*

$$\left| \Pr\left[\mathsf{Exp}^{\mathsf{indbb},0}_{\mathcal{B}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{indbb},1}_{\mathcal{B}}(\lambda) = 1\right] \right|$$

is negligible, where $\mathsf{Exp}^{\mathsf{indbb},\beta}_{\mathcal{B}}$ *is the experiment defined above.*

The definition IND-BB can be seen as a generalization of the *private elections* definition by Benaloh and Yung [10], which was defined only with respect to referendum elections, so that only $v_0 = 0$ and $v_1 = 1$ were considered, and by Benaloh and Tuinstra [9]. It also resembles the symbolic *vote privacy* definition by Delaune, Kremer and Ryan [31], which states that "no party receives information which would allow them to distinguish one situation from another one in which two voters swap their votes". Dreier, Lafoucarde and Laknech have generalised the swap-equivalent symbolic privacy definition to weighted votes [33].

However IND-BB privacy does not guarantee indistinguishability between different assignments that lead to the same result but which are not equivalent permutation-wise. For instance, consider the case where voters can give a score of 0, 1, or 2 to a (single) candidate. It may be the case that an attacker cannot distinguish the sequences of votes $[(id_a, 0); (id_b, 2)]$ from $[(id_a, 2); (id_b, 0)]$ but could well distinguish $[(id_a, 0); (id_b, 2)]$ from $[(id_a, 1); (id_b, 1)]$. In fact, this example serves as a simplified abstraction of actual voting rules, such as those in the European Parliament elections in Luxembourg [2] or the Swiss Government Federal Elections. Definitions [10, 9, 31] do not capture these real cases.

3.2 Benaloh's ballot privacy [7]

To cope with the aforementioned limitation, [7] has generalized the previous definition to an arbitrary set of voters that may vote arbitrarily provided that the tally corresponding to honest voters remains unchanged.

The following definition is a restatement, using contemporary notation, of Benaloh's privacy definition for voting schemes [7].

Definition 2 (PRIV). $\mathsf{BB}_0, \mathsf{BB}_1, V_0, V_1$ *are lists initialized at empty. The challenger starts by picking a random bit* β*, and adversary* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ *is given access to list* BB_β*. The challenger runs the setup algorithm and the keys* $(\mathbf{pk}, \mathbf{sk})$ *are created.* $\mathcal{B}_1$ *on input* $\mathbf{pk}$ *is given access to oracles* $\mathcal{O}\mathsf{vote}(\cdot), \mathcal{O}\mathsf{ballot}(\cdot)$ *as follows:*

- $\mathcal{O}\mathsf{vote}(id, v_0, v_1)$: *runs* $\mathsf{BB}_\delta \leftarrow \mathsf{BB}_\delta \| \mathsf{Vote}(id, v_\delta)), V_\delta \leftarrow V_\delta \| (id, v_\delta))$ *for* $\delta = 0, 1$.
- $\mathcal{O}\mathsf{ballot}(b)$*: runs bb* $\mapsto$ *bb*$\|b$ *on ballot boxes* $\mathsf{BB}_0, \mathsf{BB}_1$ *(that is, appends b to both boards).*

At some point, $\mathcal{B}_2$ *asks to see the tallying output. The challenger proceeds as follows: if* $\rho(V_0) \neq \rho(V_1)$*, halts. Otherwise, the challenger outputs* $(r_\beta, \Pi_\beta) \leftarrow \mathsf{Tally}(\mathsf{BB}_\beta, \mathbf{sk})$*. Finally,* $\mathcal{B}_2$ *outputs* β' *a the guess for* β*. Formally, we declare a voting scheme* $\mathcal{V}$ Benaloh's private *if for any PRIV adversary* $\mathcal{B}$*,*

$$\left| \Pr\left[\mathsf{Exp}^{\mathsf{priv},0}_{\mathcal{B}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{priv},1}_{\mathcal{B}}(\lambda) = 1\right] \right|$$

is negligible, where $\mathsf{Exp}^{\mathsf{priv},\beta}_{\mathcal{B}}$ *is the experiment defined above.*

The main drawback of Benaloh's definition is that it restricts the set of functions for which eventually a scheme could be declared ballot private. To see this, let us assume that the possible votes are $\{a, b\}$ and consider the result function `majority`. We assume that a wins in case of a tie. Let $\mathcal{B}$ be an adversary that chooses to play the game such that $V_0 = [(id_1, a); (id_2, a)]$ and $V_1 = [(id_1, a); (id_2, b)]$. Clearly $\mathtt{majority}(V_0) = \mathtt{majority}(V_1) = a$ (V_1 is a tie) so a wins). Let now $\mathcal{B}$ cast a ballot query for vote b. Then $r_0 = a$ and $r_1 = b$ and thus $\mathsf{Succ}^{\mathsf{priv}}(\mathcal{B}) = 1$, independently of the scheme $\mathcal{V}$.

3.3 Ballot privacy ESORICS 2011 - CCS 2012 [11, 14, 12]

More recently, a definition has been proposed, that aims to deal with arbitrary result functions. It is inspired by cryptographic security for encryption and roughly says that an attacker should not be able to tell whether he is seeing the real board or a fake board where all (honest) voters vote for the same arbitrary dummy vote. Of course, the tally itself is always performed on the real ballot box.

Definition 3 (BPRIV1). $\mathsf{BB}_0, \mathsf{BB}_1, \mathsf{BB}'$ *are lists initialized at empty. A distinguished vote value* $\epsilon \in \mathbb{V}$ *is chosen by the challenger. The challenger starts by picking a random bit* β*, and the adversary* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ *is given access to list* BB_β*. The challenger runs the setup algorithm to create keys* $(\mathbf{pk}, \mathbf{sk})$*. Next, the adversary* $\mathcal{B}_1$ *can query oracles* $\mathcal{O}\mathsf{vote}(\cdot)$ *and* $\mathcal{O}\mathsf{ballot}(\cdot)$ *as follows:*

- $\mathcal{O}\mathsf{vote}(v)$: *computes* $b_0 := \mathsf{Vote}(v)$ *and* $b_1 := \mathsf{Vote}(\epsilon)$*; next runs* $\mathsf{BB}_\beta \leftarrow \mathsf{BB}_\beta \| b_\beta$ *and* $\mathsf{BB}' \leftarrow \mathsf{BB}' \| b_0$.
- $\mathcal{O}\mathsf{ballot}(b)$*: runs* $bb \leftarrow bb \| b$ *on inputs* $\mathsf{BB}_\beta, \mathsf{BB}'$.

At some point, the adversary $\mathcal{B}_2$ *asks to see the tallying output. The challenger obtains* $(r, \mathit{\Pi}) \leftarrow \mathsf{Tally}(\mathsf{BB}', \mathbf{sk})$ *and returns* r *to the adversary. Finally the* BPRIV1 *adversary* $\mathcal{B}_2$ *outputs* β' *as its guess for* β.

Formally, we say that a voting scheme $\mathcal{V}$ *is* BPRIV1 secure *if no PPT algorithm* $\mathcal{B}$ *can distinguish between the outputs in the previous experiment for* $\beta = 0$ *and* $\beta = 1$*, i.e. for any PPT adversary* $\mathcal{B}$*,*

$$\left| \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{bpriv1},0}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{bpriv1},1}(\lambda) = 1\right] \right|$$

is negligible, where $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{bpriv1},\beta}$ *is the experiment defined above.*

In the BPRIV1 definition, tally is executed either over the faithful ballot box (namely, BB), or a fake box (namely, BB'), containing fake votes ϵ for honest voters. A limitation of this definition is that the adversary is not allowed to see the auxiliary data $\mathit{\Pi}$. Indeed in most cases, if the adversary can see $\mathit{\Pi}$ (e.g. a proof of correct tally of the ballot box), he would be immediately able to tell whether he has seen the real or the fake board. Therefore BPRIV1 does not fully model verifiable voting protocols such as Helios or Civitas.

Recently, Cuvelier, Pereira and Peters [29] proposed a variant of the BPRIV1 setting/definition to capture ballot privacy even in the presence of computationally unbounded adversaries, and which is named *perfectly private audit trail* (PPAT). Similarly to BPRIV1, the definition PPAT is limited in the sense that an adversary is not allowed to see the auxiliary data $\mathit{\Pi}$.

3.4 Ballot privacy - ASIACRYPT 2012 [13, 23]

A variation of the BPRIV1 definition has been proposed, that we name as BPRIV2 privacy. Its goal is to be able to fully model verifiable voting protocols where the tally does not produce just a result but also proofs of correct tally. Intuitively, to avoid that the adversary immediately wins the game because of the auxiliary data, this data needs to be simulated on the fake board. This definition makes therefore use of a simulator $\mathsf{SimProof}$ that simulates the auxiliary data of the tally when the adversary is not given the real board.

Definition 4 (BPRIV2). $\mathsf{BB}_0, \mathsf{BB}_1, L$ *are lists initialized at empty. The challenger starts by picking a random bit* β*, and the adversary* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ *is given access to list* BB_β*. The challenger runs the setup algorithm to create keys* $(\mathbf{pk}, \mathbf{sk})$*.* $\mathcal{B}_1$ *is given access to oracles* $\mathcal{O}\mathsf{corrupt}$, $\mathcal{O}\mathsf{voteLR}$ *and* $\mathcal{O}\mathsf{ballot}(\cdot)$ *as follows:*

- $\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$: *if* $v_\gamma \notin \mathbb{V}$ *for* $\gamma = 0, 1$*, halts. Else, runs* $\mathsf{BB}_\gamma \leftarrow \mathsf{BB}_\gamma \| \mathsf{Vote}(id, v_\gamma))$ *for* $\gamma = 0, 1$ *and sets* $L \leftarrow L \cup \{id\}$*, meaning that id has already voted.*
- $\mathcal{O}\mathsf{cast}(id, b)$*: if* $id \in L$ *(a honest user already cast a ballot), halts. Else runs* $\mathsf{BB}_\beta \leftarrow \mathsf{BB}_\beta \| b$.

At some point, $\mathcal{B}_2$ *asks to see the tallying output. The challenger proceeds as follows: if* $\beta = 0$*, the challenger outputs* $(r, \mathit{\Pi}) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$*. But if* $\beta = 1$*, the challenger sets* $(r, \mathit{\Pi}^\dagger) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$ *and* $\mathit{\Pi} \leftarrow \mathsf{SimProof}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \mathtt{info})$*, where* `info` *contains any information known to the challenger. The challenger outputs* $(r, \mathit{\Pi})$.

Finally the BPRIV2 adversary $\mathcal{B}_2$ *outputs* β' *a the guess for* β.

We say that a voting scheme $\mathcal{V}$ *is* BPRIV2 secure *if no PPT algorithm* $\mathcal{B}$ *can distinguish between the outputs in the experiment just described for* $\beta = 0$ *and* $\beta = 1$*, i.e. for any PPT adversary* $\mathcal{B}$*, there exists a simulator* $\mathsf{SimProof}$ *such that*

$$\left| \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{bpriv2},0}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{bpriv2},1}(\lambda) = 1\right] \right|$$

is negligible, where $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{bpriv2},\beta}$ *is the experiment defined above.*

Unfortunately, allowing the auxiliary data to be simulated actually weakens too much the definition of ballot privacy. We show next that BPRIV2 privacy declares as private, protocols that reveal *exactly* how voters voted in the tabulation proof Π. Such protocols should clearly not be declared private.

BPRIV2 fails to ensure ballot privacy Let $\mathcal{V}'$ be any BPRIV2 secure scheme. We assume that ballots of $\mathcal{V}'$ can be extracted, that is, we assume a function $\mathsf{Extract}(\mathbf{sk}, b)$ that returns the vote v corresponding to ballot b. For example, if b contains the encryption of v then $\mathsf{Extract}$ is simply the decryption function. Let $\mathsf{Leak}(\mathcal{V}')$ be the scheme obtained from $\mathcal{V}'$ such that the tally now outputs the correspondence between ballots and votes. Formally, $\mathsf{Leak}(\mathcal{V}')$ is obtained from $\mathcal{V}'$ by changing $\mathsf{Tally}', \mathsf{Verify}'$ to $\mathsf{Tally}, \mathsf{Verify}$ as follows:

- $(r, \Pi) \leftarrow \mathsf{Tally}(\mathbf{sk}, \mathsf{BB})$, where $(r, \Pi') \leftarrow \mathsf{Tally}'(\mathbf{sk}, \mathsf{BB})$ and $\Pi \leftarrow \Pi' \,||\, \{(\, b, v_b\,)\}_{b \in \mathsf{BB}}$, where $v_b \leftarrow \mathsf{Extract}(\mathbf{sk}, b)$;
- $\mathsf{Verify}(\mathsf{PBB}, r, \Pi)$ parses Π as $\Pi' \,||\, \{(\, b, v_b\,)\}_{b \in \mathsf{BB}}$ and outputs $\mathsf{Verify}'(\mathsf{PBB}, r, \Pi')$.

We write $\mathcal{V} := \mathsf{Leak}(\mathcal{V}')$. Intuitively, it is easy to see that $\mathcal{V}$ satisfies BPRIV2 although it is not private. Indeed, the simulator $\mathsf{SimProof}$ knows all the $\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$ queries made by the adversary. It may therefore pretend that any ballot b corresponding to a query $\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$ corresponds to v_0 even when it corresponds to v_1 (when $\beta = 1$). This argument is worked out in detail in Appendix B.

3.5 Ballot secrecy - ESORICS 2013 [15]

Another definition, called IND-SEC, has been recently been proposed [15] to fix the privacy breach of the definition BPRIV2. This definition can be seen as a combination of the BPRIV2 [13, 23] and the Benaloh's [7] definitions. Indeed, the IND-SEC game is intuitively defined as follows. The honest voters vote for an arbitrary sequence of votes V_0 in the first board and an other arbitrary sequence of votes V_1 in the second board. If the two sequences coincide when viewed as multisets, then the real tally is disclosed (as in Benaloh's [7] definition). If the two multisets differ, the tally is always performed on the first ballot box (even when the adversary has seen the second ballot box).

