One Vote is Enough for Analysing Privacy

Stéphanie Delaune, Joseph Lallemand, Arthur Outrey

One Vote is Enough for Analysing Privacy[1]

Stéphanie Delaune [a], Joseph Lallemand [a,*] and Arthur Outrey [b]
[a] *Univ Rennes, CNRS, IRISA, France*
E-mails: stephanie.delaune@irisa.fr, joseph.lallemand@irisa.fr
[b] *ENS Lyon, France*
E-mail: arthur.outrey@ens-lyon.fr

Abstract. Electronic voting promises the possibility of convenient and efficient systems for recording and tallying votes in an election. To be widely adopted, ensuring the security of the cryptographic protocols used in e-voting is of paramount importance. However, the security analysis of this type of protocols raises a number of challenges, and they are often out of reach of existing verification tools.

In this paper, we study *vote privacy*, a central security property that should be satisfied by any e-voting system. More precisely, we propose the first formalisation of the recent BPRIV notion in the symbolic setting. To ease the formal security analysis of this notion, we propose a reduction result allowing one to bound the number of voters and ballots needed to mount an attack. We first consider the case where voters do not revote, and the ballot box is trusted before relaxing these two conditions. Our result applies on a number of case studies including several versions of Helios, Belenios, JCJ/Civitas, and Prêt-à-Voter. For some of these protocols, thanks to our result, we are able to conduct the analysis relying on the automatic tool Proverif.

Keywords: Electronic voting, Formal methods, Protocol verification, Privacy

1. Introduction

Remote electronic voting systems aim at allowing the organisation of elections over the Internet, while providing the same guarantees as traditional paper voting. Although relying on e-voting for large-scale elections is controversial, it is already in use in many lower-stakes elections today (*e.g.* the Helios [1] voting system has been used to elect the IACR board of directors since 2010), and is likely to be used even more in the future, for better or for worse. These elections may involve a large number of voters and may have an important impact on democracy when it comes to elect political leaders. It is therefore of paramount importance to ensure the security of these systems.

As for security protocols in general, formal methods provide powerful techniques to analyse e-voting systems, and prove their security. Identifying what makes a good, secure e-voting system is a complex problem that has not yet been completely solved, and is actively being researched. It is however rather universally acknowledged that a central security guarantee e-voting systems should provide is *vote privacy*. Intuitively, this property states that votes must remain secret, so that no one can learn who voted for which candidate.

[1]This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

[*]Corresponding author.

One common way of formalising vote privacy, which we will call SWAP, is to require that an attacker is not able to distinguish between the situation where Alice is voting *yes* and Bob is voting *no* from the situation where the two voters swapped their vote. That formalisation was first proposed by Benaloh [2], originally in a computational model. It has since been adapted to the symbolic setting [3], and applied to many voting schemes, *e.g.* [4–9]. The SWAP notion was originally written considering the specific case of a referendum, where the result is the number of *yes* and *no* votes. It has then been generalised to cover other kinds of elections [10], but remains limited w.r.t. the way of counting votes – essentially, it only makes sense when the result of the election is the number of votes for each candidate, excluding more complex counting procedures such as Single Transferable Vote (STV).

More recently, a new definition, called BPRIV for "ballot privacy", has been proposed to overcome such limitations, in the case of a trusted ballot box [11], with later extensions to handle an untrusted one [12]. Essentially, BPRIV lets the attacker interact with the system, and see either real ballots, or fake ones containing fake votes. Using oracles, he can choose the values of real and fake votes, and cast any ballot he can construct (in the name of corrupted voters). In the end, the tally of real ballots is published. To be BPRIV, the attacker should be unable to distinguish the two scenarios, *i.e.* no information is leaked on the ballots' content.

Privacy-type properties, and in particular vote privacy, are often expressed using a notion of behavioural equivalence [13]. A notable exception is the definition of (α, β)-privacy [14] which nevertheless relies on some notion of static equivalence. Proving equivalences is cumbersome, and is difficult to do in details by hand, as witnessed by the manual analysis of the SWAP property done for *e.g.* the Helios protocol [5] and the Norwegian one [7]. Regarding mechanisation, several mature tools are available for analysing trace properties such as secrecy or authentication in the symbolic setting: most notably, Proverif [15, 16] and Tamarin [17]. These tools support equivalence properties [18, 19], although they remain limited to a restricted form of equivalence, called *diff-equivalence*. Some e-voting schemes have been analysed with these automated tools in the symbolic model, *e.g.* the Neuchâtel [8] or BeleniosVS [20] protocols. Proverif even has an extension called ProSwapper [21], that specifically handles swapped branches that typically occur in the SWAP definition. These tools have proved very helpful for the study of e-voting systems. However, they still suffer from limitations that restrict their applicability, as they *e.g.* cannot handle homomorphic encryption, or manipulate lists of arbitrary size to encode the bulletin board, and tend to quickly run into performance issues when the number of agents in parallel increases.

An interesting option to ease the security analysis is to rely on reduction results. This approach has been used to bound the number of agents involved in an attack for both reachability [22], and equivalence properties [23]. Reduction results bounding the number of sessions [24, 25] have also been proposed in more restricted settings. All these results do not apply in the context of e-voting protocols. Here, we would like to bound the number of voters (agents) participating in the election. However, since only one vote is counted for each voter, we can *not* replace a session played by A by one played by B, as was done *e.g.* in [23]. The only existing result in that context is the result proposed in [6], where the authors give bounds on the number of voters and ballots – respectively 3 and 10 – needed for an attack on the SWAP notion This allows them to carry out several case studies using Proverif. No such results, however, exist for the newer and more general BPRIV definition.

Contributions. Our contributions are threefold. First, we propose a definition of BPRIV adapted for the symbolic model. BPRIV has been first introduced in the computational setting where some subtleties regarding the communication model have been overlooked. In the computational setting, for instance, the casting of a ballot is handled by an oracle adding it to the ballot box. This means it is implicitly assumed that a ballot cast will necessarily reach the ballot box, and this is an important assumption when analysing weeding-based protocols (where duplicate ballots are eliminated before tallying).

Second, we identify some conditions under which BPRIV can be analysed considering only *one* honest voter and k dishonest ones, casting at most k ballots in total. The bound k depends on a property of the procedure used to count votes which we define. Actually, in most usual cases, we have $k = 1$, and the number of ballots being tallied is reduced to 1. These reduction results are generic, in particular we do not restrict the equational theory, and our result applies for different counting functions. We first establish the reduction result in the setting where voters do not revote, and the ballot box is trusted. We then propose two extensions: we show the same result still holds when allowing revote, and when facing a dishonest ballot box. In the case of the untrusted ballot box, this requires us to propose an adaptation of the game-based definition of [12] to the symbolic model.

Finally, we apply our result on several e-voting protocols from the literature relying on the tool Proverif. Even if our theoretical reductions result are generic, we are limited in practice by the features offered by the tools (*e.g.* homomorphic encryption is not supported by existing tools). Nevertheless, relying on Proverif, we successfully establish that BPRIV holds for an arbitrary number of voters in several cases. Our bounds for BPRIV, better than those obtained in [6] when considering SWAP, allow us to analyse many protocols in a reasonable time (whereas several hours were needed in some cases in [6]). We also identify an issue in the security analysis performed in [6] where a protocol has been declared secure while it is not.

This paper is an extended version of our work [26], published at the 27$^{\text{th}}$ ESORICS conference (2022): in particular, the BPRIVD definition as well as the reduction result established in Section 6 to deal with the case of a dishonest ballot box is entirely new.

2. Modelling security protocols

In this section, we introduce background notions on protocol modelling. We model security protocols in the symbolic model with a process algebra inspired from the applied pi-calculus [27]. Participants are represented by processes, while messages exchanged between participants are represented by terms. Our model is mostly standard, except that in order to model the stateful nature of e-voting protocols, we consider memory cells, that can store a persistent state across processes. We need to avoid concurrent accesses to memory cells while updating them: to that end, we use a specific instruction that atomically appends a message to the content of a memory cell.

2.1. Messages

We assume an infinite set $\mathcal{N}$ of *names* used to model keys, nonces, *etc.* We consider two infinite and disjoint sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. Variables in $\mathcal{X}$ are used to refer *e.g.* to input messages, and variables in $\mathcal{W}$, called *handles*, are used as pointers to messages learned by the attacker. Lastly, we

consider two disjoint sets of constant symbols, denoted Σ_0 and Σ_{err}. Constants in Σ_0 represent public values, *e.g.* identities, nonces or keys drawn by the attacker. This set is assumed to be infinite. Constants in Σ_{err} will typically refer to error messages. We fix a *signature* $\Sigma = \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{d}}$ consisting of a finite set of function symbols together with their arity. We distinguish between *constructors* in Σ_{c} and *destructors* in Σ_{d}. We denote $\Sigma^+ = \Sigma_{\mathsf{c}} \uplus \Sigma_0 \uplus \Sigma_{\mathsf{err}}$. We note $\mathcal{T}(\mathcal{F}, \mathcal{D})$ the set of terms built from elements in $\mathcal{D}$ by applying function symbols in the signature $\mathcal{F}$. The set of names (resp. variables) occurring in a term t is denoted $\mathsf{names}(t)$ (resp. $\mathsf{var}(t)$). A term t is *ground* if $\mathsf{var}(t) = \emptyset$. We refer to elements of $\mathcal{T}(\Sigma^+, \mathcal{N})$ as *messages*.

Example 1. *We consider the signature* $\Sigma_{\mathsf{err}} = \{\mathsf{err}_{\mathsf{vote}}, \mathsf{err}_{\mathsf{invalid}}\}$ *to model error messages. The signature* $\Sigma_{\mathsf{list}} = \{\mathsf{nil}, \mathsf{hd}, \mathsf{tl}, ::\}$ *allows us to model lists of arbitrary size. We often write* $[t_1, \ldots, t_n]$ *for* $t_1 :: \cdots :: t_n :: \mathsf{nil}$. *The operators* hd *and* tl *are used to retrieve the head and the tail of a list. Lastly, we consider* $\Sigma_{\mathsf{ex}} = \{\mathsf{aenc}, \mathsf{adec}, \mathsf{pk}, \mathsf{zkp}, \mathsf{check}_{\mathsf{zkp}}, \mathsf{true}, \langle\ \rangle_3, \mathsf{proj}_1^3, \mathsf{proj}_2^3, \mathsf{proj}_3^3, \mathsf{yes}, \mathsf{no}\}$ *to model asymmetric encryption, zero-knowledge proofs, and pairing operators. As a running example, we will consider a model of the Helios protocol (in its original version, as seen in [5]) and* $\Sigma_{\mathsf{Helios}} = \Sigma_{\mathsf{ex}} \cup \Sigma_{\mathsf{list}}$ *where symbols in* Σ_{Helios} *are constructors.*

Let $\mathsf{id}_{\mathsf{H}} \in \Sigma_0$, $r, sk \in \mathcal{N}$, *and* $pk = \mathsf{pk}(sk)$. *Intuitively,* id_{H} *represents the identity of a honest voter, and* yes *her vote (these data are known to the attacker), whereas* r *and* sk *are private names, modelling respectively the randomness used in the encryption and the private key of the authority. Let* $e_{\mathsf{yes}} = \mathsf{aenc}(\mathsf{yes}, pk, r)$, *and* $b_{\mathsf{yes}}^{\mathsf{id}_{\mathsf{H}}} = \langle \mathsf{id}_{\mathsf{H}}, e_{\mathsf{yes}}, \mathsf{zkp}(e_{\mathsf{yes}}, \mathsf{yes}, r, pk)\rangle_3$. *The first term encrypts the vote, and the second one is the ballot sent by the voter in the voting phase of Helios.*

An element of $\mathcal{T}(\Sigma^+ \cup \Sigma_{\mathsf{d}}, \mathcal{W})$ is called a *recipe* and models a computation performed by the attacker using his knowledge. A *substitution* σ is a mapping from variables to messages, and $t\sigma$ is the application of σ to term t, which consists in replacing each variable x in t with $\sigma(x)$. A *frame* ϕ is a substitution that maps variables from $\mathcal{W}$ to messages, and is used to store an attacker's knowledge.