Definition 5 (IND-SEC). $\mathsf{BB}_0, \mathsf{BB}_1, V_0, V_1$ *are lists initialized at empty. The challenger starts by picking a random bit* β*, and the adversary* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ *is given access to list* BB_β*. The challenger runs the setup algorithm and the keys* $(\mathbf{pk}, \mathbf{sk})$ *are created. On input* $\mathbf{pk}$ *the adversary* $\mathcal{B}_1$ *can query oracles* $\mathcal{O}\mathsf{voteLR}(\cdot, \cdot)$ *and* $\mathcal{O}\mathsf{ballot}(\cdot)$ *as follows:*

- $\mathcal{O}\mathsf{vote}(v_0, v_1)$: *runs* $\mathsf{BB}_\gamma \leftarrow \mathsf{BB}_\gamma \| \{\mathsf{Vote}(v_\gamma)\}$*, and updates* $V_\gamma \leftarrow V_\gamma \cup \{v_\gamma\}$ *for* $\gamma = 0, 1$.
- $\mathcal{O}\mathsf{ballot}(b)$*: runs* $bb \mapsto bb \| b$ *on ballot boxes* $\mathsf{BB}_0, \mathsf{BB}_1$.

At some point, $\mathcal{B}_2$ *asks to see the tallying output. The challenger proceeds as follows:*

- *If* $V_0 = V_1$ *(as multisets) the output is set to be that of* $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_\beta, \mathbf{sk})$.
- *If* $V_0 \neq V_1$ *the challenger outputs* $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$.

Finally the IND-SEC *adversary* $\mathcal{B}_2$ *outputs* β' *a the guess for* β.

Formally, we say that a voting scheme $\mathcal{V}$ *has* IND-SEC secrecy *if no PPT algorithm* $\mathcal{B}$ *can distinguish between the outputs in the experiment just described for* $\beta = 0$ *and* $\beta = 1$*, i.e. for any PPT adversary* $\mathcal{B}$*,*

$$\left| \ \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{indsec},0}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{indsec},1}(\lambda) = 1\right] \ \right|$$

is negligible, where $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{indsec},\beta}$ *is the experiment defined above.*

IND-SEC and Tally Uniqueness are Incompatible. While IND-SEC declares the $\mathsf{Leak}(\mathcal{V})$ voting scheme insecure, it turns out that IND-SEC secrecy and verifiability are incompatible properties. In other words, any verifiable protocol will be declared not private by IND-SEC.

Let $\mathcal{V}$ be a (correct) voting scheme with tally uniqueness for a non-trivial result function ρ. We describe next a IND-SEC adversary $\mathcal{B}$ that has advantage negligibly close to 1/2 against $\mathcal{V}$ in the IND-SEC game. More precisely, for any IND-SEC adversary $\mathcal{B}$ there exists a tally uniqueness adversary $\mathcal{B}'$ such that $\mathsf{Succ}^{\mathsf{bpriv1}}(\mathcal{B}) \geq 1 - \mathsf{Succ}^{\mathsf{uniq}}(\mathcal{B}')$. Therefore both advantages can not be negligible at the same time, and thus IND-SECl and tally uniqueness are incompatible properties.

The adversary $\mathcal{B}$ proceeds as follows. It chooses votes v, ϵ such that $\rho(v) \neq \rho(\epsilon)$ and it makes a single query $\mathcal{O}\mathsf{vote}(v, \epsilon)$, which causes $b_0 := \mathsf{Vote}(v)$ and $b_1 := \mathsf{Vote}(\epsilon)$ to be created. Then $\mathcal{B}$ sets its guess $\beta' := 0$ if $\mathsf{Verify}(\mathsf{BB}_\beta, \rho(v), \Pi) = \top$, where Π is computed by the IND-SEC challenger as $(r, \Pi) \leftarrow \mathsf{Tally}(\{b_0\}, \mathbf{sk})$; otherwise sets $\beta' := 1$. The claim follows from the following facts:

- $\mathsf{Verify}(\{b_0\}, \rho(v), \Pi) = \top$, since $\mathcal{V}$ is correct;
- $\mathsf{Verify}(\{b_1\}, \rho(\epsilon), \Pi_1) = \top$, where Π_1 is not explicitly known but it is defined by $(r_1, \Pi_1) \leftarrow \mathsf{Tally}(\{b_1\}, \mathbf{sk})$. This holds since $\mathcal{V}$ is correct;
- $\mathsf{Verify}(\{b_1\}, \rho(v), \Pi) = \bot$ with overwhelming probability. Indeed, since $\mathcal{V}$ has tally uniqueness, and given that $\mathsf{Verify}(\mathsf{Publish}(\{b_1\}), \rho(\epsilon), \Pi_1) = \top$, the equation $\mathsf{Verify}(\mathsf{Publish}(\{b_1\}), \rho(v), \Pi) = \top$ for $\rho(v) \neq \rho(\epsilon)$ is satisfied only with negligible probability.

The latter implies in particular that Helios, in any of its known flavours, *cannot be* both IND-SEC *private* and *verifiable*.

3.6 Ballot privacy for restricted adversaries - PKC 2013 [16]

Another definition has been recently proposed by Chase *et al.* [16]. As for the previous IND-SEC definition, the adversary triggers honest voters, providing a left and right votes for each voter. The tally is performed on the visible ballot box, that is, the one the adversary sees (no simulation). However, if the result announced differs depending on whether $\beta = 0$ and $\beta = 1$, then the adversary loses the game.

Definition 6 (RPRIV). *$\mathsf{BB}_0, \mathsf{BB}_1, V_0, V_1$ are lists initialized at empty. The challenger starts by picking a random bit β, and the adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is given access to list BB_β. The challenger runs the setup algorithm and the keys $(\mathbf{pk}, \mathbf{sk})$ are created. $\mathcal{B}_1$ on input $\mathbf{pk}$ is given access to oracles $\mathcal{O}\mathsf{vote}(\cdot), \mathcal{O}\mathsf{ballot}(\cdot)$ as follows:*

- *$\mathcal{O}\mathsf{vote}(id, v_0, v_1)$: runs $\mathsf{BB}_\delta \leftarrow \mathsf{BB}_\delta \| \mathsf{Vote}(id, v_\delta)), V_\delta \leftarrow V_\delta \cup \{v_\delta\}$ for $\delta = 0, 1$.*
- *$\mathcal{O}\mathsf{ballot}(id, b)$: runs $bb \mapsto bb \| b$ on ballot boxes $\mathsf{BB}_0, \mathsf{BB}_1$.*

At some point, $\mathcal{B}_2$ asks to see the tallying output. The challenger computes $(r_0, \Pi_0) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$ and $(r_1, \Pi_1) \leftarrow \mathsf{Tally}(\mathsf{BB}_1, \mathbf{sk})$. If $r_0 \neq r_1$, the adversary loses. Otherwise, the challenger replies (r_β, Π_β) to $\mathcal{B}_2$. Finally, $\mathcal{B}_2$ outputs β' a the guess for β. Formally, we declare a voting scheme $\mathcal{V}$ RPRIV private *if for any adversary $\mathcal{B}$ the advantage*

$$\left| \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{rpriv},0}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathcal{B}}^{\mathsf{rpriv},1}(\lambda) = 1\right] \right|$$ *is negligible, where $\mathsf{Exp}_{\mathcal{B}}^{\mathsf{rpriv},\beta}$ is the experiment above.*

This definition fails to capture replay attacks, whereby a malicious voter replays a previously ballot cast by a honest voter. It is known that replay attacks violate ballot secrecy. Indeed, consider a referendum election with three voters, namely, Alice, Bob, and Mallory: if Mallory replays Alice's ballot without being detected or rejected, then Mallory can reveal Alice's vote by observing the election outcome and checking which candidate obtained at least two votes. This attack was successfully implemented against Helios 2.0 by Cortier and Smyth [25, 26], who also showed this constitutes a privacy threat in real scenarios by studying the potential impact on French legislative elections.

	ESOR.11 [11]	ESOR.09 [31, 33]	PODC86 [10] STOC94 [9]	Benaloh [7]	ESOR.13 [15]	ASIA. 12 [13]	PKC13 [16]	S&P 10 [37]	CCS12 [12]	ACNS04 [35]	Ours Sec. 4
game-based notion	✓	×	✓	✓	✓	✓	✓	×	×	×	✓
detects leaky revote policies	×	×	×	×	×	×	×	✓	**?**	**?**	✓
Helios 2.0 is not private	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓
protects ag. vote comparisons	✓	×	×	✓	✓	✓	×	×	✓	✓	✓
compatible with tally uniqueness	**?**	✓	✓	✓	×	✓	✓	✓	✓	✓	✓
admits duplicate weed before tally	✓	✓	×	×	×	✓	✓	✓	✓	**?**	✓
admits duplicate weed inside tally	×	✓	×	×	×	×	×	✓	×	**?**	✓
detects leaky tally proofs	**?**	✓	✓	✓	✓	×	✓	✓	✓	✓	✓
revoting allowed	✓	✓	✓	✓	✓	✓	✓	✓	**?**	×	✓
models partially hidden board	×	×	×	×	×/✓	×	×	×	×	×	✓
admits result functions w/o partial tallying	✓	×	×	×	✓	✓	**?**	×	✓	×	✓

Table 1. Summary of our survey. ✓= definition satisfies a desirable property (the more ✓s a definition has, the better); ×= definition does not satisfy the property; **?** = result is not addressed in the definition or we could not establish it from the reference.

Let us argue that RPRIV does not capture replay attacks. Consider a protocol where voters submit their encrypted votes on a secure channel, to an election server that publishes the encrypted votes on the board (thus allowing ciphertext copying). Assume moreover that only the result of the election is published, once the voting phase is closed. Clearly, this protocol is subject to replay attacks and therefore does not ensure privacy. However, it is declared private by RPRIV.

Indeed, the only information that the adversary obtains is the encrypted ballots and the result of the election. Since the encryption scheme is IND-CPA, the adversary does not get any information from the encrypted votes, so it must use the result itself to try to win the RPRIV distinguishing game. Moreover, by definition of this game, the adversary can see the result only in cases where both ballots boxes yield identical results. Therefore the adversary cannot win the game, since encrypted votes without the result give no information, and the result does not help distinguishing the boxes, since it is identical in both boxes. For similar reasons, RPRIV would declare Helios 2.0 private.

4 Ballot privacy: a comprehensive cryptographic game-based definition for vote privacy

Based on our findings on existing definitions of privacy, we propose a new definition that avoids the aforementioned issues. As in [11, 14, 13, 12, 15], we define ballot secrecy for a voting scheme $\mathcal{V}$ in terms of a game between a challenger and an adversary.

We give our new security notion in two steps. We start with a vanilla variant that reflects the core definitional ideas behind our notion and allows a focused discussion on its features. Then, we explain how to incorporate in the definition the existence of global setup assumption (e.g. random oracles or common reference strings) as used by many existing voting protocols.

Recall that ballot privacy attempts to capture the idea that during its execution a secure protocol does not reveal information about the votes cast, beyond what is unavoidably leaked (e.g. what the result of the election leaks). As in previous definitions, we formalize this idea via an adversary that attempts to distinguish between two worlds. In the first world the adversary has (indirect) access to a ballot box that contains ballots created by honest users as well as adversarial ballots and gets to see the result corresponding to the ballot box. In the second world the adversary sees a fake board instead of the real one, yet gets to see the result of the election as tallied on the real ballot box. Since we model explicitly the additional information that the tally may include besides the result (e.g. a proof of correct tally), we require that this information does not reveal any information either: we require the existence of a simulator that can "fake" the additional information corresponding to the real result, but with respect to the fake ballot box.

We formalize this discussion via the experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ defined in Figure 2. In these games $\mathsf{BB}_0, \mathsf{BB}_1$ are ballot boxes that start out empty. Ballot box BB_0 corresponds to the real election (that will be tallied). The adversary gets access to BB_0 in the first game and access to BB_1 (a fake ballot box) in the second game. The experiment starts with generating long term keys $\mathbf{sk}$ and $\mathbf{pk}$; the adversary $\mathcal{A}$ is given the public key and access to the oracles that we describe below and formalize in Figure 2.

$\mathcal{O}\mathsf{board}$: This models the ability of the adversary to see the publishable part of the ballot box, i.e. the bulletin board. The oracle returns $\mathsf{Publish}(\mathsf{BB}_\beta)$.

$\mathcal{O}\mathsf{voteLR}$: The left-right oracle $\mathcal{O}\mathsf{voteLR}$ takes two potential votes (v_0, v_1) for user id, produces ballots b_0 and b_1 for these votes and places them on the ballot box (one on BB_0 and one one on BB_1), provided that b_β is valid with respect to BB_β.

$\mathcal{O}\mathsf{cast}$: This oracle allows the adversary to cast a ballot b on behalf on any party. If the ballot is valid with respect to BB_β, it is placed on both ballot boxes.

$\mathcal{O}\mathsf{tally}$: This oracle allows the adversary to see the result of the election. In both worlds the result is obtained by tallying BB_0. In the first experiment the additional information calculated by the tally is given to the adversary, whereas in the second experiment the additional information is simulated.

The adversary can call oracles $\mathcal{O}\mathsf{voteLR}, \mathcal{O}\mathsf{cast}, \mathcal{O}\mathsf{board}$ in any order, and any number of times. Finally, $\mathcal{A}$ can call $\mathcal{O}\mathsf{tally}$ once; after it receives the answer to his query $\mathcal{A}$ returns a guess bit on the value of β. This bit is the result returned by the game.

Definition 7 (BPRIV). *Consider a voting scheme $\mathcal{V} = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Publish}, \mathsf{Tally}, \mathsf{Verify})$ for a set I of voter identities and a result function ρ. We say the scheme has ballot privacy if there exists an algorithm* $\mathsf{SimProof}$ *such that no efficient adversary can distinguish between games $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{B},\mathcal{V}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{B},\mathcal{V}}(\lambda)$ defined by the oracles in Figure 2, that is for any efficient algorithm $\mathcal{A}$*

$$\left| \ \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V}}(\lambda) = 1\right] \ \right|$$

is negligible in λ.

4.1 Extension of the definition to setup assumptions

Most voting protocols rely on non-interactive zero knowledge (NIZK) proofs to enforce honest behaviors for the parties involved, yet such proofs require some setup assumptions (like the CRS or the RO model). To be able to analyze these protocols we need to extend our definition to account for such setups. While it would be simple enough to provide a definition of ROM-BPRIV or CRS-BPRIV, we would like our definition to abstract away details of any particular model of zero-knowledge as far as possible. Finding a truly model-agnostic notion of zero-knowledge is an open research problem so we choose to sketch an extension of ballot privacy that keeps the setup assumption abstract thus sacrificing some precision for greater generality.

```
$\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$
    $(\mathbf{pk},\mathbf{sk}) \leftarrow \mathsf{Setup}(1^k)$
    $d \leftarrow \mathcal{A}^{\mathcal{O}}(\mathbf{pk})$
    Output $d$

$\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$
    Let $b_0 = \mathsf{Vote}(id, v_0)$ and $b_1 = \mathsf{Vote}(id, v_1)$.
    If $\mathsf{Valid}(\mathsf{BB}_\beta, b_\beta) = \perp$ return $\perp$.
    Else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b_0$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b_1$

$\mathcal{O}\mathsf{cast}(id, b)$
    If $\mathsf{Valid}(\mathsf{BB}_\beta, b) = \perp$ return $\perp$.
    Else $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b$.

$\mathcal{O}\mathsf{board}()$
    return $\mathsf{Publish}(\mathsf{BB}_\beta)$

$\mathcal{O}\mathsf{tally}()$ for $\beta = 0$                  | $\mathcal{O}\mathsf{tally}()$ for $\beta = 1$
  $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$   |   $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$
                                                  |   $\Pi' \leftarrow \mathsf{SimProof}(\mathsf{BB}_1, r)$
  return $(r, \Pi)$                               |   return $(r, \Pi')$
```

Fig. 2. In the experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ defined above for $\beta = 0, 1$ adversary $\mathcal{A}$ has access to the set of oracles $\mathcal{O} = \{\mathcal{O}\mathsf{cast}, \mathcal{O}\mathsf{board}, \mathcal{O}\mathsf{voteLR}, \mathcal{O}\mathsf{tally}\}$. The adversary is allowed to querry $\mathcal{O}\mathsf{tally}$ only once. For $\beta = 1$ the experiment also depends on $\mathsf{SimProof}$ – we do not show the dependence explicitly to simplify notation.

We also note that this extension is needed, not only to allow for the analysis of more schemes, but also because in its vanilla format BPRIV security would be too strong: the existence of a simulator that can fake the proof without using a global setup, would mean that secure schemes would not satisfy tally uniqueness.

A global setup consists of a set public parameters and algorithms that can be accessed by all parties, and in particular by the adversary. These parameters are initialized at the beginning of the execution. We write $\mathsf{GlobalSetup}.\mathtt{init}$ for the algorithm that initializes the parameters of the global setup, and write $A^{\mathsf{GlobalSetup}}$ to indicate that algorithm A can access $\mathsf{GlobalSetup}$. For example, in the CRS case $\mathsf{GlobalSetup}.\mathtt{init}$ would select a random string (of some fixed length) and $\mathsf{GlobalSetup}$ simply makes this string available to all parties. In the random oracle model $\mathsf{GlobalSetup}$ consist of a truly random function to which parties only have oracle access.

The power of setup assumptions comes from the ability to generate simulated setups, indistinguishable to an adversary from a normal one. The simulated setup however, grant to a simulator additional powers that are useful in crafting reduction proofs. For example in the CRS model the fake setup would consist of a CRS indistinguishable from an honestly generated one, but which comes with a trapdoor that allows, for example, to produce valid looking proofs for false statements. In the random oracle model the oracle is under the control of a simulator who can "program" (i.e. fix) its output on some values of interest. Naturally, this programming should be such that the adversary cannot distinguish the simulated oracle from a truly random one. We write $\mathsf{SimGlobalSetup}$ for a simulated setup and $\mathsf{SimGlobalSetup}.\mathtt{init}$ for its initialization algorithm. When $\mathsf{SimGlobalSetup}$ is used, some of the parties in the system may have access to its associated trapdoor (like in the CRS setting), or control it in some other way (like in the RO model).

The extension of BPRIV to global setups is as follows. In both of the games we initialize a real and a fake setup via $\mathsf{GlobalSetup}.\mathtt{init}$ and $\mathsf{SimGlobalSetup}.\mathtt{init}$. When $\beta = 0$, that is in the game that corresponds to the real execution the adversary has access to $\mathsf{GlobalSetup}$; in this experiment $\mathsf{SimGlobalSetup}$ does not play any role – we simply initialize it for simplicity and for ease of comparing with the other experiment. When $\beta = 1$, the adversary has access to $\mathsf{SimGlobalSetup}$. However, the fake global setup is under the control of $\mathsf{SimProof}$: all calls to $\mathsf{SimGlobalSetup}$ are sent to $\mathsf{SimProof}$ which is in charge of answering them. Importantly, in this experiment the result is produced by tallying the "real" ballot box (BB_0) and using the real $\mathsf{GlobalSetup}$.

We provide a fully worked out instantiation of our definition in the ROM in Appendix C, since this is the model used for Helios.

4.2 Strong Consistency

The ballot privacy definition BPRIV strongly relies on a split between the result r and the auxiliary data Π returned by the tally. This split should be *meaningful*, that is r should correspond to the expected result and should not contain hidden auxiliary data. To enforce this, we introduce a companion definition of BPRIV, called *strong consistency*, that has two main goals. Firstly, it ensures that the result always corresponds to the result function applied to the votes, and nothing more. Secondly, it controls the damages that an intentionally leaky revote policy could *implicitly* cause while tallying. In fact, since tallying takes as inputs the ballot box and the secret key, this allows a malicious designer/election administrator to implement a leaky revote policy. We limit the damages of such a behaviour by asking the voting scheme to satisfy a stronger correctness property, that we name *strong consistency*.

Intuitively, strong consistency guarantees that the Tally algorithm behaves like ρ except possibly on invalid ballots. (Note that BPRIV is independent of ρ.) We formalize the requirement by requiring the existence of an "extraction" algorithm which, with the help of the secret key, can determine for each ballot the underlying vote and the identity/identifier with which it was created or $\perp$ if the ballot is somehow invalid. We require that from an honestly created ballot the extraction algorithm works as desired. Using the extraction algorithm we can capture the intuition that the Tally algorithm works as expected: we demand that the result reported for some valid ballot box BB by Tally is the same as the result function applied to the votes that underlie the ballots on BB, as defined using the extraction algorithm.

Definition 8 (Strong consistency). *A scheme* $\mathcal{V} = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Publish}, \mathsf{Tally}, \mathsf{Verify})$ *relative to a result function* $\rho : (I \times \mathbb{V})^* \to R$ *has strong consistency if there exist*

- *an extraction algorithm* $\mathsf{Extract}$ *that takes as input a secret key* sk *and a ballot* b *and outputs* $(id, v) \in I \times \mathbb{V}$ *or* $\perp$*;*
- *a ballot validation algorithm* $\mathsf{ValidInd}$ *that takes as input the public key of the election* pk *and a ballot* b *and outputs* $\top$ *or* $\perp$*;*

which satisfy the following conditions:

1. *For any* $(\mathsf{pk}, \mathsf{sk})$ *that are in the image of* Setup *and for any* $(id, v) \in I \times \mathbb{V}$ *if* $b \leftarrow \mathsf{Vote}(\mathsf{pk}, id, v)$ *then* $\mathsf{Extract}(\mathsf{sk}, b) = (id, v)$ *with overwhelming probability.*
2. *For any* $(\mathsf{BB}, b) \leftarrow A$*,* $\mathsf{Valid}(\mathsf{BB}, b) = \top$ *implies* $\mathsf{ValidInd}(b) = \top$.
3. *Consider an adversary* A *which is given* pk *and consider the experiment:*

```
$\mathsf{Exp}^{\mathsf{s\text{-}cons}}_{A,\mathcal{V}}(\lambda)$
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(\lambda)$;
    $\mathsf{BB} \leftarrow A$
    $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{sk}, \mathsf{BB})$
    If $r \neq \rho(\mathsf{Extract}(\mathsf{sk}, b_1), \ldots, \mathsf{Extract}(\mathsf{sk}, b_n))$
    Then return 1 Else return 0
```

Above we consider only adversaries A *that return* BB *of the form* $[b_1, \ldots, b_n]$ *such that* $\mathsf{ValidInd}(b_i) = \top$ *for* $i = 1..n$*. We require that the probability* $\Pr[\mathsf{Exp}^{\mathsf{s\text{-}cons}}_{A,\mathcal{V}}(\lambda) = 1]$ *is negligible in the security parameter.*

In case the result function ρ does not use voter identifiers, that is

$$\rho((id_1, v_1), \ldots, (id_n, v_n)) = \rho((id'_1, v_1), \ldots, (id'_n, v_n))$$

for any v_j, id_j, id'_j, then the function $\mathsf{Extract}$ is not required to output the voter identifier.

Informally, strong consistency prevents an adversary from encoding instructions in her own ballots, causing the tallying to leak information on the honest votes or prevent the validation of honestly generated ballot boxes. Indeed, the extraction algorithm works "locally" on each ballot so it cannot respond to instructions encoded in one ballot on how to treat another ballot or to return no result. This is not to say that the tallying algorithm cannot be aware of adversarial instructions — indeed, it is free to write what it likes to the auxiliary data. It is only the result that is protected by strong consistency. The rest is handled by the BPRIV definition.

Just like in the case of the ballot privacy definition, we started with a vanilla variant of strong consistency that does not account for global setups. The definition above can be extended to this more realistic setup by providing to adversary A and to the Tally algorithm access to $\mathsf{GlobalSetup}$ initialized, as usual, via $\mathsf{GlobalSetup} \leftarrow \mathsf{GlobalSetup}.init$.

4.3 Strong Correctness

This notion requires a strong independence relation between honestly created ballots and the ballot box (and the global setup) in the voting scheme, which we capture by requiring that an honestly created ballot is valid, even with respect to an adversarially created ballot box.

Definition 9 (Strong correctness). *Consider an adversary A against π that takes as input* pk *and has access to a global setup* $\mathsf{GlobalSetup}$ *generated as expected. Then,*

$$\Pr[(id, v, \mathsf{BB}) \leftarrow A(\mathsf{pk}); b \leftarrow \mathsf{Vote}(id, v) \ : \mathsf{Valid}(b, \mathsf{BB}) \neq \top]$$

is negligible. The probability is over the coins used by the adversary and Vote*, but also over the coins used in the generation of* pk.

4.4 Necessity of Strong Consistency and Strong Correctness

Let us see an example of a voting scheme that satisfies BPRIV but that implements a leaky revote policy while tallying. Let $\mathcal{V}$ be a BPRIV voting scheme for the multiset result function (i.e. $\rho(v_1, \ldots, v_n)$ outputs $\{v_1, \ldots, v_n\}$ viewed as a multiset), such that $\mathsf{Publish}(\mathsf{BB}) = \mathsf{BB}$, $\Pi = \emptyset$ (i.e. there are no proofs of correct tallying), $\mathsf{Verify}(\mathsf{PBB}, r, \emptyset) = \top$ (i.e. since there are no tallying proofs, we accept any result published by the election administrator) and $\mathbb{V} = \{0, 1\}$. To simplify further, let us assume that $\mathcal{V}$ only allows single voting (i.e. voters cannot revote).

We build a voting scheme $\mathcal{V}'$ that inherits BPRIV privacy from $\mathcal{V}$, but which possibly reveals how the first voter voted, and thus it is, intuitively, not ballot private. $\mathcal{V}'$ is obtained by replacing algorithm Tally by Tally' as follows: $\mathsf{Tally}'(\mathsf{BB}, \mathbf{sk})$ first checks whether the first ballot in BB contains a 1-vote. If so, removes this ballot from BB, and let BB' be the resulting ballot box. Finally, outputs $\mathsf{Tally}(\mathsf{BB}', \mathbf{sk})$. Let us sketch a proof that $\mathcal{V}'$ is BPRIV.

In the first place, let BB_0 contain in its first entry a 1-vote, while BB_1 contains in its first entry a 0-vote. If $\beta = 0$, the adversary sees ballot box BB_0. The output of tallying in this case is $\mathsf{Tally}'(\mathsf{BB}_0, \mathbf{sk}) = \mathsf{Tally}(\mathsf{BB}_0 \setminus \{b_1\}, \mathbf{sk})$. If $\beta = 1$, the adversary sees ballot box BB_1, but still sees the same tallying output $\mathsf{Tally}'(\mathsf{BB}_0, \mathbf{sk})$.

In the second place, let us consider the complementary case where BB_0 contain in its first entry a 0-vote, while BB_1 contains in its first entry a 1-vote. If $\beta = 0$, the adversary sees board BB_0. The output of tallying in this case is $\mathsf{Tally}'(\mathsf{BB}_0, \mathbf{sk}) = \mathsf{Tally}(\mathsf{BB}_0, \mathbf{sk})$. If $\beta = 1$, the adversary sees board BB_1, but still sees the same tallying output $\mathsf{Tally}'(\mathsf{BB}_0, \mathbf{sk})$.