Example 2. *Continuing Example 1, we consider the equational theory* E_{ex} *given below and* $\mathsf{E}_{\mathsf{list}} := \{\mathsf{hd}(x :: y) = x,\ \mathsf{tl}(x :: y) = y\}$.

$$\mathsf{E}_{\mathsf{ex}} = \left\{ \begin{array}{c} \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y), z), y) = x \quad \mathsf{proj}_i^3(\langle x_1, x_2, x_3\rangle_3) = x_i \quad \textit{with } i \in \{1,2,3\} \\ \mathsf{check}_{\mathsf{zkp}}(\mathsf{zkp}(\mathsf{aenc}(x,y,z), x, z, y), \mathsf{aenc}(x,y,z), y) \ = \ \mathsf{true} \end{array} \right\}$$

We have $\mathsf{adec}(e_{\mathsf{yes}}, sk) =_{\mathsf{E}_{\mathsf{ex}}} v$, *and* $\mathsf{check}_{\mathsf{zkp}}(\mathsf{proj}_3^3(b_{\mathsf{yes}}^{\mathsf{id}_{\mathsf{H}}}), v, r, pk) =_{\mathsf{E}_{\mathsf{ex}}} \mathsf{true}$.

In order to provide a meaning to constructor symbols, we equip (constructor) terms with an *equational theory*. We assume a set E of equations over $\mathcal{T}(\Sigma_{\mathsf{c}}, \mathcal{X})$, and define $=_{\mathsf{E}}$ as the smallest congruence containing E that is closed under substitutions.

In addition, the semantics of destructor symbols is given by a set $\mathcal{R}$ of *ordered rewriting rules* of the form $\mathsf{g}(M_1, \ldots, M_n) \to M_0$ with $M_0, M_1, \ldots, M_n \in \mathcal{T}(\Sigma_c, \mathcal{X})$. A ground expression D can be rewritten in D' if there is a position p in D, a rewrite rule $\mathsf{g}(M_1, \ldots, M_n) \to M_0$ and a substitution θ from variables to ground terms such that $D|_p =_{\mathsf{E}} \mathsf{g}(M_1\theta, \ldots, M_n\theta)$, and $D' =_{\mathsf{E}} D[M_0\theta]_p$, *i.e.* D in which the subterm at position p has been replaced with $M_0\theta$. In case more than one rule may be applied at position p, only the first such rule can be effectively used. Moreover, we assume that

the last rewriting rule defining a destructor g is of the form $\mathsf{g}(x_1, \ldots, x_n) \to M_0$ with $x_1, \ldots, x_n$ distinct variables, and thus always applies. Given a ground expression D, it may be possible to rewrite it (in an arbitrary number of steps) into a ground (constructor) term M: in that case, this term is noted $D{\Downarrow}$, and we say that D *evaluates to* $D{\Downarrow}$. Note that, in our setting, a computation never fails.

We extend the notation $=_{\mathsf{E}}$ to terms that may contain destructor symbols (that never fail). We write $u =_{\mathsf{E}} v$ when $u{\Downarrow} =_{\mathsf{E}} v{\Downarrow}$.

Example 3. *Consider* $\Sigma_{\mathsf{d}} = \{\mathsf{ite}\}$ *where* ite *is a destructor symbol of arity* 4 *that can be used to model conditional branching with the following ordered rewriting rules:*

$$\mathsf{ite}(x, x, y, z) \to y \qquad \mathsf{ite}(x, x', y, z) \to z$$

The destructor defined in Example 3 may seem of little use, since it does not let an attacker compute any value he did not already know. It does indeed not bring extra power to the attacker. However, when dealing with the case of a dishonest ballot box, having such a construction will make it easier to write recipes used in our reduction result.

In the following, we consider an arbitrary signature $\Sigma = \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{d}}$, and we simply assume that the equational theory E (equations built over Σ_{c} only), contains at least the formalisation of lists given in Example 1 and Example 2, *i.e.* $\Sigma_{\mathsf{list}} \subseteq \Sigma$ and $\mathsf{E}_{\mathsf{list}} \subseteq \mathsf{E}$.

2.2. Processes

We model protocols using a process calculus. We consider an infinite set of channel names $\mathcal{C}\mathsf{h} = \mathcal{C}\mathsf{h}_{\mathsf{pub}} \uplus \mathcal{C}\mathsf{h}_{\mathsf{pri}}$, partitioned into infinite sets of public and private channel names. We also assume an infinite set $\mathcal{M}$ of names to represent memory cells (used to store states). The syntax of processes is:

$$\begin{array}{lll} P, Q ::= \mathsf{0} & \mid \mathsf{out}(c, u).\ P & \mid m := u.\ P \\ \quad \mid P \mid Q & \mid \mathsf{in}(c, x).\ P & \mid \mathsf{read}\ m\ \mathsf{as}\ x.\ P \\ \quad \mid\ !\, P & \mid\ !\, \mathsf{new}\ d.\ \mathsf{out}(c, d).\ P & \mid \mathsf{append}(c, u, m).\ P \\ \quad \mid \mathsf{new}\ n.\ P & \mid \mathsf{let}\ x = u\ \mathsf{in}\ P & \mid \mathsf{phase}\ i.\ P \\ \quad \mid \mathsf{new}\ d.\ P & \mid \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q & \end{array}$$

where $n \in \mathcal{N}$, $x \in \mathcal{X}$, $m \in \mathcal{M}$, $u \in\ \mathcal{T}(\Sigma^+, \mathcal{X} \cup \mathcal{N})$, $d \in \mathcal{C}\mathsf{h}_{\mathsf{pri}}$, $c \in \mathcal{C}\mathsf{h}$, $i \in \mathbb{N}$.

This syntax is rather standard, except for the memory cell operations. Intuitively, $\mathsf{read}\ m\ \mathsf{as}\ x$ stores the content of m in the variable x, whereas $\mathsf{append}(c, u, m)$ represents the agent with channel c appending u to memory m. In addition, we use a special construct $!\, \mathsf{new}\ d.\ \mathsf{out}(c, d).\ P$, to generate as many times as needed a new public channel d and link it to channel c, in a single step. This could be encoded using the other instructions, but having a separate construction lets us mark it in the execution traces, which is convenient for the proofs. The constructs $\mathsf{in}(c, x).P$, $\mathsf{let}\ x = u\ \mathsf{in}; P$, and $\mathsf{read}\ m\ \mathsf{as}\ x.\ P$ bind x in P. Note that destructor symbols are not allowed in the syntax of processes. In case the recipe used by the attacker contains such a destructor, the hypothesis imposing that a computation never fails ensures that the resulting term is indeed a message. Given a process P, $fv(P)$ denotes its free variables, and we say that it is *ground* when $fv(P) = \emptyset$. Moreover, we usually omit the final $\mathsf{0}$ in processes.

Example 4. *Continuing our running example, we consider the process P:*

$$P = \mathsf{in}(c,b).\ \mathsf{if}\ \langle \mathsf{check}_{\mathsf{zkp}}(\mathsf{proj}_3^3(b), \mathsf{proj}_2^3(b), \mathsf{pk}(sk)), \mathsf{proj}_1^3(b)\rangle = \langle \mathsf{true}, \mathsf{id}_\mathsf{D}\rangle \\ \mathsf{then}\ \mathsf{out}(c,b).\ \mathsf{append}(c,b,m_{\mathsf{bb}})\ \mathsf{else}\ \mathsf{out}(b,\mathsf{err}_{\mathsf{invalid}}).$$

where $b \in \mathcal{X}$, $sk \in \mathcal{N}$, and $\mathsf{id}_\mathsf{D} \in \Sigma_0$. This represents an agent that receives a ballot b as input, and then checks the validity of the zero knowledge proof contained in b, as well as the identity of the voter. Depending on the outcome of this test, it either outputs the ballot and appends it in the cell m_{bb} modelling the ballot box, or simply outputs an error message.

Definition 1. *A* configuration *is a tuple $(i; \mathcal{P}; \phi; M)$, composed of an integer i, a multiset $\mathcal{P}$ of ground processes, a frame ϕ, and a mapping M from a subset of memory names $\mathcal{M}$ to messages. We write $\mathcal{P}$ instead of $(0; \mathcal{P}; \emptyset; \emptyset)$.*

The semantics of our calculus is defined as a transition relation $\overset{a}{\Rightarrow}$ on configurations. Each transition step is labelled with an action a representing what the attacker can observe when performing it (it can be an input, an output, an append action, or a silent action ϵ). This relation is defined in a standard manner, and is fully displayed in Figure 1.

For instance, considering an input on a public channel, *i.e.* the rule In, the attacker can inject any message $R\phi$ he is able to build using his current knowledge ϕ. The outputs performed on a public channel are made available to the attacker either directly through the label when it corresponds to an error message (rule Out-Err), or indirectly through the frame (rule Out). The rule Append corresponding to our new append action $\mathsf{append}(c,u,m)$ simply consists in appending a term u to the memory cell m.

Definition 2. *The set of* traces *of a configuration K is defined as*

$$\mathsf{traces}(K) = \{(\mathsf{tr}, \phi) \mid \exists i, \mathcal{P}, M \text{ such that } K \overset{\mathsf{tr}}{\Rightarrow}{}^{\star} (i; \mathcal{P}; \phi; M)\}$$

where $\dot{\Rightarrow}^{\star}$ is the reflexive transitive closure of $\dot{\Rightarrow}$, concatenating all (non-silent) actions into the sequence tr.

Example 5. *Continuing Example 4 with $\phi_{\mathsf{yes}} = \{\mathsf{w}_0 \mapsto \mathsf{pk}(sk), \mathsf{w}_1 \mapsto b_{\mathsf{yes}}^{\mathsf{id_H}}\}$, and the configuration $K_0^{\mathsf{yes}} = (2; \{P\}; \phi_{\mathsf{yes}}; \{m_{\mathsf{bb}} \mapsto \mathsf{nil}\})$. We have:*

$$K_0^{\mathsf{yes}} \xrightarrow{\mathsf{in}(c,\mathsf{w}_1).\mathsf{out}(c,\mathsf{err}_{\mathsf{invalid}})} (2; \emptyset; \{\mathsf{w}_0 \mapsto \mathsf{pk}(sk), \mathsf{w}_1 \mapsto b_{\mathsf{yes}}^{\mathsf{id_H}}\}; \{m_{\mathsf{bb}} \mapsto \mathsf{nil}\})$$

$$K_0^{\mathsf{yes}} \xrightarrow{\mathsf{in}(c,R_0).\mathsf{out}(c,\mathsf{w}_2).\mathsf{append}(c)} (2; \emptyset; \{\mathsf{w}_0 \mapsto \mathsf{pk}(sk), \mathsf{w}_1 \mapsto b_{\mathsf{yes}}^{\mathsf{id_H}}, \mathsf{w}_2 \mapsto b_{\mathsf{yes}}^{\mathsf{id_D}}\}; \{m_{\mathsf{bb}} \mapsto b\})$$

with $R_0 = \langle \mathsf{id}_\mathsf{D}, \mathsf{proj}_2^3(\mathsf{w}_1), \mathsf{proj}_3^3(\mathsf{w}_1)\rangle_3$, and $b_{\mathsf{yes}}^{\mathsf{id_D}} = R_0\phi_{\mathsf{yes}}^{\mathsf{id_H}} =_{\mathsf{E_{ex}}} \langle \mathsf{id}_\mathsf{D}, e_{\mathsf{yes}}, zkp\rangle_3$. The term zkp here denotes the zero-knowledge proof from $b_{\mathsf{yes}}^{\mathsf{id_H}}$. It does not contain the identity of the voter who computes it, and can therefore be reused by a dishonest voter to cast the ballot in her own name.

2.3. Equivalences

Our definition of the BPRIV property relies on two usual notions of equivalence in the symbolic model: *static equivalence*, for the indistinguishability of sequences of messages, and *trace equivalence*, for the indistinguishability of processes.