Finally, if BB_0 and BB_1 both contain ballots at their first entry for the same vote $v \in \{0, 1\}$, then Tally' does not help distinguishing when compared to the original Tally. Thus, for every adversary $\mathcal{A}'$ against BPRIV of $\mathcal{V}'$ there exists an adversary $\mathcal{A}$ against BPRIV of $\mathcal{V}$ with roughly the same distinguishing advantage. If $\mathcal{V}$ is BPRIV, so is $\mathcal{V}'$. Unfortunately, $\mathcal{V}'$ reveals how a honest voter voted, since the output of Tally tells whether the first ballot on BB contains the vote 1.

Our solution to defeat these leaky revote policies embedded in Tally is to ask for strong consistency. Indeed, it is easy to see that $\mathcal{V}'$ does not satisfy strong consistency. Note that whenever the first ballot of BB is a 1-vote, then running Tally and running the alternative tally procedure using $\mathsf{Extract}$ results in different outputs.

Regarding the need for strong correctness, let us assume a BPRIV voting scheme $\mathcal{V}'$. We build a new voting scheme $\mathcal{V}$ such that ballots b in $\mathcal{V}$ are obtained by appending a t-bit to ballots b' in $\mathcal{V}'$ (say at the beginning of b'). The vote algorithm in $\mathcal{V}$ just appends a 0-bit to the result of the vote algorithm in $\mathcal{V}'$. Tally in $\mathcal{V}$ just drops the appended bit from any ballot b and next applies the tally from $\mathcal{V}'$. $\mathsf{Valid}(\mathsf{BB}, b)$ is defined as follows:

- if there is any ballot in BB starting with 1-bit, rejects b
- otherwise, drops the the appended bit from any ballot in BB and from b, let BB' and b' be the corresponding outputs, and computes $\mathsf{Valid}'(\mathsf{BB}', b')$

$\mathcal{V}$ is BPRIV but not strongly correct. In particular, a malicious voter can cause votes from honest voters to be rejected. Actually, this remark also applies to other privacy definitions, such as IND-BB.

4.5 Revote policies

Formalizing revote policies has been rarely done in the literature on foundations of electronic voting. Previously, revote policies were considered as an external component of the voting scheme. Roughly speaking, one used to proceed as follows. Firstly the cryptographic workflow of the voting protocol is described, next its ballot privacy is proven, and then the revote policy is decided at the implementation level by the election administrator. The bottom line is that reasonable revote policies will not impact the ballot privacy of the underlying voting scheme, so they can chosen independently of the scheme (we just need to make sure that the protocol implements the given revote policy).

One of our findings is that a good ballot privacy definition must restrict the class of revote policies allowed. This is because the algorithms implementing a revote policy might need the secret key as an input — some voting schemes, such as JCJ/Civitas, need the election secret key to implement revote. This opens the door to intentionally leaky revote policies, which could use this secret key to gain information on voters' votes.

In Civitas [20], ballots with invalid credentials are removed based on a plaintext-equivalence test that uses some trapdoor unavailable to the ballot privacy attacker. This can be captured in a strongly consistent way by including this operation in ρ. Given a ballot b that contains a vote v and a credential $cred$, the $\mathsf{Extract}$ function should return both $cred$ and v. Then ρ should remove any vote that corresponds to an invalid credential.

Other schemes also require the election secret key to be able to clean the ballots. Sometimes, (part of) the private key is needed to determine the validity of ciphertexts, e.g., when Cramer-Shoup encryption is used [42]. This can be captured by having $\mathsf{Extract}$ performing the appropriate tests. In some other cases, e.g., in mixnet-based schemes, the validity of ballots can only be determined even later, after full decryption. We also include this cleaning operation in ρ: whenever ρ sees an invalid vote v, then this vote shall be removed. In this way, Tally remains strongly consistent.

5 A simulation-based model of ballot privacy

In this section we define an ideal functionality for voting protocols, in the spirit of Groth [35] and de Marneffe et al. [30], but handling voters that submit multiple votes. The inspection of this functionality should make it obvious to the reader that, if a trusted party were available to provide its service, then we would have a voting system offering all expected privacy guarantees and that all adversaries against that functionality are harmless.

We show that any voting scheme satisfying BPRIV and strong consistency offers at least as much[7] privacy guarantees as this ideal functionality. This will be demonstrated using the traditional real-world/ideal world paradigm: we show that anything that can be done by an adversary against the voting protocol can also be done by an adversary interacting with the ideal functionality. Since we know that any adversary against the ideal functionality is harmless, this shows that the real protocol adversary must be harmless too. Our treatment leaves out many technical

[7] Actually, BPRIV, strong correctness, and strong consistency are a strictly stronger condition than securely realising our ideal functionality. Essentially, these conditions define one particular simulator that work for the simulation-based definition. The converse implication is not necessarily true.

details of the execution model and focuses on the crucial part. We expect that it could be cast formally in traditional security frameworks in which universal composition theorems hold, even though we are not concerned with any such composition result here.

We start by describing our ideal world setting, then the real world one, and eventually show the implications of our game-based security definitions.

5.1 An ideal functionality for voting

We describe an ideal voting functionality for a set of identities I and a result function $\rho : (I \times \mathbb{V})^* \to R$. This functionality has a simple behavior:

1. It first expects to receive one or more votes, both from honest and from corrupted voters. Every time a vote is received, the identity of the voter and the vote content are stored, and the functionality lets the adversary know who submitted a vote (but not the content of the vote, of course). This captures the idea that submitting a vote is a public action but could be relaxed in settings where voting would be private (though this usually has a cost in terms of eligibility verifiability.)
2. When it receives the order to tally, the functionality evaluates the ρ function on the sequence of pairs of identities and votes that it has received, and sends the result to the adversary.

Following the traditional ideal world/real world terminology, we give the control of the honest voters to an entity called the *environment*, which submits honest votes of its choice directly to the functionality. This single entity coordinating the honest voters ensures that security will be satisfied whoever the honest votes are and whatever distribution they follow. The ideal-world adversary, which we also call the *simulator* (following the tradition), can also submit arbitrary votes to the functionality, on behalf of (maybe temporarily) corrupted voters. Furthermore, this simulator can receive arbitrary information from the environment. This captures information that the adversary could obtain about the honest votes externally from the voting protocol (e.g., through polls, . . .). It also interacts with the environment as part of its adversarial behavior, reporting its achievements. Eventually, the environment outputs a single bit that expresses whether or not it feels that it is interacting in the ideal-wold we just described. If the environment can notice that it is not running in this ideal world when it is actually running with a real protocol, then the real world protocol will be claimed to be insecure.

Definition 10. *The functionality $F_{\mathsf{voting}}(\rho)$, interacting with an environment $\mathfrak{E}$ and an adversary S, proceeds as follows:*

---

1. *On input* $\texttt{vote}(id, v)$ *from $\mathfrak{E}$ or S, store (id, v) and send* $\texttt{ack}(id)$ *to S.*
2. *On input* $\texttt{tally}$ *from S, return to both $\mathfrak{E}$ and S the result of the function ρ applied on the sequence of (id, v) pairs received, then halt.*

This functionality has clear privacy properties: it reveals who votes, and the result of the election, but nothing more. Furthermore, the votes from the simulator are sent to the functionality in complete independence of the honest votes from the environment unless the environment itself tipped the simulator in the first place (which cannot be prevented by any voting system): the functionality does not give S any information related to the honest votes before it provides the election result. This seems to be the best we can hope for in terms of privacy of votes.

The full process in which an environment $\mathfrak{E}$, an adversary S and an ideal functionality $F_{\mathsf{voting}}(\rho)$ interact, returning the single output bit β produced by the environment, is written $\beta \leftarrow \mathsf{idealexec}(\mathfrak{E} \| S \| F_{\mathsf{voting}}(\rho))$.

5.2 The real protocol execution

In the real world, we have a voting scheme $\mathcal{V} = (\mathsf{GlobalSetup}, \mathsf{Setup}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Publish}, \mathsf{Tally}, \mathsf{Verify})$ played by a set of honest voters, an adversary A who can take control of voters, a honest administrator, and an honest ballot box. A global setup might be available to all these parties, e.g., under the form of a random oracle or a common random string, as provided by GlobalSetup. These three honest elements, i.e. the administrator, the ballot box and the global setup, can be seen as incorruptible ideal functionalities, and we say that a protocol running in the presence of these three functionalities runs in the ABG-hybrid model. If they are not readily available, these functionalities can be securely implemented using lower-level protocols, which are kept out of our model here. For instance, one would typically have multiple administrators running a threshold scheme in a real election.

As in the ideal world, there is an environment $\mathfrak{E}$ who commands the election: it asks the administrator to setup the election, provides voting instructions to the honest voters, asks to read the ballot box, and can also have arbitrary interactions with A.

An execution of the scheme $\mathcal{V}$ in interaction with environment $\mathfrak{E}$ and adversary $\mathcal{A}$ works as follows.

1. At first, and if there is a global setup, $\mathfrak{E}$ sends a `globalsetup.init` command in order to initialize the setup that is available to everyone.
2. $\mathfrak{E}$ sends a `setup` command to the administrator, who sends the election public key $\mathbf{pk}$ to everyone.
3. The ballot box creates an empty BB and can have two types of interaction:
   (a) It can receive a $\texttt{ballot}(b)$ command. In this case, it runs $\mathsf{Valid}(\mathsf{BB}, b)$ and, if successful, appends b to BB.
   (b) It can receive a `publish` command, on which returns $\mathsf{Publish}(\mathsf{BB})$ to the party that issued that command.
4. The following commands can be issued in arbitrary sequence.
   (a) $\mathfrak{E}$ can send vote instructions $\texttt{vote}(id, v)$ to honest voters, who compute a ballot $b = \mathsf{Vote}(id, v)$ and send $\texttt{ballot}(b)$ to the ballot box.
   (b) Possibly in coordination with $\mathfrak{E}$, the adversary A can submit arbitrary $\texttt{ballot}(b)$ commands to the ballot box on behalf of dishonest voters.
   (c) Anyone can send `publish` queries to BB.
5. At some point, the adversary sends `tally` to the ballot box. The content of BB is then sent to the administrator who, using the secret key $\mathbf{sk}$, runs Tally on the ballots and sends the result r and proof Π to the adversary.

The scheduling mechanism we use can be seen as token passing with the environment as master scheduler: $\mathfrak{E}$ is active first and, every time a command is issued to a party, this party gets the token. When a party halts, $\mathfrak{E}$ gets the token back, until it sends its output bit which halts the entire system.

The full process in which an environment $\mathfrak{E}$ and an adversary A run in an execution of a voting scheme $\mathcal{V}$ as described above, returning the single output bit β produced by $\mathfrak{E}$, is written $\beta \leftarrow \mathsf{realexec}(\mathfrak{E} \| A \| \mathcal{V})$.

5.3 Secure implementation

The following definition captures the idea that a secure protocol should not leak more information than whatever is leaked in the ideal world.

Definition 11. *We say that scheme $\mathcal{V}$ securely implements functionality $F_{\mathsf{voting}}(\rho)$ in the ABG-hybrid model if for any real adversary A there exists an ideal adversary S such that for any environment $\mathfrak{E}$ the distribution* $\mathsf{idealexec}(\mathfrak{E}||S||F_{\mathsf{voting}}(\rho))$ *is computationally indistinguishable from* $\mathsf{realexec}(\mathfrak{E}||A||\mathcal{V})$.

The following theorem establishes that correctness, ballot privacy, and strong consistency for the voting scheme are sufficient conditions to ensure that its associated voting protocol is secure in the sense defined above.

Theorem 1. *If a voting scheme $\mathcal{V}$ for result function ρ is strongly correct, strongly consistent, and ballot private, then the protocol for $\mathcal{V}$ securely implements $F_{\mathsf{voting}}(\rho)$ in the ABG-hybrid model.*

Proof. We prove the stronger statement that there is a simulator, with black-box access to the adversary, which works for any (adversary, environment) pair. We proceed by "game hopping".

Game 0 is described by $\mathsf{realexec}(\mathfrak{E}||A||\mathcal{V})$.

Game 1 is a modified version of Game 0:

- The board that the adversary sees contains ballots to some fixed vote $v^* \in \mathbb{V}$ instead of the honest voters' votes.
- The election result is obtained from the tally of a ballot box containing ballots for the real votes (which is hidden from the adversary).
- The auxiliary data are faked, using $\mathsf{SimProof}$ as defined from the BPRIV property.

The reader can immediately observe the close correspondence between the changes made to Game 0 and the guarantees offered by the BPRIV definition. Indeed, we show that Game 0 and Game 1 cannot be distinguished, unless the underlying voting scheme is not BPRIV secure.

We now give a more detailed description of Game 1 and sketch the proof for the above statement. The initialization step in Game 1 includes, besides the steps in the intialization of Game 0, the initialization of the fake board BB_1. In the case of a global setup, a simulated global setup $\mathsf{SimGlobalSetup}$ is produced and add added on BB_1: we formally add:

$$\mathsf{BB}_1 \leftarrow []; \mathsf{SimGlobalSetup} \leftarrow \mathsf{SimGlobalSetup}.\mathtt{init}$$

to the initialization step in Game 0, and the adversary gets access to $\mathsf{SimGlobalSetup}$.

Next, a `setup` command is issued to the administrator, and the adversary gets access to pk.

The adversary and the environment $\mathfrak{E}$ are allowed the same queries as in Game 0. These are answered as follows:

- on $\mathtt{vote}(id, v) \in (I \times \mathbb{V})$ from $\mathfrak{E}$:
  $b_0 \leftarrow \mathsf{Vote}(id, v)$; $b_1 \leftarrow \mathsf{Vote}(id, v^*)$; if $\mathsf{Valid}(\mathsf{BB}_1, b_1)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b_0$, $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b_1$.
- on $\mathtt{ballot}(b) \in (\{0,1\}^*)$ from A:
  if $\mathsf{Valid}(\mathsf{BB}_1, b)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b$ and $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b$.
- on `read` from A:
  return $\mathsf{Publish}(\mathsf{BB}_1)$ to the adversary.
- on `tally` from A:
  run $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{sk}, \mathsf{BB}_0)$, $\Pi_1 \leftarrow \mathsf{SimProof}(r, \mathsf{BB}_1)$; return (r, Π_1) to the adversary and halt.

We write $\mathsf{realexec}'(\mathfrak{E}||A||\mathcal{V})$ for the output of A in the above execution (which is also the output of Game 1).

We claim that for any adversary A and environment $\mathfrak{E}$, the distance between the outputs of the games Game 0 and Game 1, is at most the advantage of some adversary B against BPRIV security of $\mathcal{V}$.

Let D be a distinguisher (for the outputs of $\mathsf{realexec}$ and $\mathsf{realexec}'$). Adversary B against BPRIV uses D and operates as follows. B runs $\mathfrak{E}$ and A internally and answers their queries as follows.

- on $\mathtt{vote}(id, v) \in I \times \mathbb{V}$ from $\mathfrak{E}$: adversary B sets $v_0 \leftarrow v$ and $v_1 \leftarrow v^*$ and submits (id, v_0, v_1) to its own $\mathcal{O}\mathsf{voteLR}$ oracle.
- on $\mathtt{ballot}(b)$ from A: adversary B issues b to its own $\mathcal{O}\mathsf{cast}$ oracle.
- on `read` from A: adversary B queries $\mathcal{O}\mathsf{board}$ and forwards the result to A.
- on `tally` from A: adversary B queries $\mathcal{O}\mathsf{tally}$ and forwards the result to A.
- queries issued by A to his global setup are forwarded by B to his own global setup; the answers are sent to A.

When $\mathfrak{E}$ stops, B runs D on the local output of $\mathfrak{E}$ and outputs whatever D outputs. When B is in the BPRIV game with $\beta = 0$ the view that B provides to A and the $\mathfrak{E}$ is as in $\mathsf{realexec}(\mathfrak{E}||A||\mathcal{V})$. At the same time, if $\beta = 1$, then the view of A and $\mathfrak{E}$ is the one they have in $\mathsf{realexec}'(\mathfrak{E}||A||\mathcal{V})$. It follows that whenever D successfully distinguishes between the outputs of these two games, then B successfully distinguishes the corresponding experiments $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}}$ and $\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V}}$.

Game 2 We introduce the next game in order to establish a set of invariants that hold for the execution above; these will serve as stepping stone to identify and argue about the properties of the simulator S.

Recall that, in Game 1, ballot box BB_0 contains the real ballots submitted (by the adversary and the honest parties) and ballot box BB_1 contains only fake ballots. In this game we introduce a third box BB_2 which contains a list (id, v) of the actual votes cast by the users. The ballot box is initially empty and is updated when ballots are submitted. Specifically, we modify the way queries $\mathtt{vote}(id, v)$ and $\mathtt{ballot}(b)$ are processed in Game 1, as follows (to ease readability we underline the parts added in Game 2):

- on $\mathtt{vote}(id, v) \in (I \times \mathbb{V})$ from $\mathfrak{E}$:
  $b_0 \leftarrow \mathsf{Vote}(id, v)$; $b_1 \leftarrow \mathsf{Vote}(id, v^*)$;
  if $\mathsf{Valid}(\mathsf{BB}_1, b_1)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b_0$, $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b_1$, $\underline{\mathsf{BB}_2 \leftarrow \mathsf{BB}_2 \| (id, v) \text{ and return } (id, \mathtt{ack}) \text{ to the simulator}}$.
- on $\mathtt{ballot}(b) \in (I \times \{0,1\}^*)$:
  if $\mathsf{Valid}(\mathsf{BB}_1, b)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b, \mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b$; $\underline{(id, v) \leftarrow \mathsf{Extract}(\mathsf{sk}, b); \mathsf{BB}_2 \leftarrow \mathsf{BB}_2 \| (id, v).}$

As a result from these changes, every time a ballot is added to BB_0 and BB_1, a corresponding vote is added on BB_2: this is just the vote submitted in clear in the case of a $\mathtt{vote}$ query, or the extracted vote in the case of a $\mathtt{ballot}$ query, using the $\mathsf{Extract}$ algorithm resulting from the strong consistency guarantee. The view of the adversary is not modified in any way for the moment.

When A issues $\mathtt{tally}$, Game 2 checks that the content of BB_2 coincides with the votes extracted from BB_0[8]. If this is not the case the game aborts the execution, otherwise the result provided to the adversary is computed as in Game 1. Formally, the tally step in Game 2 is as follows:

- on $\mathtt{tally}$ from A:
  $\underline{\text{if } \mathsf{BB}_2 \neq \mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0) \text{ output } \perp \text{ and halt;}}$
  run $(r, \Pi) \leftarrow \mathsf{Tally}^{\mathsf{GlobalSetup}}(\mathsf{sk}, \mathsf{BB}_0)$;
  run $\Pi_1 \leftarrow \mathsf{SimProof}(r, \mathsf{BB}_1)$; return (r, Π_1) to the adversary.

Here, the view of the adversary can differ from Game 1 only if the test $\mathsf{BB}_2 \neq \mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0)$ succeeds. We claim that this can only happen with negligible probability, thanks to the properties of $\mathsf{Extract}$ guaranteed by strong consistency (Item 1. of Def. 8).

Indeed, we show how to build an adversary B that breaks strong consistency with a probability identical to the probability that $\mathsf{BB}_2 \neq \mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0)$ in Game 2.

B starts by emulating Game 2 internally, running by itself the roles of the administrator, adversary, environment, ... and inspects whether, in the end, $\mathsf{BB}_2 \neq \mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0)$. If it is the case, B extracts the query that makes those boards differ.

This query can certainly not be a $\mathtt{setup}$, $\mathtt{read}$ or $\mathtt{tally}$ query, since these queries do not modify any board. It can also not be a $\mathtt{ballot}$ query, since the vote that is added on BB_2 in that case is, by definition, the extraction of the ballot added on BB_0. Eventually, if it is a $\mathtt{vote}(id, v)$ query, then B found a situation where the extraction of $\mathsf{Vote}(id, v)$ (which appears on BB_0) differs from (id, v) (which appears on BB_2). But, from the first item of the definition of strong consistency, this can only happen with negligible probability, and so does the $\mathsf{BB}_2 \neq \mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0)$ property.

Game 3 This game is identical to the one above, with the difference that, during $\mathtt{vote}$ queries, we check whether the b_0 ballot is valid with respect to BB_0 and trigger an error othewise. Formally, we make the following changes (underlining the changes compared to Game 2, as before).

- on $\mathtt{vote}(id, v) \in (I \times \mathbb{V})$ from $\mathfrak{E}$:
  $b_0 \leftarrow \mathsf{Vote}(id, v)$; $b_1 \leftarrow \mathsf{Vote}(id, v^*)$;
  $\underline{\text{if not } \mathsf{Valid}(\mathsf{BB}_0, b_0) \text{ then output } \perp \text{ and halt.}}$
  if $\mathsf{Valid}(\mathsf{BB}_1, b_1)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b_0$, $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b_1$, $\mathsf{BB}_2 \leftarrow \mathsf{BB}_2 \| (id, v)$ and return $(id, \mathtt{ack})$ to the simulator.

[8] Applying $\mathsf{Extract}$ to a list of ballots means apply it component-wise.

A difference between Game 2 and Game 3 only appears when $\mathsf{Valid}(\mathsf{BB}_0, b_0)$ is false. However, this would immediately violate the strong correctness property (Def. 9.)

Indeed, an adversary against strong correctness can start emulating Game 3 internally, using a honestly generated public key provided as in the strong correctness experiment, up to a guess on which $\mathsf{vote}(id, v)$ query would result in falsifying $\mathsf{Valid}(\mathsf{BB}_0, b_0)$. Then, it can submit (id, v, BB_0) as output to the strong correctness experiment and, if the guess is correct (which happens with non negligible probability since there are at most a polynomial number of vote queries), it wins the experiment. But, since our voting scheme is strongly correct, this can only happen with negligible probability.

Game 4 This game is identical to the one above, with the difference that, during `vote` and `ballot` queries, we check whether ballots satisfy the ValidInd predicate (defined as part of strong consistency) after having tested ballots with Valid. Formally, we make the following changes to these queries:

- on $\mathtt{vote}(id, v) \in (I \times \mathbb{V})$ from $\mathfrak{E}$:
  $b_0 \leftarrow \mathsf{Vote}(id, v)$; $b_1 \leftarrow \mathsf{Vote}(id, v^*)$;
  if not $\mathsf{Valid}(\mathsf{BB}_0, b_0)$ then output $\perp$ and halt.
  if not $\mathsf{ValidInd}(b_0)$ then output $\perp$ and halt.
  if $\mathsf{Valid}(\mathsf{BB}_1, b_1)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b_0$, $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b_1$, $\mathsf{BB}_2 \leftarrow \mathsf{BB}_2 \| (id, v)$ and return $(id, \mathtt{ack})$ to the simulator.
- on $\mathtt{ballot}(b) \in (I \times \{0, 1\}^*)$: if $\mathsf{Valid}(\mathsf{BB}_1, b)$ then
  if not $\mathsf{ValidInd}(b)$ then output $\perp$ and halt, else
  $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b, \mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b$; $(id, v) \leftarrow \mathsf{Extract}(\mathsf{sk}, b); \mathsf{BB}_2 \leftarrow \mathsf{BB}_2 \| (id, v)$.

A difference between Game 3 and Game 4 can only appear if one of the ValidInd tests fails while the Valid test on the same ballot succeeds. However, such an event would lead to a contradiction of the second requirement of strong consistency, which requires that a ballot that is Valid for any board should also be ValidInd. An adversary could indeed emulate Game 4 internally and, as soon as a ValidInd test fails while the Valid test on the same ballot succeeds for a specific board, it would output that board and ballot, which would be in contradiction with Item 2. of the definition of strong consistency.

Game 5 This game is identical to the one above with the difference that, instead of running Tally on BB_0 in order to obtain the result r, the ρ function is evaluated on BB_2.

Formally, we modify the `tally` query as follows:

- on `tally` from A:
  if $\mathsf{BB}_2 \neq \mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0)$ output $\perp$ and halt;
  $r = \rho(\mathsf{BB}_2)$.
  run $\Pi_1 \leftarrow \mathsf{SimProof}(r, \mathsf{BB}_1)$; return (r, Π_1) to the adversary.

We claim that if $\mathcal{V}$ is strongly consistent (Definition 8) then the outputs of Game 4 and Game 5 are indistinguishable.

Consider a reduction B against Property 3. of strong consistency. Reduction B obtains pk, then simulates the execution in Game 5 up to the point where A issues `tally`. At this point B outputs BB_0 and halts. Since $\mathsf{BB}_2 = \mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0)$, the output distributions of Games 4 and 5 only differ if $R(\mathsf{Tally}(\mathsf{sk}, \mathsf{BB}_0)) \neq \rho(\mathsf{Extract}(\mathsf{sk}, \mathsf{BB}_0))$, where $R(r, \Pi) := r$ extracts the result from a tally. Furthermore, the tests added in Game 4 guarantee that BB_0 only contains ballots that satisfy ValidInd. So, if a difference happens, the reduction B wins against the strong consistency property of $\mathcal{V}$.

Game 6 This game is identical to the one above with the difference that we remove:

- The test on BB_2 added in Game 2;
- The Valid test added in Game 3;
- The $\mathsf{ValidInd}$ tests added in Game 4.

We proved in those games that these tests only make a difference with negligible probability, and we can use those same arguments to justify that a difference between Game 5 and Game 6 will only appear with negligible probability.

To sum up, the queries in Game 6 are treated as follows:

- on $\mathtt{vote}(id, v) \in (I \times \mathbb{V})$ from $\mathfrak{E}$:
  $b_0 \leftarrow \mathsf{Vote}(id, v)$; $b_1 \leftarrow \mathsf{Vote}(id, v^*)$
  if $\mathsf{Valid}(\mathsf{BB}_1, b_1)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b_0$, $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b_1$, $\mathsf{BB}_2 \leftarrow \mathsf{BB}_2 \| (id, v)$ and return $(id, \mathtt{ack})$ to the simulator.
- on $\mathtt{ballot}(b) \in (\{0,1\}^*)$ from A:
  if $\mathsf{Valid}(\mathsf{BB}_1, b)$ then $\mathsf{BB}_0 \leftarrow \mathsf{BB}_0 \| b$; $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b$; $(id, v) \leftarrow \mathsf{Extract}(\mathsf{sk}, b)$ and $\mathsf{BB}_2 \leftarrow \mathsf{BB}_2 \| (id, v)$.
- on $\mathtt{read}$ from A:
  return $\mathsf{Publish}(\mathsf{BB}_1)$ to the adversary.
- on $\mathtt{tally}$ from A:
  run $r \leftarrow \rho(\mathsf{BB}_2)$; run $\Pi_1 \leftarrow \mathsf{SimProof}(r, \mathsf{BB}_1)$; return (r, Π_1) to the adversary and halt.

Ideal adversary Finally, we construct a simulator S for which the output in $\mathsf{idealexec}(\mathfrak{E}||S||F_{\mathsf{voting}}(\rho))$ is distributed identically to the output of Game 6, thus completing the proof.

The simulator runs $\mathsf{SimGlobalSetup}.\mathtt{init}$ to obtain an environment $\mathsf{SimGlobalSetup}$ (if needed), runs Setup to obtain $(\mathsf{pk}, \mathsf{sk})$ and initializes a board BB_1. It then starts running A internally and answers queries as follows:

- When S receives $(id, \mathtt{ack})$ from $F_{\mathsf{voting}}(\rho)$, it produces $b \leftarrow \mathsf{Vote}(id, v^*)$. If $\mathsf{Valid}(\mathsf{BB}_1, b)$ returns true, then S executes $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b$.
- When A issues $\mathtt{ballot}(b)$, S runs $\mathsf{Valid}(\mathsf{BB}_1, b)$. If this returns false, then S does nothing. Otherwise, S runs $\mathsf{BB}_1 \leftarrow \mathsf{BB}_1 \| b$ and $v \leftarrow \mathsf{Extract}(\mathsf{sk}, b)$ and sends (id, v) to F_{voting}.
- When A issues query $\mathtt{read}$, S returns $\mathsf{Publish}(\mathsf{BB}_1)$.
- When A issues query $\mathtt{tally}$, S issues $\mathtt{tally}$ to F_{voting} and obtains r, calculates $\Pi_1 \leftarrow \mathsf{SimProof}(r, \mathsf{BB}_1)$ and returns (r, Π_1) to the adversary.