PAR $(i; \{\!|P_1 \mid P_2|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|P_1, P_2|\!\} \cup \mathcal{P}; \phi; M)$

ZERO $(i; \{\!|0|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \mathcal{P}; \phi; M)$

NEW-N $(i; \{\!|\mathsf{new}\ n.\ P|\!\} \cup \mathcal{P}; \phi) \overset{\epsilon}{\Rightarrow} (i; \{\!|P\{n \mapsto n'\}|\!\} \cup \mathcal{P}; \phi; M)$
if $n \in \mathcal{N}$, and $n' \in \mathcal{N}$ is a fresh name not occurring in any message considered

NEW-CH $(i; \{\!|\mathsf{new}\ c.\ P|\!\} \cup \mathcal{P}; \phi) \overset{\epsilon}{\Rightarrow} (i; \{\!|P\{c \mapsto c'\}|\!\} \cup \mathcal{P}; \phi; M)$
if $c \in \mathcal{C}\mathsf{h}_{\mathsf{pri}}$, and $c' \in \mathcal{C}\mathsf{h}_{\mathsf{pri}}$ is a fresh channel not occurring in any process considered

OUT-ERR $(i; \{\!|\mathsf{out}(c, \mathsf{c}_{\mathsf{err}}).\ P|\!\} \cup \mathcal{P}; \phi; M) \xrightarrow{\mathsf{out}(c,\mathsf{c}_{\mathsf{err}})} (i; \{\!|P|\!\} \cup \mathcal{P}; \phi; M)$
if $c \in \mathcal{C}\mathsf{h}_{\mathsf{pub}}$, $\mathsf{c}_{\mathsf{err}} \in \Sigma_{\mathsf{err}}$

OUT $(i; \{\!|\mathsf{out}(c, u).\ P|\!\} \cup \mathcal{P}; \phi; M) \xrightarrow{\mathsf{out}(c,w)} (i; \{\!|P|\!\} \cup \mathcal{P}; \phi \cup \{\mathsf{w} \mapsto u\}; M)$
if $c \in \mathcal{C}\mathsf{h}_{\mathsf{pub}}$, u ground term not equal (modulo E) to a constant in Σ_{err}, $\mathsf{w} \in \mathcal{W} \setminus \mathsf{dom}(\phi)$

IN $(i; \{\!|\mathsf{in}(c, x).\ P|\!\} \cup \mathcal{P}; \phi; M) \xrightarrow{\mathsf{in}(c,R)} (i; \{\!|P\{x \mapsto R\phi\Downarrow\}|\!\} \cup \mathcal{P}; \phi; M)$
if $c \in \mathcal{C}\mathsf{h}_{\mathsf{pub}}$, and R is a recipe such that $\mathsf{var}(R) \subseteq \mathsf{dom}(\phi)$

PRIV $(i; \{\!|\mathsf{out}(c, u).\ P,\ \mathsf{in}(c, x).\ Q|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|P, Q\{x \mapsto u\}|\!\} \cup \mathcal{P}; \phi; M)$
if $c \in \mathcal{C}\mathsf{h}_{\mathsf{pri}}$, and u is a ground term

LET $(i; \{\!|\mathsf{let}\ x = u\ \mathsf{in}\ P|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|P\{x \mapsto u\}|\!\} \cup \mathcal{P}; \phi; M)$ if u is ground

THEN $(i; \{\!|\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|P|\!\} \cup \mathcal{P}; \phi; M)$
if u, v are ground and $u =_{\mathsf{E}} v$

ELSE $(i; \{\!|\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|Q|\!\} \cup \mathcal{P}; \phi; M)$
if u, v are ground and $u \neq_{\mathsf{E}} v$

REPL-CH
$(i; \{\!|!\,\mathsf{new}\ d.\mathsf{out}(c, d).P|\!\} \cup \mathcal{P}; \phi; M) \xrightarrow{\mathsf{sess}(c,d')} (i; \{\!|P\{d \mapsto d'\}, !\,\mathsf{new}\ d.\mathsf{out}(c, d).P|\!\} \cup \mathcal{P}; \phi; M)$

REPL $(i; \{\!|!P|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|P, !P|\!\} \cup \mathcal{P}; \phi; M)$

WRITE $(i; \{\!|m := u.\ P|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|P|\!\} \cup \mathcal{P}; \phi; M\{m \mapsto u\})$ if u is ground

READ $(i; \{\!|\mathsf{read}\ m\ \mathsf{as}\ x.\ P|\!\} \cup \mathcal{P}; \phi; M) \overset{\epsilon}{\Rightarrow} (i; \{\!|P\{x \mapsto u\}|\!\} \cup \mathcal{P}; \phi; M)$
if $M(m)$ is a message

APPEND
$(i; \{\!|\mathsf{append}(c, u, m)|\!\}.\ P \cup \mathcal{P}; \phi; M) \xrightarrow{\mathsf{append}(c)} (i; \{\!|P|\!\} \cup \mathcal{P}; \phi; M\{m \mapsto u :: M(m)\})$
if $m \in \mathsf{dom}(M)$

PHASE $(i; \mathcal{P}; \phi; M) \xrightarrow{\mathsf{phase}\ i+1} (i + 1; \mathcal{P}'; \phi; M)$
where $\mathcal{P}' = \{\!|P \mid \mathsf{phase}\ i + 1.\ P \in \mathcal{P}|\!\} \cup \{\!|\mathsf{phase}\ j.\ P \mid \mathsf{phase}\ j.\ P \in \mathcal{P}\ \wedge\ j > i + 1|\!\}$
(keeping multiplicity)

Figure 1. Semantics of our calculus

Definition 3. *Two frames ϕ and ϕ' are* statically equivalent*, denoted by $\phi \sim \phi'$, if $\mathsf{dom}(\phi) = \mathsf{dom}(\phi')$ and for any recipes $R_1, R_2 \in \mathcal{T}(\Sigma^+ \cup \Sigma_\mathsf{d}, \mathsf{dom}(\phi))$, we have:*

$$R_1\phi =_\mathsf{E} R_2\phi \;\Leftrightarrow\; R_1\phi' =_\mathsf{E} R_2\phi'.$$

When establishing our reduction result, we will reason on the notion of static equivalence. In particular, we will assume an attack trace exists, and that this attack comes from publishing the result of the election, *i.e.* that the two processes are in trace equivalence until the result is output. In such cases, we will deduce that the results output by the tally on either side are different (modulo E). This result is formally stated and proved below and will be used in the proof of our main result.

Lemma 1. *Let t_L and t_R be two public terms,* i.e. *$t_\mathsf{L}, t_\mathsf{R} \in \mathcal{T}(\Sigma_\mathsf{c}, \Sigma_0)$. Let $\phi_\mathsf{L}, \phi_\mathsf{R}$ be two frames such that $\phi_\mathsf{L} \sim \phi_\mathsf{R}$, and $\mathsf{w_{tall}} \in \mathcal{W} \setminus \mathsf{dom}(\phi_\mathsf{L})$. We have:*

$$\phi_\mathsf{L} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{L}\} \not\sim \phi_\mathsf{R} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{R}\} \text{ if, and only if, } t_\mathsf{L} \neq_\mathsf{E} t_\mathsf{R}.$$

Proof. First, assume that $t_\mathsf{L} \neq_\mathsf{E} t_\mathsf{R}$. In such a case, let $M = \mathsf{w_{tall}}$, and $N = t_\mathsf{L} \in \mathcal{T}(\Sigma_\mathsf{c}, \Sigma_0)$. We have that the test $M = N$ holds in $\phi_\mathsf{L} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{L}\}$, and not in $\phi_\mathsf{R} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{R}\}$. Indeed, we have that: $M\phi_\mathsf{L} = \mathsf{w_{tall}}\phi_\mathsf{L} = t_\mathsf{L} = N\phi_\mathsf{L}$; and $M\phi_\mathsf{R} = \mathsf{w_{tall}}\phi_\mathsf{R} = t_\mathsf{R} \neq_\mathsf{E} t_\mathsf{L} = N\phi_\mathsf{R}$. Therefore, we have that $\phi_\mathsf{L} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{L}\} \not\sim \phi_\mathsf{R} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{R}\}$.

Now, we assume that $\phi_\mathsf{L} \sim \phi_\mathsf{R}$, and $t_L =_\mathsf{E} t_R$. Consider w.l.o.g. a test $M = N$ that holds in $\phi_\mathsf{L} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{L}\}$. Let $M' = M\{\mathsf{w_{tall}} \mapsto t_\mathsf{L}\}$, and $N' = N\{\mathsf{w_{tall}} \mapsto t_\mathsf{L}\}$. Then $M' = N'$ is a test that holds in ϕ_L, and thus in ϕ_R (thanks to our hypothesis $\phi_\mathsf{L} \sim \phi_\mathsf{R}$). Since, $t_\mathsf{L} =_\mathsf{E} t_\mathsf{R}$, we easily conclude that $M = N$ holds in $\phi_\mathsf{R} \cup \{\mathsf{w_{tall}} \mapsto t_\mathsf{R}\}$. This allows us to conclude. □

Trace equivalence is the active counterpart of static equivalence. Two configurations are in trace equivalence if, however the attacker behaves, the resulting sequences of messages observed by the attacker are in static equivalence.

Definition 4. *Two ground processes P, Q are in* trace inclusion*, denoted by $P \sqsubseteq_t Q$, if for all $(\mathsf{tr}, \phi) \in \mathsf{traces}(P)$, there exists ϕ' such that $(\mathsf{tr}, \phi') \in \mathsf{traces}(Q)$ and $\phi \sim \phi'$. We say that P and Q are* trace equivalent*, denoted by $P \approx_t Q$, if $P \sqsubseteq_t Q$ and $Q \sqsubseteq_t P$.*

Example 6. *We can consider a configuration K_0^{no} similar to K_0^{yes} but with* no *instead of* yes *in the initial frame. We can establish that $K_0^{\mathsf{no}} \approx_t K_0^{\mathsf{yes}}$. This is a non trivial equivalence. Now, let us replace P with P^+ in both configurations, adding a simple process modelling the tally (for one vote),* e.g.

$$P^+ = P \;\mid\; \mathsf{phase}\ 3.\ \mathsf{read}\ m_{\mathsf{bb}}\ \mathsf{as}\ bb.\ \mathsf{let}\ res = \mathsf{adec}(\mathsf{proj}_2^3(bb), sk)\ \mathsf{in}\ \mathsf{out}(c_r, res).$$

The resulting trace equivalence does not *hold. This is simply due to the fact that $\mathsf{tr} = \mathsf{in}(c, R_0).\mathsf{out}(c, \mathsf{w}_2).\mathsf{append}(c).\mathsf{phase}\,3.\mathsf{out}(c_r, \mathsf{w}_3)$ can be executed starting from both configurations, and the resulting frames contains $\mathsf{w}_3 \mapsto \mathsf{no}$ on the left, and $\mathsf{w}_3 \mapsto \mathsf{yes}$ on the right. This breach of equivalence is not, strictly speaking, an attack, as the processes do not formalise the* BPRIV *property. However it follows the same idea as the ballot copy attack against Helios from [5]: a*

dishonest voter copies a honest voter's ballot, introducing an observable difference in the result. This attack can be prevented by patching Helios, either by weeding out duplicate ballots from the ballot box, or by adding the voter's id to the ZKP, which then becomes invalid for any other voter.

In the following, we will consider action-deterministic configurations. Intuitively, for an action-deterministic configuration K, once the trace tr is fixed, the configurations that are reachable following the trace tr are equal up to some α-renaming.