The claim is that the view of A and $\mathfrak{E}$ in the above game is identical to their view in Game 6. Indeed:

- The $\mathtt{read}$ queries from the adversary are answered from the board BB_1 in both cases;
- The $\mathtt{tally}$ query applies ρ to the sequence of votes received by the functionality, votes that precisely match those posted on BB_2 in Game 6.

6 Application to Helios

Helios [5] is a remote voting protocol aimed at providing both privacy and verifiability. Helios builds on Cramer *et al* [28] and Benaloh [8]. The attractiveness of Helios resides on its open-source nature, simplicity and that it consists of well-known cryptographic building blocks. Furthermore, it has been used several times to run binding elections, including the election of the president of the University of Louvain-La-Neuve and the election of the board directors of the International Association for Cryptographic Research (on a regular basis since 2010) [1] and of the ACM. As such, Helios is the ideal candidate to test any new cryptographic model for electronic voting. It is not surprising then that Helios has been shown to ensure ballot privacy in the past [35, 25, 11, 13]. However, as we have seen in the previous sections, those analysis use unsatisfactory vote privacy definitions or do not apply directly to Helios.

We show that our BPRIV and strong consistency definitions can be realized by Helios, which implies the first simulation-based vote privacy proof for Helios. Specifically, we analyze what could be considered the standard version of Helios nowadays, that uses strong Fiat-Shamir proofs [13], implements duplicate weeding [25] and homomorphic tallying. With respect to the threat model, we consider an honest single trustee and an honest bulletin board. Except for the single trustee, the other adversarial assumptions are similar to those used in previous ballot privacy analyses of Helios. Adapting our proof to a multi-authority scenario is expected to carry over in a similar manner as in [23], by extending BPRIV to a multi-trustee scenario.

More explicitly, the Helios version that we analyze uses the ElGamal [34] IND-CPA cryptosystem $\mathcal{D} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ in a given group $\mathbb{G}$ where the Decisional Diffie-Hellman assumption holds; the NIZK proof system [18, 27] $\mathsf{DisjProof}_H(g, \mathsf{pk}, R, S)$ to prove in zero-knowledge that (R, S) encrypts g^0 or g^1 (with proof builder $\mathsf{DisjProve}$ and proof verifier $\mathsf{DisjVerify}$); and the NIZK proof system [18] $\mathsf{EqDl}_G(g, R, \mathsf{vk}, c)$ to prove in zero-knowledge that $\log_g \mathsf{vk} = \log_R c$ for $g, R, \mathsf{vk}, c \in \mathbb{G}$ (with proof builder $\mathsf{ProveEq}$ and proof verifier $\mathsf{VerifyEq}$). H and G are hash functions mapping to $\mathbb{Z}_q$.

Helios allows both "referendum" style votes (a single yes/no question) and more complex ballots; in addition it allows for several revote policies. The result function in Helios is fixed to return the number of votes that each option received (since it tallies homomorphically). We formalise Helios for referendum style votes and the "only your last vote counts" revote policy.

Theorem 2. *Helios is BPRIV, strongly consistent, and strongly correct under the DDH assumption in the Random Oracle Model.*

Our BPRIV proof is largely inspired by [13]: we make use of the fact that $\mathsf{Vote}(id, v)$ produces non-malleable ciphertexts (when interpreted as a public key encryption algorithm) and that tallying proofs are in fact zero-knowledge proofs of decryption correctness. Details can be found in Appendix D.

7 Conclusion

We have reviewed the literature in order to find a game-based cryptographic definition of vote privacy suitable for remote voting protocols, in particular compatible with verifiability. We have identified shortcomings in several previous such definitions, and concluded that none of the existing definitions was satisfactory. Based on our findings, we have proposed a new definition, BPRIV, that avoids the existing limitations. In particular, BPRIV provides a more precise model of the tally function, with a revote policy and explicit auxiliary data.

We have additionally introduced the notions of strong consistency and strong correctness, and we have been able to show that together with BPRIV, they imply simulation-based privacy. More precisely, we showed that a single-pass protocol for computing some result function ρ, that is secure in the BPRIV game-based sense and strongly correct and consistent, achieves at least the same level of privacy as the ideal protocol for ρ. One immediate interpretation of this result is that from a scheme secure in the BPRIV sense the adversary can extract as much information as it can extract from only seeing the result and nothing more. This is a very desirable and intuitively appealing property as it reduces understanding the level of privacy of a protocol to that of understanding the level of privacy of a corresponding ideal protocol. Furthermore proofs using game-based definitions are more standard and easier to construct than those using the simulation-based notions. Whenever possible, proving game-based privacy is desirable.

We have shown that BPRIV can indeed be realized by a real-scale protocol, namely Helios [4]. Since Helios satisfies BPRIV and is strongly consistent, it immediately follows that Helios is secure for a simulation-based notion of privacy.

We can see several directions for future work. Like previous papers in the same area we model a single, trusted talling functionality (and assume that this could be implemented via a threshold scheme). It would be useful to spell out the required arguments and verify this assumption. A detailed treatment of mixnets (with more than one mixer) would also be interesting. Another extension could be to consider dishonest ballot boxes where, among other things, honest ballots could get "lost". On a similar note, the use of voter credentials to prevent ballot stuffing by the ballot box has gained some interest recently (there is also a Helios variant called Belenios that implements this [24]) and we think it would be interesting to model credentials in a privacy notion too.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 258865 (ProSecure) and grant agreement 609611 (PRACTICE), and under the 1317971 Walloon Region Greentic–Truedev project. This work has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO.

References

1. International Association for Cryptologic Research. Elections page at `http://www.iacr.org/elections/`.
2. Ballot paper for European Parliament election in Luxembourg, 2014. `http://upload.wikimedia.org/wikipedia/commons/b/bf/Ballot_paper_European_Parliament_elections_2014_in_Luxembourg.JPG`.
3. B. Adida. Helios: Web-based Open-Audit Voting. In *17th USENIX Security Symposium*, pages 335–348, 2008. Helios website: `http://heliosvoting.org`.
4. B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *Electronic Voting Technology Workshop/Workshop on Trustworthy Elections*. Usenix, Aug. 2009.
5. B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, 2009.
6. J. Benaloh. Verifiable secret-ballot elections. Technical Report 561, Yale University Department of Computer Science, September 1987.
7. J. Benaloh. Verifiable secret-ballot elections. PhD thesis, Yale University, 1987.
8. J. Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *Proceedings of the Second Usenix/ACCURATE Electronic Voting Technology Workshop*, 2007.
9. J. C. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In F. T. Leighton and M. T. Goodrich, editors, *STOC*, pages 544–553. ACM, 1994.
10. J. C. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters (extended abstract). In J. Y. Halpern, editor, *PODC*, pages 52–62. ACM, 1986.
11. D. Bernhard, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi. Adapting Helios for provable ballot secrecy. In Springer, editor, *16th European Symposium on Research in Computer Security (ESORICS'11)*, volume 6879 of *LNCS*, 2011.
12. D. Bernhard, V. Cortier, O. Pereira, and B. Warinschi. Measuring vote privacy, revisited. In *19th ACM Conference on Computer and Communications Security (CCS'12)*, Raleigh, USA, October 2012. ACM.
13. D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In X. Wang and K. Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.
14. D. Bernhard, O. Pereira, and B. Warinschi. On necessary and sufficient conditions for private ballot submission. Cryptology ePrint Archive, Report 2012/236, 2012. `http://eprint.iacr.org/`.
15. D. Bernhard and B. Smyth. Ballot privacy and ballot independence coincide. In Springer, editor, *Proceedings of the 18th European Symposium on Research in Computer Security (ESORICS'13)*, Lecture Notes in Computer Science, 2013.
16. M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn. Verifiable elections that scale for free. In K. Kurosawa and G. Hanaoka, editors, *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 479–496. Springer, 2013.
17. D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora. Scantegrity: End-to-End Voter-Verifiable Optical-Scan Voting. *IEEE Security and Privacy*, 6(3):40–46, 2008.
18. D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 1992.
19. M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a Secure Voting System. In *29th Security and Privacy Symposium (S&P'08)*. IEEE, 2008.
20. M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *Proc. IEEE Symposium on Security and Privacy*, pages 354–368, 2008.
21. J. Cohen (Benaloh) and M. Fischer. A robust and verifiable cryptographically secure election scheme. In *26th Symposium on Foundations of Computer Science.*, pages 372–382, Portland, OR, 1985. IEEE.
22. L. Coney, J. L. Hall, P. L. Vora, and D. Wagner. Towards a privacy measurement criterion for voting systems. In *In National Conference on Digital Government Research*, 2005.
23. V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. Distributed ElGamal à la Pedersen: Application to helios. In A.-R. Sadeghi and S. Foresti, editors, *WPES*, pages 131–142. ACM, 2013.
24. V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. Election verifiability for Helios under weaker trust assumptions. In M. Kutylowski and J. Vaidya, editors, *Computer Security - ESORICS 2014 Proceedings, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2014.

25. V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *24th IEEE Computer Security Foundations Symposium (CSF'11)*, pages 297–311. IEEE, 2011.
26. V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.
27. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.
28. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology (EUROCRYPT'97)*, page 103118, 1997.
29. E. Cuvelier, O. Pereira, and T. Peters. Election verifiability or ballot privacy: Do we need to choose? In Springer, editor, *Proceedings of the 18th European Symposium on Research in Computer Security (ESORICS'13)*, Lecture Notes in Computer Science, 2013.
30. O. de Marneffe, O. Pereira, and J.-J. Quisquater. Simulation-based analysis of e2e voting systems. In A. Alkasar and M. Volkamer, editors, *Proceedings of the First Conference on E-Voting and Identity (VOTE-ID 2007)*, number 4896 in LNCS, pages 137–149. Springer, Oct. 2007.
31. S. Delaune, S. Kremer, and M. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
32. S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
33. J. Dreier, P. Lafourcade, and Y. Lakhnech. Defining privacy for weighted votes, single and multi-voter coercion. In S. Foresti, M. Yung, and F. Martinelli, editors, *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, pages 451–468. Springer, 2012.
34. T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
35. J. Groth. Evaluating security of voting schemes in the universal composability framework. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS*, volume 3089 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2004.
36. A. Juels, D. Catalano, and M. Jakobsson. Coercion-Resistant Electronic Elections. In *4th Workshop on Privacy in the Electronic Society (WPES 2005)*, pages 61–70. ACM, 2005.
37. R. Küsters, T. Truderung, and A. Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 122–136. IEEE, 2010.
38. R. Küsters, T. Truderung, and A. Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *IEEE Symposium on Security and Privacy (S&P 2011)*, pages 538–553. IEEE Computer Society, 2011.
39. T. Moran and M. Naor. Split-ballot voting: everlasting privacy with distributed trust. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, (CCS 2007)*, pages 246–255, Alexandria, Virginia, USA, 2007.
40. R. L. Rivest and W. D. Smith. ThreeVotingProtocols: ThreeBallot, VAV, and Twin. In *Electronic Voting Technology Workshop (EVT 2007)*, 2007.
41. P. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. The prêt à voter verifiable election system. *IEEE Transactions on Information Forensics and Security*, 4:662–673, 2009.
42. D. Wikström. Simplified submission of inputs to protocols. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *Security and Cryptography for Networks, 6th International Conference, SCN 2008*, volume 5229 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2008.

$$\begin{array}{lcl}
\text{If } \beta = 0 & \overset{\mathcal{B}\text{ sees}}{\Longrightarrow} & \{b_0\},\{b_c\},\ \{(\,b_0, v_{b_0}\,)\}\,,\ \{v_{b_1}\}\,,\ \{(\,b_c, v_{b_c}\,)\}\,, \Pi' \\
\text{If } \beta = 1 & \overset{\mathcal{B}\text{ sees}}{\Longrightarrow} & \{b_1\},\{b_c\},\ \{(\,b_1, v_{b_0}\,)\}\,,\ \{v_{b_1}\}\,,\ \{(\,b_c, v_{b_c}\,)\}\,, \Pi^{\dagger} \\
\hline
\text{If } \beta = 0 & \overset{\mathcal{B}'\text{ sees}}{\Longrightarrow} & \{b_0\},\{b_c\},\ \{v_{b_0}\}\,,\ \{v_{b_1}\}\,,\ \{v_{b_c}\}\,, \Pi' \\
\text{If } \beta = 0 & \overset{\mathcal{B}'\text{ sees}}{\Longrightarrow} & \{b_1\},\{b_c\},\ \{v_{b_0}\}\,,\ \{v_{b_1}\}\,,\ \{v_{b_c}\}\,, \Pi^{\dagger}
\end{array}$$

Fig. 3. Comparison between BPRIV adversaries.

A Test cases for BPRIV definitions

During our work on this paper, we listed some insecure variations on voting protocols. Future designers of ballot privacy-related notions can use these as "safety checks": if any of these schemes is not *insecure* under some notion of privacy then the notion may be too weak. We argue that none of these insecure schemes can satisfy BPRIV.

A.1 Helios v3 — Replay attacks

Any voting scheme in which an adversary can read a honest voter's ballot from the ballot box and resubmit a ballot for the same vote as her own fails to ensure vote privacy, as argued in Section 3.6 and illustrated on Helios (version 3) [26].