Definition 5. *A configuration K is* action-deterministic *if for any tr, any configurations $K_1 = (i_1; \mathcal{P}_1; \phi_1; M_1)$ and $K_2 = (i_2; \mathcal{P}_2; \phi_2; M_2)$ such that $K \xRightarrow{\mathsf{tr}} K_1$ and $K \xRightarrow{\mathsf{tr}} K_2$, we have $i_1 = i_2$, and ϕ_1 and ϕ_2 are equal modulo α-renaming of names generated during the execution.*

Consider two ground processes P and Q whose associated configurations $(0; \{P\}; \emptyset; \emptyset)$ and $(0; \{Q\}; \emptyset; \emptyset)$ are action-deterministic. A witness of non-inclusion for $P \not\sqsubseteq_t Q$ is actually a trace tr for which there exists ϕ_P such that $(\mathsf{tr}, \phi_P) \in \mathsf{traces}(P_P)$, and

- either there does not exist ϕ_Q such that $(\mathsf{tr}, \phi_Q) \in \mathsf{traces}(P_Q)$;
- or such a ϕ_Q exists and $\phi_P \not\sim \phi_Q$.

Indeed, once tr is fixed, the resulting configuration is unique up to α-renaming, thus there is no need to consider all the frames ϕ_Q such that $(\mathsf{tr}, \phi_Q) \in \mathsf{traces}(Q)$ to establish that they are not in static equivalence with ϕ_Q. It is sufficient to consider one representative.

3. Modelling the general BPRIV notion

In this section, we present our formal model of e-voting protocols, and our BPRIV privacy notion. While BPRIV itself is not novel, our symbolic formalisation is.

3.1. Modelling e-voting protocols

When modelling voting systems, we often need to encode some computations (*e.g.* performed by the ballot box) that cannot be represented by recipes (*e.g.* iterating through an arbitrary-sized list). We encode these computations as processes, that do not share any names, channels, or memory cells with the rest of the process, except for a channel to return the result of the computation.

Definition 6. *A* computation *is a process $C_d(\vec{p})$ without free names, channels, or variables (not counting those in d, $\vec{p}$), without memory cell operations, and without phases. It is parametrised by a channel d, and terms $\vec{p}$, meant to be the channel where the result is output, and the terms given as input parameters.*

This process must be such that for all inputs $\vec{p}$, there exists a ground term t_0 such that for all channel name d, we have

$$\mathsf{traces}(C_d(\vec{p})) = \{(\epsilon, \emptyset)\} \cup \{(\mathsf{out}(d, \mathsf{w}), \{\mathsf{w} \mapsto t_0\}) \mid \mathsf{w} \in \mathcal{W}\}.$$

We then call t_0 the result *of the computation. As it does not depend on the channel, we will often omit it and let $C(\vec{p})$ denote the result.*

To use such a process to compute a term inside a process P, we will typically run it in parallel with an input waiting to retrieve the result on d, followed by the continuation process. We will write as a shortcut $\mathsf{let}\ x = C(\vec{p})\ \mathsf{in}\ P$ for $\mathsf{new}\ d.\ (C_d(\vec{p}) \mid \mathsf{in}(d, x).\ P)$, where d is a fresh private channel name (*i.e.* that does not appear anywhere else in the ambiant process).

We assume a set $\mathsf{Votes} \subseteq \mathcal{T}(\Sigma, \Sigma_0)$ of public ground terms representing the possible values of the votes. A voting system is modelled as a collection of processes that model the behaviour of voters, and a process representing the tallying authority. The election process is composed of several phases.

Phases 0 and 1: Setup. In the first two phases of the process, the election material is generated and published. More precisely, the election public key is published in the initial phase, and the public credentials of voters in phase 1.

Phase 2: Casting. The voters send their ballots to the ballot box. In our model, a memory m_{bb} will play the role of the ballot box, recording all ballots received by the voting server. This ballot box will be tallied at the end of the election. In fact, as we will see later on, when writing the BPRIV property, we will rather store the lists of ballots $(bb_0, bb_{\mathsf{obs}})$ in m_{bb}, containing real and "observable" (sometimes fake) ballots. The voters' processes will first publish their ballot on a dedicated public channel, and then append it to the memory cell m_{bb}. This models the fact that voters are authenticated when they submit their ballot, and the ballot cannot be modified on its way to the ballot box. While the attacker can modify messages on the public channel, he cannot directly access the memory cell, and thus he cannot impersonate the voter to submit a different ballot. However, the attacker is able to block a ballot before it reaches the ballot box.

Each voter has a private credential $cr \in \mathcal{N}$, with an associated public credential computed by a recipe $\mathsf{Pub}(cr, u)$, that may use a random value u. Some protocols, such as Civitas, use this value to randomise the public credential, while others, such as Belenios, do not use it – in such cases we can omit it. We will, in addition, use different channel names for the public channels used by each voter. This is more convenient when reasoning about traces, as it makes it easier to observe which voters have voted in a given trace.

To model the construction of ballots, we assume a recipe Vote with 5 variables: the term $\mathsf{Vote}(pk, id, cr, v, r)$ represents a ballot generated for voter id with credential cr, public election key pk, randomness r, and containing a vote v.

When modelling vote privacy, the attacker chooses the vote v he wants the voter to use to construct the ballot. Hence, we will need to check that v is indeed a possible value for a vote, *i.e.* $v \in \mathsf{Votes}$. If the set of candidates is finite, this can be tested exhaustively. In other cases, such as write-in votes, it can be done *e.g.* if legal votes have a specific format (start with a tag, *etc.*), or trivially if any value is legal. In a voting scheme, once a ballot is received by the voting server, another check is performed to ensure the ballot is *valid*, *i.e.* correctly constructed. The exact nature of this validity test depends on the construction of the ballot, and thus on the protocol considered. Typically, it can consist in verifying signatures or zero-knowledge proofs included in the ballot. To keep our model generic, we simply assume a recipe Valid with four variables: $\mathsf{Valid}(id, pcr, b, pk)$ represents the validity test performed for the agent id, whose public credential is pcr, who submits a ballot b. The term it computes is meant to be equal to true if, and only if, ballot b cast by id is valid w.r.t. her public credential pcr and the election public key pk. We incorporate this validity check directly in the process modelling the voter, before publishing and adding the ballot to m_{bb}.

In reality, it is performed by the ballot box, but this modelling choice is both simpler (no need for an extra process) and closer to the cryptographic game (where the voting oracle performs the test).

The formal definition of the voter's process is given in Section 3.2 as it incorporates elements specific to the modelling of the property.

Example 7. *Continuing Example 2, for Helios, we use the following recipes:*

$$\mathsf{Vote}_{\mathsf{Helios}}(pk, id, v, r) = \langle id, \mathsf{aenc}(v, pk, r), \mathsf{zkp}(\mathsf{aenc}(v, pk, r), v, r, pk)\rangle_3$$
$$\mathsf{Valid}_{\mathsf{Helios}}(id, b, pk) = \mathsf{check}_{\mathsf{zkp}}(\mathsf{proj}_3^3(b), \mathsf{proj}_2^3(b), pk).$$

Phase 3: Tallying. In the final phase, the $\mathsf{Tally}(sk)$ process is in charge of reading the contents of the ballot box, and using the key sk to compute and publish the result on a dedicated channel c_r. To leave it as generic as possible, we simply assume a computation $C_{\mathsf{Tally}}(bb, sk)$, that takes as parameters a list bb of ballots, and sk, and computes the result as specified by the protocol. We then assume the following form for Tally:

$$\mathsf{Tally}(sk) = \mathsf{read}\ m_{\mathsf{bb}}\ \mathsf{as}\ bb.\ \mathsf{let}\ res = C_{\mathsf{Tally}}(bb, sk)\ \mathsf{in}\ \mathsf{out}(c_r, res).$$

Example 8. *We continue Example 7 and we consider for simplicity the case of a referendum with two possible votes* yes *and* no*. We assume function symbols* $\mathsf{zero}/0$ *and* $\mathsf{incr}/1$*, without any associated equations, that we use to count in unary. Slightly abusing notations with the use of pattern-matching in input, the tallying computation can be written as follows:*

$$\begin{array}{l} C_{\mathsf{Tally}}(bb, sk) = \\ \quad \mathsf{new}\ c.\ (\ \ \mathsf{out}(c, \langle \mathsf{zero}, \mathsf{zero}, bb\rangle_3) \\ \qquad\qquad \mid\ \mathsf{in}(c, \langle x, y, \mathsf{nil}\rangle_3).\ \mathsf{out}(c_r, \langle x, y\rangle) \\ \qquad\qquad \mid\ !\ \mathsf{in}(c, \langle x, y, \langle id, b, p\rangle_3 :: l\rangle_3).\ \mathsf{let}\ v = \mathsf{adec}(b, sk)\ \mathsf{in} \\ \qquad\qquad\quad \mathsf{if}\ v = \mathsf{yes}\ \mathsf{then}\ \mathsf{out}(c, \langle \mathsf{incr}(x), y, l\rangle_3)\ \mathsf{else}\ \mathsf{out}(c, \langle x, \mathsf{incr}(y), l\rangle_3).\ ) \end{array}$$

3.2. A symbolic definition of BPRIV

We model vote privacy by adapting the BPRIV notion, originally formulated as a cryptographic game [11], to our symbolic setting. The idea remains the same as for the original notion: an attacker should not learn any information on the votes contained in the ballots, other than the final result of the election. This is modelled by letting the attacker suggest two possible values for the vote of each honest voter: a "real" one and a "fake" one. The attacker then sees the honest voters' ballots, containing either the real or fake votes, and then in the end the real result of the election, computed on the real votes. We model the behaviour of honest voter id, who uses channel c, private and public credentials cr, pcr, and election public key pk in these two scenarios by the

two following processes.

$$
\begin{array}{l}
\mathsf{HVoter}^{\mathsf{L}}(c, id, cr, pcr, pk) = \\
\quad \mathsf{in}(c, z). \\
\quad \mathsf{let}\ (v^0, v^1) = (\mathsf{proj}_1^2(z), \mathsf{proj}_2^2(z))\ \mathsf{in} \\
\quad \mathsf{if}\ v^0, v^1 \in \mathsf{Votes}\ \mathsf{then} \\
\quad\quad \mathsf{new}\ r^0.\ \mathsf{new}\ r^1. \\
\quad\quad \mathsf{let}\ b^0 = \mathsf{Vote}(pk, id, cr, v^0, r^0)\ \mathsf{in} \\
\quad\quad \mathsf{let}\ b^1 = \mathsf{Vote}(pk, id, cr, v^1, r^1)\ \mathsf{in} \\
\quad\quad \mathsf{if}\ \mathsf{Valid}(id, pcr, b^0, pk) = \mathsf{true} \\
\quad\quad\quad \mathsf{then}\ \mathsf{out}(c, b^0).\ \mathsf{append}(c, b^0, m_{\mathsf{bb}}) \\
\quad\quad\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{invalid}}) \\
\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{vote}})
\end{array}
$$

$$
\begin{array}{l}
\mathsf{HVoter}^{\mathsf{R}}(c, id, cr, pcr, pk) = \\
\quad \mathsf{in}(c, z). \\
\quad \mathsf{let}\ (v^0, v^1) = (\mathsf{proj}_1^2(z), \mathsf{proj}_2^2(z))\ \mathsf{in} \\
\quad \mathsf{if}\ v^0, v^1 \in \mathsf{Votes}\ \mathsf{then} \\
\quad\quad \mathsf{new}\ r^0.\ \mathsf{new}\ r^1. \\
\quad\quad \mathsf{let}\ b^0 = \mathsf{Vote}(pk, id, cr, v^0, r^0)\ \mathsf{in} \\
\quad\quad \mathsf{let}\ b^1 = \mathsf{Vote}(pk, id, cr, v^1, r^1)\ \mathsf{in} \\
\quad\quad \mathsf{if}\ \mathsf{Valid}(id, pcr, b^1, pk) = \mathsf{true} \\
\quad\quad\quad \mathsf{then}\ \mathsf{out}(c, b^1).\ \mathsf{append}(c, b^0, m_{\mathsf{bb}}) \\
\quad\quad\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{invalid}}) \\
\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{vote}})
\end{array}
$$

In both cases, the process receives the two possible vote instructions (v^0, v^1) from the attacker, and constructs two corresponding ballots b^0, b^1. It then tests for validity, and publishes, either the real b^0 (on the left), or the fake b^1 (on the right). However, since the result is always computed on the real votes, the ballot secretly added to the ballot box m_{bb} is always b^0. If any of the tests fail, we return error messages $\mathsf{err}_{\mathsf{invalid}}, \mathsf{err}_{\mathsf{vote}} \in \Sigma_{\mathsf{err}}$.

The attacker has complete control over the ballots submitted by dishonest voters. Hence, we model them by a process that receives an arbitrary ballot from the attacker, and adds it to the ballot box m_{bb} after checking its validity:

$$
\begin{array}{l}
\mathsf{DVoter}(c, id, cr, pcr, pk) = \mathsf{in}(c, b).\ \mathsf{if}\ \mathsf{Valid}(id, pcr, b, pk) = \mathsf{true} \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{then}\ \mathsf{out}(c, b).\ \mathsf{append}(c, b, m_{\mathsf{bb}})\ \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{invalid}}).
\end{array}
$$

To a reader used to symbolic modelling of protocols, it may seem strange that dishonest voters are modelled by a process, rather than being left completely under the control of the attacker. It may similarly be surprising that the voters' processes include the validity checks and write directly to the ballot box, while these operations are not actually performed by the voter but by an independent entity (typically the server storing the ballot box). While not essential for our results, we decided to adopt this style of modelling to follow more closely the original formulation as a cryptographic game. In that formalism, the protocol and the scenario considered are modelled as oracles. The attacker has access to an oracle for each voter, and the oracle takes care of everything that happens when the voter votes. For honest voters, the attacker may submit two possible votes to the oracle, and the oracle constructs ballots accordingly, checks their validity, and records them in the ballot box. For dishonest voters, he may submit any ballot, and the oracle checks its validity and adds it to the box. Our symbolic processes are written in the same spirit: they should be seen as models of what happens when a voter votes, rather than directly models of the voter's behaviour.

We then consider n voters: for each $i \in [\![1, n]\!]$, we let $\vec{v_i} = (c_i, id_i, cr_i, pcr_i)$, where $c_i \in \mathcal{C}\mathsf{h}_{\mathsf{pub}}$ is a dedicated public channel, $id_i \in \Sigma_0$ is the voter's identity, $cr_i \in \mathcal{N}$ her private credential, and $pcr_i = \mathsf{Pub}(cr_i, u_i)$ her public credential randomised with $u_i \in \mathcal{N}$. We will say that for $i \neq j$, $\vec{v_i}$ and $\vec{v_j}$ are *distinct voters*, to signify that they have different identities, credentials, and channels, *i.e.* $c_i \neq c_j \ \wedge\ id_i \neq id_j \ \wedge\ cr_i \neq cr_j \ \wedge\ u_i \neq u_j \ \wedge\ u_i \neq cr_j \ \wedge\ cr_i \neq u_j$.

We then define the BPRIV property as follows.

Definition 7. *A voting scheme is* BPRIV *for p honest voters and $n-p$ dishonest voters, written* $\mathsf{BPRIV}(p, n-p)$*, if* $\mathsf{Election}^{\mathsf{L}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}) \approx_t \mathsf{Election}^{\mathsf{R}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$ *where*

$$\begin{aligned}\mathsf{Election}^{\mathsf{X}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}) = \mathsf{new}\ sk.\ m_{\mathsf{bb}} := \mathsf{nil}.\ \mathsf{out}(ch, \mathsf{pk}(sk)).\\ (\ \mathsf{phase}\ 1.\ \mathsf{out}(c_1, pcr_1).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{X}}(\overrightarrow{v_1}, \mathsf{pk}(sk))\\ \mid \ldots\\ \mid \mathsf{phase}\ 1.\ \mathsf{out}(c_p, pcr_p).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{X}}(\overrightarrow{v_p}, \mathsf{pk}(sk))\\ \mid \mathsf{phase}\ 1.\ \mathsf{out}(c_{p+1}, \langle cr_{p+1}, pcr_{p+1}\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v}_{p+1}, \mathsf{pk}(sk))\\ \mid \ldots\\ \mid \mathsf{phase}\ 1.\ \mathsf{out}(c_n, \langle cr_n, pcr_n\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v_n}, \mathsf{pk}(sk))\\ \mid \mathsf{phase}\ 3.\ \mathsf{Tally}(sk)\ )\end{aligned}$$

with $ch \in \mathcal{C}\mathsf{h}_{\mathsf{pub}}$, $\mathsf{X} \in \{\mathsf{L}, \mathsf{R}\}$.

While we designed our symbolic definition to follow as closely as possible the original computational formulation of the property, there are two notable differences.

First, in the original notion, the oracle modelling honest voters was executed atomically: once the adversary submits his vote instructions, the generated ballot is immediately placed in the ballot box. In contrast, in our formalism, we allow executions where the process HVoter is not executed until its end: the attacker could send vote instructions, receive the ballot on the public channel, and leave the process at that point, without executing the end, so that the ballot is never added to the ballot box. This difference is an important one, and is fully intentional: we wanted to model a scenario where the attacker can intercept and block ballots on their way to the ballot box. This gives him more power, and thus makes for a stronger privacy property. A consequence of that choice however, is that our definition is not suited to studying protocols that rely on weeding out duplicate ballots from the ballot box (*e.g.* some fixed versions of Helios). Indeed, the weeding operation only makes sense when assuming that all generated ballots have reached the ballot box – otherwise, some duplicates could be missed, if the original was blocked.

Second, many voting schemes include mechanisms allowing everyone to check that the tallying authority computed the result correctly. Typically, the talliers publish, alongside the result itself, zero-knowledge proofs showing that they *e.g.* correctly decrypted the ballots in the ballot box. In BPRIV however, having them output this proof would immediately break the property. The proof only holds for the actual ballots being tallied, so the attacker could just check it against the ballots he saw, which would succeed on the left but fail on the right. The original formalisation handles this by using a simulator for the proof on the right. This sort of operation does not really have a counterpart in the symbolic model, and we decided (for now) to simply abstract this proof away and not model it.

3.3. Auxiliary properties

In [11], the authors propose two companion properties to BPRIV, called *strong correctness* and *strong consistency*. Together with BPRIV, they imply a strong simulation-based notion of vote privacy. Although we do not prove such a simulation – these are not really used in the symbolic

model – we still define symbolic counterparts to the original computational side-conditions. They are useful when establishing our reduction result, and we will from now on assume they hold.

Strong correctness. Honest voters should always be able to cast their vote, *i.e.* their ballots are always valid. Formally, for any $id, cr, r, u, sk \in \Sigma_0 \cup \mathcal{N}$, $v \in \mathsf{Votes}$, we must have: $\mathsf{Valid}(id, \mathsf{Pub}(cr, u), \mathsf{Vote}(\mathsf{pk}(sk), id, cr, v, r), \mathsf{pk}(sk)) =_{\mathsf{E}} \mathsf{true}$.

Strong consistency. The tally itself should only compute the result of the election, and nothing else – it cannot accept hidden commands from the attacker coded as special ballots, *etc.* Formally we assume two functions $\mathsf{extract}$ and count:

- $\mathsf{extract}(b, sk)$ is meant to extract the vote, and the voter's *id* and credential from b, using key sk, or return $\perp$ if b is not readable (ill-formed, *etc.*).
- count is the counting function, meant to compute the result from the list of votes. It is assumed to always return a public term in $\mathcal{T}(\Sigma, \Sigma_0)$.

We assume that: if $\mathsf{Valid}(id, \mathsf{Pub}(cr, u), b, \mathsf{pk}(sk)) =_{\mathsf{E}} \mathsf{true}$ then $\mathsf{extract}(b, sk) = (id, cr, v)$ for some $v \in \mathsf{Votes}$. In other words, extraction always succeeds on valid ballots. Moreover, $\mathsf{extract}$ must behave as expected on honestly generated ballots, *i.e.* $v = v_0$ when $b = \mathsf{Vote}(\mathsf{pk}(sk), cr, v_0, r)$. We let $\mathsf{extract}([b_1, \ldots, b_n], sk)$ be the list of non-$\perp$ values in $[\mathsf{extract}(b_1, sk), \ldots, \mathsf{extract}(b_n, sk)]$.

Lastly, we assume that these functions characterise the behaviour of the C_{Tally} computation, *i.e.* for all list bb of messages, for all $sk \in \mathcal{N}$, we have:

$$C_{\mathsf{Tally}}(bb, sk) = \mathsf{count}(\mathsf{lst}(\mathsf{extract}(bb, sk)))$$

where lst is a function that only keeps the vote in each tuple returned by $\mathsf{extract}$. Later on, when considering the case of revote, lst will be replaced with a function applying a revoting policy to determine which vote to keep for each voter.

Example 9. *The* Valid *recipe and* C_{tally} *computation from Examples 7 and 8 satisfy these assumptions, where* $\mathsf{extract}$ *simply decrypts the ciphertext in the ballot, and* count *returns the pair of the numbers of votes for* yes *and* no.

4. Reduction result

We first establish our reduction in the case where voters vote only once. Some systems allow voters to vote again by submitting a new ballot that will *e.g.* replace their previous one, in the interest of coercion-resistance. We extend our result to that setting in Section 5. Our BPRIV definition stated in Section 3 is parametrized by the number n of voters among which p are assumed to be honest. We prove our reduction result in two main steps. We first establish that it is enough to consider the case where $p = 1$, *i.e.* one honest voter is enough (see Section 4.3), and then we prove that the number of dishonest voters can be bounded as well (see Section 4.4). Before detailing these two parts, we first formally state our reduction result in Section 4.1, and we give in Section 4.2 a precise characterisation of an attack trace regrding the property BPRIV (when such a trace exists).

4.1. Main result

In order to reduce the number of dishonest voters needed to mount an attack against BPRIV, we need an additional assumption on the counting function used in the e-voting protocol. Roughly, as formally stated below, we have to ensure that when there is a difference in the result when considering n votes, then a difference still exists when considering at most k votes.

Definition 8. *A counting function* count *is* k-bounded *if for all* n*, for all lists* $l_{\mathsf{tally}} = [v_1, \ldots, v_n]$ *and* $l'_{\mathsf{tally}} = [v'_1, \ldots, v'_n]$ *of size* $n > k$ *of elements in* Votes*, such that* $\mathsf{count}(l_{\mathsf{tally}}) \neq_{\mathsf{E}} \mathsf{count}(l'_{\mathsf{tally}})$*, there exist* $k' \leqslant k$*, and* $i_1 < \ldots < i_{k'}$*, such that* $\mathsf{count}([v_{i_1}, \ldots, v_{i_{k'}}]) \neq_{\mathsf{E}} \mathsf{count}([v'_{i_1}, \ldots, v'_{i_{k'}}])$.

This assumption needed to establish our reduction results captures the most common counting functions such as multiset, sum, majority presented below.

Multiset. The result is the multiset of all votes. Formally, in our setting, a term representing that multiset is computed: for all n, $\mathsf{count}_{\#}([v_1, \ldots, v_n]) = f(\{\!|v_1, \ldots, v_n|\!\})$ where f is a function such that $f(M_1) =_{\mathsf{E}} f(M_2)$ (equality on terms) iff $M_1 =_{\#} M_2$ (equality on multisets). For instance, if we just output the list of all votes, the order cannot matter, *i.e.* $\mathsf{count}_{\#}([a, b]) =_{\mathsf{E}} \mathsf{count}_{\#}([b, a])$.

Sum. A total of points total is given to each voter who decides to distribute them among the candidates of his choice. The result is a vector of integers representing the total of points obtained by each candidate. Assuming c candidates, for all n, we have: $\mathsf{count}_{\Sigma}([v_1, \ldots, v_n]) = f(\sum_{i=1}^{n} v_i)$ where $v_i = (p_1, \ldots, p_c)$ with $1 \leqslant i \leqslant n$, and $p_1, \ldots, p_c \in \mathbb{N}$ with $p_1 + \ldots + p_c \leqslant \mathsf{total}$, and f is a function from vectors of c integers to terms such that $f(\overrightarrow{u_1}) =_{\mathsf{E}} f(\overrightarrow{u_2})$ (equality on terms) iff $\overrightarrow{u_1} = \overrightarrow{u_2}$ (equality on vectors of integers).