More precisely, consider an adversary who, for any two distinct honest votes v_0 and v_1, can perform the following sequence of queries:

$$\begin{array}{l}\texttt{start-voting(); vote(}v_0,v_1\texttt{); vote(}v_1,v_0\texttt{);}\\ bb \leftarrow \texttt{board(); } b \leftarrow \texttt{ first(}bb\texttt{); ballot(}b\texttt{);}\\ (r,a) \leftarrow \texttt{ tally()}\end{array}$$

Here `first` returns the first ballot of *bb*. We may also imagine more sophisticated attacks where the adversary has to change e.g. identifying information on the ballot. We deliberately choose two vote queries that would produce the same "left" and "right" tallies, if b was independent of the honest ballots,

In the BPRIV game, if the ballot b is not rejected (by Publish) then the tally for $\beta = 0$ is $\rho(v_0, v_1, v_0)$ and for $\beta = 1$ it is $\rho(v_1, v_0, v_1)$. If these are not the same value, which happens for example if ρ counts the number of v_0 and v_1 votes submitted, then our adversary can win the BPRIV game (with probability 1), hence the broken scheme is *not* BPRIV secure.

A.2 Leaky auxiliary data

Consider any voting scheme where the result function ρ is supposed to output how many times each choice was voted for, i.e. one could write $\rho(v_0, v_1, v_0, v_0) = \{(v_0, 3), (v_1, 1)\}$. Imagine that, unfortunately, a scheme's tally also contains all votes in clear, in the order that they were cast. This is clearly not desirable. Intuitively, there are two distinct ways of discarding such a scheme. Either the (leaky) output of the tally is classified as being the result itself. Then the scheme does not implement the desired functionality and this is captured with our "strong consistency" notion. Or this leaky output is contained in the auxiliary data. In that case, we can construct a BPRIV adversary against such a scheme:

$$\begin{array}{l}\texttt{start-voting(); vote(}v_0,v_1\texttt{); vote(}v_1,v_0\texttt{);}\\ (r,a) \leftarrow \texttt{ tally()}\end{array}$$

To show that this adversary can win, we recall that the output of tally is split into a result r and auxiliary data $\mathit{\Pi}$ in such a way that r is guaranteed not to leak any information. In other words, r is the correct/intended result so the list of all votes must be part of $\mathit{\Pi}$. In the case $\beta = 0$, the adversary sees auxiliary data that reveals all the votes on BB_0, i.e. the sequence (v_0, v_1). In contrast, if $\beta = 1$ then $\mathit{\Pi}$ is computed as a (probabilistic) function of BB_1 and r, neither of which can contain any information about the order of the votes in BB_0. So the probability of $\mathit{\Pi}'$ containing the list (v_0, v_1) is at most the probability of guessing the adversary's "left" votes without access to any ballots. Therefore, the adversary can distinguish $\beta = 0$ from $\beta = 1$ with high probability.

B BPRIV2 does not guarantee ballot privacy

Let us prove formally that $\mathcal{V}$ defined in Section 3.4 is BPRIV2 secure, while it should intuitively not be declared private.

$$\begin{array}{l}\mathsf{SimProof}(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \texttt{info}) := \mathit{\Pi}^\dagger \,||\\ \{(\,b, v_b\,)\}_{b \in \mathsf{BB}_0 \cap \mathsf{BB}_1} \,||\, \{(\,b_1, v_{b_0}\,)\}_{b_1 \in \overline{\mathsf{BB}}_1, b_0 \in \overline{\mathsf{BB}}_0}\end{array}$$

where $\overline{\mathsf{BB}}_\gamma := \mathsf{BB}_\gamma \setminus (\mathsf{BB}_0 \cap \mathsf{BB}_1)$, $v_b := \mathsf{Extract}(\mathbf{sk}, b)$ and $\mathit{\Pi}^\dagger := \mathsf{SimProof}'(\mathsf{BB}_0, \mathsf{BB}_1, \mathbf{pk}, \texttt{info})$. Intuitively, $\mathsf{SimProof}$ decrypts correctly the adversarial ballots but mimics the case $\beta = 0$ for all honest ballots. Then it turns out that if $\mathcal{V}'$ is BPRIV2 private then so is $\mathcal{V}$. Indeed, let us write $\mathsf{BB}_\beta = H_\beta \cup C$, where H_β is the sublist of honest ballots output by the $\mathcal{O}\mathsf{voteLR}$ oracle for $\beta = 0, 1$, and $C := \mathsf{BB}_\beta \setminus H_\beta$. By construction $C \subseteq \mathsf{BB}_0 \cap \mathsf{BB}_1$. We want to see that for every BPRIV2 adversary $\mathcal{B}$ against $\mathcal{V}$ there exists a BPRIV2 adversary $\mathcal{B}'$ against scheme $\mathcal{V}'$

such that $\mathsf{Succ}^{\mathsf{bpriv2}}(\mathcal{B}) \approx_{\mathsf{negl}} \mathsf{Succ}^{\mathsf{bpriv2}}(\mathcal{B}')$, their advantages only differ in a negligible quantity. We compare the data seen by BPRIV2 adversaries $\mathcal{B}$ against scheme $\mathcal{V}$ and adversaries $\mathcal{B}'$ against scheme $\mathcal{V}'$ in Figure 3, wherein $b_0 \in H_0, b_1 \in H_1, b_c \in C$. We observe that the only extra information that adversary $\mathcal{B}$ against $\mathcal{V}$ sees compared to adversary $\mathcal{B}'$ against $\mathcal{V}'$, are the relations $\{(\, b_0, v_{b_0}\,)\}$ and $\{(\, b_c, v_{b_c}\,)\}$ when $\beta = 0$, and the relations $\{(\, b_1, v_{b_0}\,)\}$ and $\{(\, b_c, v_{b_c}\,)\}$ when $\beta = 1$. However this does not allow $\mathcal{B}$ to have a greater distinguishing advantage than $\mathcal{B}'$. Indeed, since the new matching between ballots and votes is not verifiable (the original tallying proofs $\Pi', \Pi^{\dagger}$ did not change and $\mathcal{V}'$ is BPRIV2 private), adversary $\mathcal{B}'$ can simulate $\mathcal{B}$'s view using its own view. It suffices for $\mathcal{B}'$ to tell apart the multisets of votes V_0 and V_C (this can be done easily, since V_0 are the left vote queries of $\mathcal{B}$ and $V_C := r \setminus V_0$), and later link H_0 or H_1 to V_0 at its liking. Finally, $\mathsf{Succ}^{\mathsf{bpriv2}}(\mathcal{B}) \approx_{\mathsf{negl}} \mathsf{Succ}^{\mathsf{bpriv2}}(\mathcal{B}')$.

C Ballot privacy in the random oracle model

In this section we formalise ballot privacy in the random oracle model (ROM). This is the model used for the Helios [3] voting protocol.

C.1 The real world

The random oracle model models a scenario where all algorithms may call a hash function $H : D \to C$ for fixed domain D and codomain C. Usually, we have $D = \{0,1\}^*$ and $C = \{0,1\}^n$ for some fixed value of n e.g. 256. Ideally, a hash function would mimic a random function — for a uniformly sampleable codomain C, this means that for any $d \in D$ the value $H(d)$ is uniform in C and for any two distinct $d, d' \in D$ the values $H(d), H(d')$ are independent. It does not matter if D is not uniformly sampleable (that is, countably infinite).

In Helios, a hash function (SHA-256) is used to compute the proofs of correct tallying which form the auxiliary data.

C.2 The model

In the random oracle model, there is an execution environment that simulates a random function $D \to C$ which all algorithms may call as an "oracle" $\mathcal{O}$. It starts out with an empty table T. Whenever an algorithm calls $\mathcal{O}(d)$ for $d \in D$, the environment checks if there is an entry (d, c) in T; if so it returns c. If not, it picks $c' \in C$ uniformly at random, adds (d, c') to T and returns c'. This way, the environment can efficiently simulate a truly random function.

In addition, a random oracle model environment may provide one or more simulation algorithms. These are just regular efficient algorithms but because they are part of the execution environment, they are allowed to add extra entries (d, c) to the table T. This is known as the ability to "program the oracle".

One might ask whether one does not need a restriction on simulation algorithms to prevent them from doing something "non-random" like mapping many values in D to the same C or adding an entry (d, c) when there is already a (d, c') with $c \neq c'$ in T. The answer is no. A random oracle model definition usually proceeds along the lines of "a scheme is secure (in some sense) if no adversary can distinguish this real algorithm (that may call $\mathcal{O}$) from the simulation algorithm S". In this style of definition, if a simulation algorithm does anything stupid, an adversary will be able to distinguish it from a real algorithm. Therefore such a simulation algorithm does not prove anything.

Typical simulation algorithms only need to program the oracle at points which they frashly choose with high entropy. In fact, a common simulation strategy is to choose a $c \in C$ uniformly at random, somehow compute a d-value from this and then add (d, c) to the list T. The value d will have as much entropy as c so the chance of it already appearing in T is proportional to $1/|C|$ with a constant that represents an upper bound on the number of times anyone has called $\mathcal{O}$ so far. For $|C| = 2^{256}$ or (the more modern choice) $|C| = 2^{512}$ a collision in D is highly unlikely.

C.3 BPRIV in the ROM

The ROM definition of BPRIV is simply the definition of BPRIV w.r.t. the random oracle model described above. Thus, all algorithms that are part of the voting scheme in question and the adversary who plays the BPRIV game may all call the oracle $\mathcal{O}$ as a subroutine. This oracle is part of the execution model, not the game. During tallying for $\beta = 1$, the game calls the execution model's $\mathsf{SimProof}$ algorithm which may program the oracle. To show that a particular voting scheme $\mathcal{V}$ is secure therefore requires two steps:

- Provide an implementation of $\mathsf{SimProof}$ that may access a "global variable" T holding a list of pairs in $D \times C$.
- Prove that the two games are indistinguishable w.r.t to the simulator provided above. This will of course only work for a "good" simulator that does not do anything unexpected to T.

D Helios has vote privacy

In this section we prove that Helios is BPRIV, strongly consistent and correct, and thus, by virtue of Theorem 1, Helios has vote privacy.

D.1 Description of Helios

Helios uses the ElGamal [34] IND-CPA cryptosystem $\mathcal{D} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ in a given group $\mathbb{G}$ where the Decisional Diffie-Hellman assumption holds; the NIZK proof system [18, 27] $\mathsf{DisjProof}_H(g, \mathsf{pk}, R, S)$ to prove in zero-knowledge that (R, S) encrypts g^0 or g^1 (with proof builder $\mathsf{DisjProve}$ and proof verifier $\mathsf{DisjVerify}$); and the NIZK proof system [18] $\mathsf{EqDl}_G(g, R, \mathsf{vk}, c)$ to prove in zero-knowledge that $\log_g \mathsf{vk} = \log_R c$ for $g, R, \mathsf{vk}, c \in \mathbb{G}$ (with proof builder $\mathsf{ProveEq}$ and proof verifier $\mathsf{VerifyEq}$). H and G are hash functions mapping to $\mathbb{Z}_q$.

Helios allows both "referendum" style votes (a single yes/no question) and more complex ballots; in addition it allows for several revote policies. The result function in Helios is fixed to return the number of votes that each option received (since it tallies homomorphically). We formalise Helios for referendum style votes and the "only your last vote counts" revote policy below as a set of six algorithms

$$\mathcal{V}^{\mathsf{helios}} = (\mathsf{Setup}, \mathsf{Vote}, \mathsf{Valid}, \mathsf{Publish}, \mathsf{ValidBoard}, \mathsf{Tally}, \mathsf{Verify})$$

and give the proof for this case and explain the extension to the general case at the end of this section.

$\mathsf{Setup}(1^\lambda)$ chooses $\mathbb{G}$ a cyclic group of order q and $g \in \mathbb{G}$ a generator. It randomly chooses $\mathsf{sk} \xleftarrow{R} \mathbb{Z}_q$ and sets $\mathsf{pk} = g^{\mathsf{sk}}$. Hash functions $G, H : \{0,1\}^\star \to \mathbb{Z}_q$ are chosen. It outputs $\mathbf{pk} \leftarrow (\mathbb{G}, q, \mathsf{pk}, L, G, H, \mathbb{V} = \{0,1\})$, the public key of the election and $\mathbf{sk} = (\mathbf{pk}, \mathsf{sk})$.

$\mathsf{Vote}\,(id, v)$ is used by a voter with identifier id to create a ballot b corresponding to vote v as follows: encrypts $v \in \{0,1\}$ as $C = \mathsf{Enc}(\mathsf{pk}, g^v) = (R, S)$. Computes a proof $\pi = \mathsf{DisjProve}_H(g, \mathsf{pk}, R, S, r)$ showing that the encrypted vote is 0 or 1. The ballot is defined as $b = (id, (C, \pi))$. The voter submits the ballot b by authenticating itself to the ballot box.

$\mathsf{Valid}(\mathsf{BB}, b)$ first checks that the ballot is *valid*, that is, that all proofs are correct. Formally, it parses the ballot b as $(id,\ (C, \pi))$. It then checks whether: (1) $id \in I$; (2) $\mathsf{DisjVerify}_H(g, \mathsf{pk}, C, \pi) = 1$. If any step fails, it returns $\perp$. Next, b is rejected if C appears previously in BB. Otherwise, returns $\top$.

$\mathsf{Publish}(\mathsf{BB})$ updates BB by applying the *only the last vote counts* revote policy. I.e., for any id appearing in an entry of BB only keeps the entry corresponding to the last such appearance of id. Next, removes the identifier id of every entry in BB and publishes the result, called PBB.

$\mathsf{Tally}(\mathsf{BB}, \mathbf{sk})$ consists of the following steps:

(1) Parses each ballot $b \in \mathsf{BB}$ as $(id_b, (C_b, \pi_b))$.
(1) If there exist id_b and $id_{b'}$ for $b, b' \in \mathsf{BB}$ such that $id_b \neq id_{b'}$ and $C_b = C_{b'}$, output $\perp$.