Majority. The majority function between two choices yes and no simply outputs yes if $\#\mathsf{yes} > n/2$ where n is the number of votes, and no otherwise. For all n, $\mathsf{count}_{\mathsf{Maj}}([v_1, \ldots, v_n]) = \mathsf{yes}$ if $\#\{i \mid v_i = \mathsf{yes}\} > n/2$; and $\mathsf{count}_{\mathsf{Maj}}([v_1, \ldots, v_n]) = \mathsf{no}$ otherwise. Here, yes and no are two public constants ($\mathsf{yes} \neq_{\mathsf{E}} \mathsf{no}$).

Lemma 2. *The functions* $\mathsf{count}_{\#}$*,* count_{Σ}*, and* $\mathsf{count}_{\mathsf{Maj}}$ *are* 1*-bounded.*

Proof. Let $[v_1, \ldots, v_n]$ and $[v'_1, \ldots, v'_n]$ be two lists of votes with $n > 1$, such that $\mathsf{count}_{\#}([v_1, \ldots, v_n]) \neq \mathsf{count}_{\#}([v'_1, \ldots v'_n])$. Since $\mathsf{count}_{\#}$ is a function, we have $\{\!|v_1, \ldots, v_n|\!\} \neq \{\!|v'_1, \ldots, v'_n|\!\}$, and thus there exists i_0 such that $v_{i_0} \neq v'_{i_0}$. Hence, $\mathsf{count}([v_{i_0}]) \neq \mathsf{count}([v'_{i_0}])$, which concludes the proof for $\mathsf{count}_{\#}$. A similar reasoning applies for count_{Σ}, and $\mathsf{count}_{\mathsf{Maj}}$. □

We can now state our main reduction theorem establishing that to study BPRIV, it suffices to consider one honest voter, and at most k dishonest ones, as soon as the counting function is k-bounded.

Theorem 1. *Let* $\mathcal{V}$ *be a voting scheme whose associated counting function is* k*-bounded for some* $k \geqslant 1$*, and* p, n *be two integers such that* $1 \leqslant p \leqslant n$*. If* $\mathcal{V}$ *does not satisfy* $\mathsf{BPRIV}(p, n-p)$*, then* $\mathcal{V}$ *does not satisfies* $\mathsf{BPRIV}(1, k)$*. Moreover, in that case there exists a witness of this attack where no more than* k *ballots reached the ballot box.*

This theorem is an easy consequence of Proposition 2 and Proposition 3 stated and proved in Section 4.3 and in Section 4.4.

Example 10. *The ballot copy attack on Helios (with the 1-bounded multiset count) from [5], mentioned in Example 6, can be performed against* $\mathsf{BPRIV}(p, n-p)$*: a honest voter is told to vote* yes *or* no*, her ballot is copied by a dishonest voter but remains valid, and the result is then* $\{\!|\mathsf{yes}, \mathsf{yes}|\!\}$ *on the left (as the "*yes*" ballot was seen and copied), and* $\{\!|\mathsf{yes}, \mathsf{no}|\!\}$ *on the right (as the "*no*" ballot was seen).*

In accordance with Theorem 1, one honest voter, one dishonest, and one accepted ballot are actually sufficient: the attacker can simply block the honest ballot, so that only the copy is counted leading to $\{\!|\mathsf{yes}|\!\}$ *on the left and* $\{\!|\mathsf{no}|\!\}$ *on the right, which suffices for the attack.*

4.2. Characterisation of an attack trace

In the proofs in the next two sections (i.e. Sections 4.3 and 4.4), we will start with an attack trace on the election process involving n voters, and show that an attack trace still exists considering less (honest) voters. To ease the proofs of these reduction results, we start by giving a precise characterisation of an attack trace (when such a trace exists). This characterisation is stated in Proposition 1. We first show that the election processes we study are action-deterministic.

Lemma 3. *The two ground processes* $\mathsf{Election}^{\mathsf{L}}_{p,n-p}(\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n})$ *and* $\mathsf{Election}^{\mathsf{R}}_{p,n-p}(\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n})$ *are action-deterministic for any* n*, and any* $p \leqslant n$*.*

Proof. For these two processes, until phase 3, each process in parallel has its own public dedicated channel. Thus, the action mentioned on the trace tr indicates which action will be triggered, there is no ambiguity, and it is therefore clear that the resulting frames are equal up to α-renaming.

Now, when reaching phase 3, the process Tally is a computation process that may involved private channels, and thus leads to non-determinism. However, by definition of a computation process, we know that this process will result on a unique output on the public channel c_r, and the value of this output only depends on the parameters given to the computation process, here sk and the content of m_{bb}. The content of m_{bb} is entirely determined by tr and the content of the frame. When considering the same trace tr, we obtain frame which are equal up to α-renaming, and we will obtain the same public term for the tally. □

We can now show that when considering an attack trace tr, *i.e.* a witness of non-inclusion between two election processes, the attack trace can be considered w.l.o.g. to be Σ_{err}-free. That is, tr does not contain any occurrence of c_{err} for any $c_{\mathsf{err}} \in \Sigma_{\mathsf{err}}$. We can also assume that the non-equivalence comes from static non-equivalence, and that inputs in phase 2 are messages representing valid voting options.

Proposition 1. *Let* $\mathcal{V}$ *be a voting scheme such that*

$$\mathsf{Election}^{\mathsf{L}}_{p,n-p}(\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n}) \not\approx_t \mathsf{Election}^{\mathsf{R}}_{p,n-p}(\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n}).$$

Let tr *be a witness of this non-equivalence of minimal length. Then* tr *is such that:*

- $\mathsf{Election}^{\mathsf{L}}_{p,n-p}(\overrightarrow{v_1},\overrightarrow{v_2},\ldots,\overrightarrow{v_n}) \stackrel{\mathsf{tr}}{\Longrightarrow} (i_L;\mathcal{P}_L;\phi_L;M_L)$ *for some* $(i_L;\mathcal{P}_L;\phi_L;M_L)$*;*
- $\mathsf{Election}^{\mathsf{R}}_{p,n-p}(\overrightarrow{v_1},\overrightarrow{v_2},\ldots,\overrightarrow{v_n}) \stackrel{\mathsf{tr}}{\Longrightarrow} (i_R;\mathcal{P}_R;\phi_R;M_R)$ *for some* $(i_R;\mathcal{P}_R;\phi_R;M_R)$*;*
- $i_L = i_R$*,* $\phi_L \not\sim \phi_R$*, and* tr *is* Σ_{err}*-free.*

Moreover, for any $i \in \{1,\ldots,p\}$*, if* $\mathsf{in}(c_i,R)$ *occurrs in* tr *in phase 2 (for some* R*), then there exists* $(v_0,v_1) \in \mathsf{Votes}\times\mathsf{Votes}$ *such that* $R\phi_L =_{\mathsf{E}} R\phi_R =_{\mathsf{E}} (v_0,v_1)$*.*

Proof. Assume first that the minimal witness of this non-equivalence is actually a witness for the following non-inclusion: $\mathsf{Election}^{\mathsf{L}}_{p,n-p}(\overrightarrow{v_1},\overrightarrow{v_2},\ldots,\overrightarrow{v_n}) \not\sqsubseteq_t \mathsf{Election}^{\mathsf{R}}_{p,n-p}(\overrightarrow{v_1},\overrightarrow{v_2},\ldots,\overrightarrow{v_n})$. As the processes under consideration are action-deterministic (Lemma 3), this witness is a trace tr such that $\mathsf{Election}^{\mathsf{L}}_{p,n-p}(\overrightarrow{v_1},\ldots,\overrightarrow{v_n}) \stackrel{\mathsf{tr}}{\Longrightarrow} (i_{\mathsf{L}};\mathcal{P}_{\mathsf{L}};\phi_{\mathsf{L}};M_{\mathsf{L}})$, and for which:

(1) there does not exist $(i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$ such that $\mathsf{Election}^{\mathsf{R}}_{p,n-p}(\overrightarrow{v_1},\ldots,\overrightarrow{v_n}) \stackrel{\mathsf{tr}}{\Longrightarrow} (i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$; or
(2) such a trace exists, *i.e.* $\mathsf{Election}^{\mathsf{R}}_{p,n-p}(\overrightarrow{v_1},\ldots,\overrightarrow{v_n}) \stackrel{\mathsf{tr}}{\Longrightarrow} (i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$ but $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$ (note that we necessarily have that $i_{\mathsf{L}} = i_{\mathsf{R}}$).

We first assume that such a witness of minimal length satsifies the requirements stated in item 1, *i.e.* there does not exist $(i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$ such that $\mathsf{Election}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0},\overrightarrow{v_1},\ldots,\overrightarrow{v_n}) \stackrel{\mathsf{tr}}{\Longrightarrow} (i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$. Note that it means that, at some point, the outcome of a test is not the same on both sides, and this leads to an output that can not be mimicked on the other side. When the test under consideration is public (*i.e.* corresponds to a computation that can be performed by the attacker), we get a contradiction since the trace tr without its last output will already lead to a witness of non-inclusion. The only remaining case is the validity test performed by the honest voter but here we know that such a test can not failed. Indeed, we have assumed strong correctness, *i.e.*:

$$\mathsf{Valid}(id,\mathsf{Pub}(cr,u),\mathsf{Vote}(\mathsf{pk}(sk),id,cr,v,r),\mathsf{pk}(sk)) =_{\mathsf{E}} \mathsf{true}.$$

Therefore, we know that such a minimal witness is due to a problem regarding static equivalence: there exists $(i_{\mathsf{L}};\mathcal{P}_{\mathsf{L}};\phi_{\mathsf{L}};M_{\mathsf{L}})$ such that $\mathsf{Election}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0},\overrightarrow{v_1},\ldots,\overrightarrow{v_n}) \stackrel{\mathsf{tr}}{\Longrightarrow} (i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$ but $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$.

It remains to establish that tr can be considered to be Σ_{err}-free. Assume that tr contains an action of the form $\mathsf{out}(c_i,c_{\mathsf{err}})$ for some c_i and some $c_{\mathsf{err}} \in \Sigma_{\mathsf{err}}$. Then, the trace tr' without this action still passes on both sides, and leads to the exact same frames. Indeed, in the processes considered, the errors are always placed at the end of a branch, and hence not executing them does not change the remaining trace. Therefore such an action can not occur in a minimal witness.

Finally, for any honest voter i, if $\mathsf{in}(c_i,R)$ occurs in tr in phase 2, it must be that the test "$\mathsf{if}\ v^0,v^1 \in \mathsf{Votes}$" succeeds on the left and eventually the corresponding output is performed, or the test fails on the left and eventually an error message is output. In the first case, there exist $(v_0,v_1) \in \mathsf{Votes}^2$ such that $R\phi_L =_{\mathsf{E}} (v_0,v_1)$, and thus by minimality of the witness $R\phi_R =_{\mathsf{E}} (v_0,v_1)$. In the second case, we have $R\phi_L \neq_{\mathsf{E}} (v_0,v_1)$ for any $(v_0,v_1) \in \mathsf{Votes}^2$, and, again by minimality of the witness, $R\phi_R \neq_{\mathsf{E}} (v_0,v_1)$ for any (v_0,v_1). Since tr is Σ_{err}-free, we know that the corresponding error message is not output in the trace, but in this case, by minimality of tr, we know that this input is not useful to get a witness of non-equivalence. □

4.3. Reduction to one honest voter

When designing symbolic definitions that formalise security properties, even when an arbitrary number of participants are involved, a common modelling choice is to particularise the definition on a small number of honest agents, on which the property should hold. For instance, a key agreement property is often formalised by requiring that two fixed (but arbitrary) honest agents agree on the key at the end of their session, even in presence of arbitrarily many dishonest agents. A more general definition would require that the same holds for any number of honest agents running the protocol in parallel, so that any two honest agent agree on a key once they finish a session together. The choice of fixing the honest agents when formalising the property produces a simpler property, with less honest sessions in parallel, which is usually easier on the automated tools. It is usually justified by arguing (more or less formally) that it implies the more general version: given an arbitrary number of honest agents, for any pair of agents, we can see from their point of view all other agents as potentially corrupted, and thus the simpler property applies and shows they agree.