(2) Updates BB with the *only the last vote counts* revote policy.
(3) Computes the result ciphertext $C_\Sigma = (R_\Sigma, S_\Sigma) = \big(\prod_{b\in\mathsf{BB}} R_b, \prod_{b\in\mathsf{BB}} S_b\big)$, where $C_b = (R_b, S_b)$. This of course relies on the homomorphic property of the El Gamal encryption scheme.
(4) Computes $g^r \leftarrow S_\Sigma \cdot (R_\Sigma)^{-\mathsf{sk}}$. Then r to be published is obtained from g^r in time $\sqrt{\tau}$ for r lying in the interval $[0, \tau]$ and τ equals the number of legitimate voters.
(5) Finally $\Pi := \mathsf{ProveEq}_G\big(g, \mathsf{pk}, R_\Sigma, S_\Sigma \cdot (g^r)^{-1}, \mathsf{sk}\big)$.

$\mathsf{Verify}(\mathsf{PBB}, r, \Pi)$ consists of the following steps:

(1) Checks that every pair (C, π) in PBB is consistent (i.e. the corresponding NIZK proof verifies) and that a single C does not appear twice in PBB. If any of these checks fails, then returns 0 unless the result is itself $\perp$, in which case outputs 1.
(2) Computes the result ciphertext $(R_\Sigma, S_\Sigma) = \big(\prod_{b\in\mathsf{BB}} R_b, \prod_{b\in\mathsf{BB}} S_b\big)$.
(3) Returns $\mathsf{VerifyEq}_G\big(g, \mathsf{pk}, R_\Sigma, S_\Sigma \cdot (g^r)^{-1}, \Pi\big)$ as output.

D.2 Helios is BPRIV

We define a sequence of games, starting with the adversary interacting with the BPRIV challenger with $\beta = 0$, and ending up with the adversary interacting with the BPRIV challenger with $\beta = 1$. Each transition will be noticed by the adversary with only a certain negligible probability. As a result, we will obtain a bound on the BPRIV distinguishing advantage of any adversary by a negligible quantity, effectively showing that Helios is BPRIV private.

Before stating the sequence of games, we make several observations. At any point, since the visible bulletin board is built through a succession of $\mathcal{O}\mathsf{voteLR}(id, v_0, v_1)$ and $\mathcal{O}\mathsf{cast}(id, b)$ queries, our BPRIV reduction can associate, to any entry (id_i, b_i) in the visible bulletin board, a tuple $(id_i, b_i^0, b_i^1, v_i^0, v_i^1)$, where: b_i^0, b_i^1 are the ballots seen by the adversary when $\beta = 0, 1$ respectively; v_i^0, v_i^1 are the votes contained in the corresponding ballots b_i^0, b_i^1, whenever known. Typically, if the i-th entry in the bulletin board has been built through a successful $\mathcal{O}\mathsf{cast}(id, b)$ query, then $b_i^0 = b_i^1$ and $v_i^0 = v_i^1 = \emptyset$. If (id_i, b_i) is the i-th entry in the visible board, then $b_i^\beta = b_i$.

For the sake of completeness, let us recall that Helios' voting algorithm makes use of $\mathsf{DisjProof}_H(g, \mathsf{pk}, R, S)$, which is the NIZK system associated to the language $\mathcal{L}_{\mathsf{EqDl}} = \{(g_1, g_2, y_1, y_2) \mid \log_{g_1} y_1 = \log_{g_2} y_2\}$, where $g_1, g_2 \in \mathbb{G}$. The system $\mathsf{DisjProof}_H$ is obtained via the Fiat-Shamir transform applied to the the Chaum-Pedersen Σ-protocol, where H is the random oracle. It works as follows: prover and verifier have as input $(\mathbb{G}, q, (g_1, y_1), (g_2, y_2))$; prover additionally has a witness $x = \log_{g_1} y_1 = \log_{g_2} y_2$ to the statement as additional input. The prover chooses $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and sends $com_1 = g_1^r$ and $com_2 = g_2^r$ to the verifier. The latter sends a random challenge $ch \overset{R}{\leftarrow} \mathbb{Z}_q$ to the prover who then responds with $res = r + x \cdot ch$. The verifier accepts iff $g_1^{res} = com_1 \cdot y_1^{ch}$ and $g_2^{res} = com_2 \cdot y_2^{ch}$. For this Σ-protocol, $\mathsf{Simulate}_\Sigma(g_1, g_2, y_1, y_2, ch, res)$ returns $com_1 \leftarrow g_1^{res}/y_1^{ch}$ and $com_2 \leftarrow g_2^{res}/y_2^{ch}$.

Helios is always relative to a result function $\rho : (I \times \mathbb{V})^* \to R$, which must enjoy certain homomorphic properties if tallying is to be computed homomorphically. In any case, Helios' tallying can always (at least, from a functional point of view) be computed by decrypting every surviving individual ballot as $r = \rho\left((id_1, v_1), \ldots, (id_n, v_n)\right)$.

Let us recall the indistinguishability-based definition of non-malleability (NM-CPA) for public key encryption schemes, by means of the following game:

1. The challenger runs $(pk, sk) \leftarrow \mathsf{KeyGen}(\lambda)$ and gives pk to the adversary.
2. The adversary picks two messages m_0, m_1 and hands them to the challenger who picks a bit $\beta \leftarrow \{0, 1\}$ and returns an encryption c^* of m_β.
3. The adversary may submit a vector $\boldsymbol{c} = (c_i)_i$ of ciphertexts. For each c_i, if $c_i = c^*$ (the challenge ciphertext) then the challenger returns $\perp$, otherwise he returns $\mathsf{Dec}(sk, c_i)$.
4. The adversary may submit a guess β' for β.

The adversary wins if $\beta' = \beta$ and his advantage is defined as $|\Pr[\beta' = \beta] - 1/2|$.

[Game $\mathbf{G}_{-1}$] Let $\mathbf{G}_{-1}$ be the BPRIV game corresponding to Experiment $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}^{\mathsf{Helios}}}$ (Definition 7 and Figure 2 for $\beta = 0$). In particular, the BPRIV adversary $\mathcal{A}$ sees the ballot box BB_0 and an oracle $\mathcal{O}\mathsf{tally}()$ faithfully answered.

[Game $\mathbf{G}_0$] Let $\mathbf{G}_0$ be the game obtained by introducing the following change in Game $\mathbf{G}_{-1}$. In the latter game, the BPRIV's adversary is expecting to see the output $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{PBB}_0, \mathbf{sk})$. The change consists on simulating the tallying proof $\Pi' \leftarrow \mathsf{Simulate}^{G}_{\mathsf{EqDl}}(g, R_{\Sigma}, y, S_{\Sigma} \cdot g^{-r})$ by programming the Random Oracle G. Thanks to the zero-knowledge property of the NIZK proof system EqDl, the distinguish probability of the BPRIV adversary in Game $\mathbf{G}_0$ is negligibly close to that in Game $\mathbf{G}_{-1}$. From now on, the tallying proof is always *simulated.*

Let n be the number of entries in the ballot box BB_0.

Next, for $i = 1$ to n, we define a sequence of games, in which we will be changing the contents of BB_0 from Game $\mathbf{G}_{0,i-1}$ to Game $\mathbf{G}_{0,i}$. Let us call $\mathsf{BB}_{0,i}$ the contents of BB_0 at Game $\mathbf{G}_{0,i}$.

[Game $\mathbf{G}_{0,i}$] It is obtained from Game $\mathbf{G}_{0,i-1}$ by taking one of two possible actions, depending on the contents of the tuple $(id_i, b_i^0, b_i^1, v_i^0, v_i^1)$ that the security reduction keeps internally:

- if $b_i^0 = b_i^1$; do nothing
- if $b_i^0 \neq b_i^1$; replace the i-th entry (id_i, b_i^0) in BB_0 with (id_i, b_i^1)

We argue that, thanks to the NM-CPA property [13] of the ElGamal scheme plus NIZK proof system EqDl, the distinguishing probability of the BPRIV adversary in Game $\mathbf{G}_{0,i-1}$ is negligibly close to that in Game $\mathbf{G}_{0,i}$ (for he case where $\mathbf{G}_{0,i-1} \neq \mathbf{G}_{0,i}$, as otherwise the distinguishing probabilities do not change). To see this, consider a NM-CPA adversary $\mathcal{B}_i$ that makes use of the distinguish advantage of a BPRIV adversary between Games $\mathbf{G}_{0,i-1}$ and $\mathbf{G}_{0,i}$. NM-CPA adversary $\mathcal{B}_i$ simulates the environment for $\mathcal{A}$:

- uses programming of the random oracle G to simulate the tallying proof Π'
- excepts for the i-th entry in $\mathsf{BB}_{0,i}$, answers a query $\mathcal{O}\mathsf{vote}(id_j, v_j^0, v_j^1)$ as follows:
  - if $j < i$, then computes $\mathsf{Vote}(id_j, v_j^1)$
  - if $j > i$, then computes $\mathsf{Vote}(id_j, v_j^0)$
- sets the i-th entry in BB_0 to be the answer of the NM-CPA challenger on query v_i^0, v_i^1
- makes a NM-CPA decryption call $(b_1^L, \ldots, b_{i-1}^L, b_{i+1}^L, \ldots, b_n^L)$ and lets $(v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$ be the output of this call
- defines

$$r = \rho\left((id_1, v_1), \ldots, (id_{i-1}, v_{i-1}), (id_i, v_i^0),\right.\\ \left.(id_{i+1}, v_{i+1}), \ldots, (id_n, v_n)\right)$$

  where ρ is the addition result function modulo the revote policy.

[Game $\mathbf{G}_1$] Let us rename Game $\mathbf{G}_{0,n}$ as Game $\mathbf{G}_1$. Then, the view of the adversary in Game $\mathbf{G}_1$ corresponds to the view of the BPRIV adversary when $\beta = 1$. Thus, thanks to the transition of games previously depicted, we conclude that $\left| \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V}^{\mathsf{helios}}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V}^{\mathsf{helios}}}(\lambda) = 1\right] \right|$ is negligible in λ for any BPRIV adversary $\mathcal{A}$. Helios is thus BPRIV private.

D.3 Generalisations

Helios allows for three kinds of ballots. The referendum case we gave above. The case of approval voting (each voter can vote yes/no for a number of candidates or options) is handled in Helios by having each voter submit a vector of ciphertext/proof pairs, one for each option. Duplicate ballots are removed as before, each option is then tallied separately. In the hybrid argument in our proof, for n voters each submitting a k-vector of ciphertexts there are $k \cdot n$ ElGamal encryptions so we do a hybrid argument with $k \cdot n$ games. The rest of the proof is as before; in particular all k options are tallied "in parallel" with a single decryption query in the NM-CPA reduction. Finally, Helios allows for complex ballots in which each voter may pick e.g. up to m out of k candidates. Helios handles this case like approval voting except that each voter makes an additional "overall" proof that they have chosen at most m candidates in their k-vector of ciphertexts (and a ballot without a valid overall proof is weeded out during tallying). This does not affect our security proof.

Helios also allows for several revote policies, in particular (a) only your last vote counts and (b) you may only vote once, equivalently only your first vote counts. One could easily extend Helios to handle any revote policy that can be determined by looking at the ids of the submitted ballots alone. Our proof is unaffected by this generalisation. All our games simply delegate to a given result function for the revoting policy. For the same reason, our proof would also apply to variants of Helios that used a different result function, for example `majority` instead of `counting`. In conclusion, the special case that we have given in our proof is actually sufficient to cover the general case for Helios.

D.4 Helios is strongly consistent and strongly correct

To argue that Helios is strongly consistent we define the following extraction and ballot validation algorithms:

1. $\mathsf{Extract}((C, \pi), \mathsf{sk})$ first checks the proof π and return $\perp$ if this fails. Then decrypts C with the secret key sk and returns the result.
2. $\mathsf{ValidInd}(b)$ checks that the ballot is *valid*, that is, that all proofs are correct. Formally, it parses the ballot b as $(id,\ (C, \pi))$. It then checks whether: (1) $id \in I$; (2) $\mathsf{DisjVerify}_H(g, \mathsf{pk}, C, \pi) = 1$. If any step fails, returns $\perp$; otherwise returns $\top$.

The first condition of strong consistency (cf. Section 4.2) is that honestly created ballots extract correctly. This is just the correctness property of the building blocks: correctly created ZK proofs also verify and correctness of encryption says that decrypting a ciphertext returns the encrypted message. The same argument shows that Helios is strongly correct.

The second condition is trivially satisfied in this case.

The third condition says that no adversary can produce a ballot box on which the tally algorithm succeeds but the result is incorrect (w.r.t. extraction). The tally algorithm of Helios fails if there are any invalid or duplicate ballots on the ballot box, so for strong consistency it suffices to consider ballot boxes where all ballots are valid and unique. In this case, the extractor decrypts each ballot individually and then computes the result using the result function ρ (which may also weed out some votes). The talliers by contrast apply the revote policy over the ballots first, then homomorphically add all ballots and decrypt the result. For the "last vote counts" revote policy, it is clear that Tally and ρ remove the same ballots since this policy depends only on the identities and not the ciphertexts in the ballots. The same argument could be made for any revote policy that only considers identities. Finally, the homomorphic property of (ElGamal) encryption guarantees that adding ballots then decrypting (as in Tally) produces the same result as extracting then applying the `counting` result function. Finally, since the ballot box is honestly built (namely, all ballots are valid and unique), then tally never returns $\perp$.

To argue that Helios is strongly correct we need to convince ourselves the following property: it should not be possible for an adversary to build a ballot box BB such that a fresh honestly created ballot b for a voter id that did not vote before is rejected. In Helios, $\mathsf{Valid}(\mathsf{BB}, b) = \perp$ for a honest ballot $b = (C, \pi)$ implies that C appears before in BB. The latter can only happen with negligible probability, as the ciphertext C is the output of a probabilistic encryption scheme.