A similar choice is implicitly made when considering the swapping definition for vote privacy. Indeed, the more general version would require that two scenarios, where the votes of any number of honest voters have been permuted, are always indistinguishable. This general formulation would in fact be closer to the one used in the computational setting. In contrast, the symbolic swapping definition consider two particular honest voters Alice and Bob, whose votes are exchanged. To justify this choice, it could be argued that, as any permutation can be decomposed in a finite sequence of swaps of two elements, by applying the seemingly weaker property as many times as needed, we can recover the general version. This argument is however not often formalised.

In order to remain faithful to the original computational BPRIV notion, and to define a strong privacy property, we decided to write our symbolic BPRIV property in a general way, *i.e.* considering an arbitrary number of honest voters. Each voter receives two vote instructions (v_0, v_1) from the attacker, and shows him the ballot for one or the other. Reducing the number of honest voters by replacing them with dishonest ones is non trivial as the behaviour of an honest voter can not be mimicked by a dishonest one, or simply compensated by some steps performed by the attacker. This comes from the fact the behaviour of an honest voter is *not* exactly the same on both sides of the equivalence, as it is the case for a dishonest voter. Nevertheless, we establish the following result: one honest voter is enough.

Proposition 2. *Consider a voting scheme $\mathcal{V}$, and p, n such that $1 \leqslant p \leqslant n$. If $\mathcal{V}$ does not satisfy* $\mathsf{BPRIV}(p, n-p)$*, then it does not satisfy* $\mathsf{BPRIV}(1, n-1)$*.*

The general idea of this proof is to show we can isolate one specific honest voter whose ballot is the one causing $\mathsf{BPRIV}(p, n-p)$ to break. We then leave that voter as the only honest one, and use dishonest voters to simulate the $p-1$ others, and obtain an attack against $\mathsf{BPRIV}(1, n-1)$.

The difficulties are *(i)* how to find this particular voter, and *(ii)* how to simulate the honest voters with dishonest ones. The simulation would be easy for a honest voter *id* voting for the same candidate v on both sides: simply use the dishonest voter to submit a ballot $\mathsf{Vote}(pk, id, cr, v, r)$ for some random r, and the correct credential cr. However, in the Election processes, *id* uses different values v_0, v_1 on the left and on the right, so that we cannot easily construct a single dishonest ballot simulating *id*'s on both sides at the same time.

To solve both issues, the main idea is to go gradually from the $\mathsf{Election}^{\mathsf{L}}$ process, where all HVoters are $\mathsf{HVoter}^{\mathsf{L}}$ and use the real vote (their v_0), to the $\mathsf{Election}^{\mathsf{R}}$ process, where they are $\mathsf{HVoter}^{\mathsf{R}}$ and

We fix n distinct voters $\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}$, with for all i $\overrightarrow{v_i} = (c_i, id_i, cr_i, pcr_i)$, $pcr_i = \mathsf{Pub}(cr_i, u_i)$, and $p \in \{1, \ldots, n\}$. For any $i \in \{0, \ldots, p\}$, we define:

$$
\begin{aligned}
P_i = {} & \mathsf{new}\ sk.\ m_{\mathsf{bb}} := \mathsf{nil}.\ \mathsf{out}(ch, \mathsf{pk}(sk)).\\
& (\ \mathsf{phase}\ 1.\mathsf{out}(c_1, pcr_1).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{R}}(\overrightarrow{v_1}, \mathsf{pk}(sk))\\
& \mid \ldots\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_i, pcr_i).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{R}}(\overrightarrow{v_i}, \mathsf{pk}(sk))\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_{i+1}, pcr_{i+1}).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{L}}(\overrightarrow{v_{i+1}}, \mathsf{pk}(sk))\\
& \mid \ldots\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_p, pcr_p).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{L}}(\overrightarrow{v_p}, \mathsf{pk}(sk))\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_{p+1}, \langle cr_{p+1}, pcr_{p+1}\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v_{p+1}}, \mathsf{pk}(sk))\\
& \mid \ldots\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_n, \langle cr_n, pcr_n\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v_n}, \mathsf{pk}(sk))\\
& \mid \mathsf{phase}\ 3.\ \mathsf{Tally}(sk)\ )
\end{aligned}
$$

Figure 2. Proof of Proposition 2 - Intermediate processes P_i

use the fake one (their v_1). We consider intermediate processes $P_0, \ldots, P_p$: as displayed in Figure 2, in P_i, the first i HVoters are $\mathsf{HVoter}^{\mathsf{R}}$, and the others are $\mathsf{HVoter}^{\mathsf{L}}$. Since $\mathsf{BPRIV}(p, n-p)$ does not hold, $P_0 = \mathsf{Election}^{\mathsf{L}}$ and $P_p = \mathsf{Election}^{\mathsf{R}}$ are not equivalent. Hence, there must exist some i_0 such that P_{i_0+1} and P_{i_0} are not equivalent. These two processes differ only by the $i_0 + 1^{\text{th}}$ HVoter, who is $\mathsf{HVoter}^{\mathsf{L}}$ in P_{i_0}, and $\mathsf{HVoter}^{\mathsf{R}}$ in P_{i_0+1}. This voter will be our particular voter, who will remain honest, solving issue *(i)*. All other HVoters behave the same in P_{i_0} and P_{i_0+1}: they vote with their right vote for the first i_0, and their left for the last $p - i_0 - 1$. For them, issue *(ii)* is thus solved, and we can simulate them with dishonest voters. This way, we recover an attack with only one honest voter, and $(n-p) + (p-1) = n-1$ dishonest voters.

Note that, in the case of the earlier reduction result from [6] for the SWAP definition, a simple version of vote privacy is used from the start. They consider only two honest voters who swap their votes, and not the general definition (as stated *e.g.* in [10, 11]) involving an arbitrary permutation between an arbitrary number of honest voters. Due to this, in [6], this first step was trivial. The argument in our case is more involved, as we start from the general notion.

Before proving the reduction result, let us first observe that since the Valid recipe and the C_{Tally} computation process do not use any private names, and always return public values, their output cannot depend on the random values used in the ballots/credentials. More precisely, these random values can be renamed and/or replaced with public fresh names without changing the outcome of Valid or C_{Tally}. This property, which we will refer to as *randomness independence*, is a direct consequence of the construction of terms and semantics of processes in our symbolic model. We will use it in the proof of the reduction theorem, and for this reason we state it formally below.

Lemma 4. *Consider a key $sk \in \mathcal{N}$, with the associated $pk = \mathsf{pk}(sk)$, and n distinct voters $id_1, \ldots, id_p, id_{p+1}, \ldots, id_n \in \Sigma_0$, meant to represent p honest voters and $n-p$ dishonest ones, each*

with their credential $cr_i \in \mathcal{N}$. Let ϕ_0 denote the frame of public keys and credentials

$$\phi_0 = \{\ w_0 \mapsto pk, w_1 \mapsto \mathsf{Pub}(cr_1, u_1), \ldots, w_p \mapsto \mathsf{Pub}(cr_p, u_p)),\\ w_{p+1} \mapsto \langle cr_{p+1}, \mathsf{Pub}(cr_{p+1}, u_{p+1})\rangle, \ldots, w_n \mapsto \langle cr_{p+1}, \mathsf{Pub}(cr_{p+1}, u_{p+1})\rangle\}.$$

Consider a frame ϕ_1 of m ballots, honestly generated by honest voters $id_{i_1}, \ldots, id_{i_m}$ (two ballots can potentially be generated by the same voter):

$$\phi_1 = \{\mathsf{w}'_1 \mapsto \mathsf{Vote}(pk, id_{i_1}, cr_{i_1}, v_1, r_1), \ldots, \mathsf{w}'_m \mapsto \mathsf{Vote}(pk, id_{i_m}, cr_{i_m}, v_m, r_m)\}$$

with votes $v_1, \ldots, v_n \in \mathsf{Votes}$, using distinct random values $r_1, \ldots, r_m \in \mathcal{N} \setminus \{sk, u_1, \ldots, u_n\}$. Let ϕ denote $\phi_0 \cup \phi_1$. Consider recipes R_1, R_2, R_3, R_4 on $\mathsf{dom}(\phi)$. Also consider an arbitrary injective renaming $\sigma : \{r_1, \ldots, r_m, u_1, \ldots, u_m\} \to \Sigma_0 \cup \mathcal{N} \setminus \{sk\}$, such that for any r in its domain, $\sigma(r)$ does not appear in any $R_1, R_2, R_3, R_4, \mathsf{Valid}, C_{\mathsf{Tally}}$. Then we have:

- $\mathsf{Valid}(R_1\phi, R_2\phi, R_3\phi, pk) =_{\mathsf{E}} \mathsf{true} \Leftrightarrow \mathsf{Valid}(R_1\phi\sigma, R_2\phi\sigma, R_3\phi\sigma, pk) =_{\mathsf{E}} \mathsf{true}$*; and*
- $C_{\mathsf{tally}}(R_4\phi, sk) =_{\mathsf{E}} C_{\mathsf{tally}}(R_4\phi\sigma, sk)$.

We can now recall and give a detailed proof of Proposition 2.

Proposition 2. *Consider a voting scheme $\mathcal{V}$, and p, n such that $1 \leqslant p \leqslant n$. If $\mathcal{V}$ does not satisfy $\mathsf{BPRIV}(p, n-p)$, then it does not satisfy $\mathsf{BPRIV}(1, n-1)$.*

Proof. We will show that under our assumptions we have $P_i \approx_t P_{i+1}$ for any $i \in \{0, \ldots, p-1\}$ where P_i are the processes displayed in Figure 2. Since $P_0 = \mathsf{ElectionL}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$ and $P_p = \mathsf{ElectionR}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$, by transitivity of $\approx_t$, this property suffices to prove the theorem.

Fix some index $i \in \{0, \ldots, p-1\}$. Observe that P_i and P_{i+1} differ only in the behaviour of the $(i+1)^{\text{th}}$ voter id_{i+1}, which is modelled by the process $\mathsf{HVoterL}(\overrightarrow{v_{i+1}}, \mathsf{pk}(sk))$ in process P_i, and by the process $\mathsf{HVoterR}(\overrightarrow{v_{i+1}}, \mathsf{pk}(sk))$ in P_{i+1}. All other honest voters are identical in P_i and P_{i+1}: they always follow the attacker's instructions in the same way, either always voting for the right vote (for voters id_j, $j \leqslant i$) or the left vote (for voters id_j, $j \geqslant i+2$). Therefore, the main idea of the proof is that all these other voters can be simulated by the attacker, since their behaviour is known and the same on both sides. The only remaining honest voter will be id_{i+1}, to which we will apply the assumption that BPRIV holds for one honest voter.

To prepare the terrain for applying this assumption later on, we define two additional processes Q_{L}, Q_{R}, where this "simulation" is performed, *i.e.* where all voters except id_{i+1} are controlled by the attacker. Formally, the processes for these voters are replaced by instances of process DVoter.

The process Q_{X} with $\mathsf{X} \in \{\mathsf{L}, \mathsf{R}\}$ is as follows:

$$
\begin{aligned}
Q_{\mathsf{X}} = \ & \mathsf{new}\ sk.\ m_{\mathsf{bb}} := \mathsf{nil}.\ \mathsf{out}(ch, \mathsf{pk}(sk)).\\
& (\ \mathsf{phase}\ 1.\mathsf{out}(c_1, \langle cr_1, pcr_1\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v_1}, \mathsf{pk}(sk))\\
& \mid \ldots\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_i, \langle cr_i, pcr_i\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v_i}, \mathsf{pk}(sk))\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_{i+1}, pcr_{i+1}).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^X(\overrightarrow{v_{i+1}}, \mathsf{pk}(sk))\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_{i+2}, \langle cr_{i+2}, pcr_{i+2}\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v_{i+2}}, \mathsf{pk}(sk))\\
& \mid \ldots\\
& \mid \mathsf{phase}\ 1.\mathsf{out}(c_n, \langle cr_n, pcr_n\rangle).\ \mathsf{phase}\ 2.\ \mathsf{DVoter}(\overrightarrow{v_n}, \mathsf{pk}(sk))\\
& \mid \mathsf{phase}\ 3.\ \mathsf{Tally}(sk)\ )
\end{aligned}
$$

In fact, up to permutation of the parallel branches, these two processes are instances of the generic election process, with one honest voter (id_{i+1}) and $n-1$ dishonest voters ($id_j, j \neq i+1$):

$$Q_{\mathsf{X}} = \mathsf{ElectionX}_{1,n-1}(\overrightarrow{v_{i+1}}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_i}, \overrightarrow{v_{i+2}}, \ldots, \overrightarrow{v_n})$$

Thanks to the assumption that BPRIV holds for one honest voter, we have $Q_{\mathsf{L}} \approx_t Q_{\mathsf{R}}$.

By contradiction, let us now assume that $P_i \not\approx_t P_{i+1}$. Using Lemma 3, P_i, P_{i+1}, Q_{L}, Q_{R} are action-determinate. Let tr be a witness of this non-equivalence of minimal length. Thanks to Proposition 1, tr is such that:

- $P_i \overset{\mathsf{tr}}{\Longrightarrow} (i; \mathcal{P}_{\mathsf{L}}; \phi_{\mathsf{L}}; M_{\mathsf{L}})$ for some $i, \mathcal{P}_{\mathsf{L}}, \phi_{\mathsf{L}}, M_{\mathsf{L}}$;
- $P_{i+1} \overset{\mathsf{tr}}{\Longrightarrow} (i; \mathcal{P}_{\mathsf{R}}; \phi_{\mathsf{R}}; M_{\mathsf{R}})$ for some $\mathcal{P}_{\mathsf{R}}, \phi_{\mathsf{R}}, M_{\mathsf{R}}$;
- $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$, and tr is Σ_{err}-free.

Moreover, for any $j \in \{1, p\}$, if $\mathsf{in}(c_j, R)$ occurs in tr in phase 2 (for some R), then there exist $(v_0, v_1) \in \mathsf{Votes} \times \mathsf{Votes}$ such that $R\phi_{\mathsf{L}} = R\phi_{\mathsf{R}} =_{\mathsf{E}} (v_0, v_1)$. When such an input exists, let $\mathsf{instr}(j)$ denote this pair of votes, which is the instruction given by the attacker to voter j in tr.

In addition, by action-determinacy, ϕ_{L} and ϕ_{R} are unique up to α-renaming of fresh names – without loss of generality, let us assume that the same symbols are used for matching private fresh names in both frames, *i.e.* the random values used for constructing a honest ballot on either side are given the same name, and similarly for the election key.

Our next step is to construct a sequence of actions $\overline{\mathsf{tr}}$, that describes how to simulate the execution tr of P_i (resp. P_{i+1}) in an execution of Q_{L} (resp. Q_{R}).

Intuitively, the attacker interacting with Q_{L} or Q_{R} performs the same actions as the original one interacting with P_i or P_{i+1}, except that all honest voters but id_{i+1} are simulated using dishonest voters. Hence, whenever the attacker (for P_i, P_{i+1}) provides two votes (v_0, v_1) to an honest voter id_j (with $1 \leqslant j \leqslant p$ and $j \neq i+1$), we instead let the attacker (for Q_{L}, Q_{R}) construct the corresponding ballot $\mathsf{Vote}(pk, id_j, cr_j, v_0, r_0)$ and provide it to the process for id_j, who is now dishonest. Note that, since the result computed in the end by the tally always counts the "left" vote v_0, we must construct the ballot containing that vote, so that the result obtained in the end is the right one.

A subtle detail is that when constructing this ballot, the attacker will not be able to use the same private name r_0 originally used by the honest voter in tr. He must instead use a public name.

To keep notations relatively light, we introduce, for each private name r generated by the process for an honest voter other than id_{i+1} in P_i or P_{i+1} an associated public name, that the attacker may use instead, which we will call $\tilde{r}$. This name must be fresh, *i.e.* not appear in any of the processes or recipes considered until now (including those used in the inputs in tr). We also let σ denote the function mapping each such public $\tilde{r}$ to the corresponding private r.

Due to the form of the processes, we can assume w.l.o.g. that tr is a prefix of:

$$\mathsf{out}(ch, \mathsf{w}_0).\mathsf{phase}\,1.\mathsf{out}(c_{i_1}, \mathsf{w}_{i_1}).\ldots.\mathsf{out}(c_{i_p}, \mathsf{w}_{i_p}).\mathsf{phase}\,2.\mathsf{tr}_{\mathsf{cast}}.\mathsf{phase}\,3.\mathsf{out}(c_{res}, \mathsf{w}_{\mathsf{tall}})$$

where $\mathsf{tr}_{\mathsf{cast}}$ contains only inputs and outputs on the channels $\{c_i\}_{1\leqslant i\leqslant n}$, with at most one input on each c_i, and, when this input is present, at most one output on c_i, placed after the input. Without loss of generality, call R_i the recipe provided in the input on c_i in $\mathsf{tr}_{\mathsf{cast}}$, and w'_i the frame variable recording the output on c_i (if they exist).

We now define recipes that we will use to let the attacker compute ballots for honest voters simulated by dishonest ones. For any $j \in [\![1, p]\!]$ with $j \neq i+1$ such that an input $\mathsf{in}(c_j, R_j)$ occurs in $\mathsf{tr}_{\mathsf{cast}}$, we let $B^0_j = \mathsf{Vote}(\mathsf{w}_0, id_j, \mathsf{proj}^2_1(\mathsf{w}_j), v_0, \tilde{r}_0)$ and $B^1_j = \mathsf{Vote}(\mathsf{w}_0, id_j, \mathsf{proj}^2_1(\mathsf{w}_j), v_1, \tilde{r}_1)$ where $(v_0, v_1) = \mathsf{instr}(j)$ and $\tilde{r}_0$, $\tilde{r}_1$ are fresh public names associated by σ to the private names r_0, r_1 used to construct the ballots for voter j in P_i and P_{i+1}.

Let $\overline{\mathsf{tr}}$ be the trace containing the same actions as tr, except that in $\mathsf{tr}_{\mathsf{cast}}$ (if tr reaches $\mathsf{tr}_{\mathsf{cast}}$),

- any input $\mathsf{in}(c_j, R_j)$ for $1 \leqslant j \leqslant p, j \neq i+1$, *i.e.* the input of the attacker's instructions for honest voter j, is replaced with $\mathsf{in}(c_j, B^0_j)$.
- any input $\mathsf{in}(c_j, R_j)$ for $j > p$, *i.e.* the attacker's instruction for dishonest voter j, os replaced with $\mathsf{in}(c_j, S_j)$, where

$$S_j = R_j\{\mathsf{w}'_k \mapsto B^1_k\}_{1\leqslant k\leqslant i}\{\mathsf{w}'_k \mapsto B^0_k\}_{i+1<k\leqslant p}.$$

By construction of $\bar{t}$, and from the shape of the processes Q_{L}, Q_{R}, it is clear that $\bar{t}$ is executable in Q_{L} and Q_{R}. All inputs and outputs in phases 0, 1, and 3 can be performed as expected. There are only two points where $\bar{t}$ might *a priori* be non-executable in phase 2, that are related to the validity checks:

- If the validity check in a DVoter process for a voter id_j with $j > p$ failed, preventing an output on c_j that was possible in tr: by construction, the ballot b' on which the validity check fails in $\overline{\mathsf{tr}}$ and the ballot b output by this voter in tr, on which the test succeeds, are obtained by the same recipe applied to two frames of honest ballots that differ only on the random values used (the $\tilde{r}$ or the r). By the randomness independence property (Lemma 4), this is not possible.
- If the validity check in a DVoter process for a voter id_j with $j \leqslant p$ failed, preventing an output on c_j that was possible in tr: by the consistency assumption (Section 3.3), validity tests always succeed on honestly generated ballots, and this is not possible.

Executing $\overline{\mathsf{tr}}$ in Q_{L} and Q_{R} respectively produces frames $\overline{\phi}_{\mathsf{L}}$, $\overline{\phi}_{\mathsf{R}}$. By action-determinacy, they are unique up to α-renaming fresh names – without loss of generality, let us assume that the same symbols are used for matching private fresh names in both frames, *i.e.* the random values used for constructing a honest ballot on either side are given the same name, and similarly for the election key. In addition, we will also assume these symbols are the same as for the corresponding names in ϕ_{L}, ϕ_{R}.

Note that, by construction, the recipes B_j^0, B_j^1 from earlier, when applied to $\overline{\phi}_{\mathsf{L}}$ and $\overline{\phi}_{\mathsf{R}}$, compute ballots b_0, b_1 such that $b_0\sigma$ and $b_1\sigma$ are the two ballots computed by honest voter j in tr in P_i and P_{i+1} respectively. Similarly, the recipe S_j used in $\overline{\mathsf{tr}}$ to compute dishonest ballots produces, when applied to $\overline{\phi}_{\mathsf{L}}$ and $\overline{\phi}_{\mathsf{R}}$, a ballot b such that $b\sigma$ is the ballot provided by the attacker to dishonest voter j in tr in P_i and P_{i+1} respectively.

The last step of our proof will be to describe the relation between $\overline{\phi}_{\mathsf{L}}$, $\overline{\phi}_{\mathsf{R}}$, and ϕ_{L}, ϕ_{R}. As we will see, this will bring out a contradiction, as the first two are assumed statically equivalent and the other two are not.

We construct a frame of recipes R, giving for each variable $w \in \mathsf{dom}(\phi_{\mathsf{L}}) = \mathsf{dom}(\phi_{\mathsf{R}})$ a recipe $R(w)$ with variables in $\mathsf{dom}(\overline{\phi}_{\mathsf{L}}) = \mathsf{dom}(\overline{\phi}_{\mathsf{R}})$, such that $\phi_{\mathsf{L}} = (R\overline{\phi}_{\mathsf{L}})\sigma$ and $\phi_{\mathsf{R}} = (R\overline{\phi}_{\mathsf{R}})\sigma$, *i.e.*

$$\forall w \in \mathsf{dom}(\phi_{\mathsf{L}}).\ \phi_{\mathsf{L}}(w) = (R(w)\overline{\phi}_{\mathsf{L}})\sigma \ \wedge\ \phi_{\mathsf{R}}(w) = (R(w)\overline{\phi}_{\mathsf{R}})\sigma \tag{1}$$

R is constructed as follows:

- For w_0, storing the election key output in phase 0: this output is also performed in tr, and $R(\mathsf{w}_0) = \mathsf{w}_0$ is adequate.

- For all w_j present in $\mathsf{dom}(\phi_{\mathsf{L}})$, storing credentials output in phase 1:
  - if $j = i+1$, ϕ_{L} and ϕ_{R} as well as $\overline{\phi}_{\mathsf{L}}$, $\overline{\phi}_{\mathsf{R}}$ contain the public credential pcr_j in w_j, and thus $R(\mathsf{w}_j) = \mathsf{w}_j$ works;
  - if $1 \leqslant j \leqslant p$ and $j \neq i+1$, ϕ_{L} and ϕ_{R} contain the public credential pcr_j in w_j, while $\overline{\phi}_{\mathsf{L}}$ and $\overline{\phi}_{\mathsf{R}}$ contain $\langle cr_j, pcr_j\rangle$; thus $R(\mathsf{w}_j) = \mathsf{proj}_2^2(\mathsf{w}_j)$ works;
  - if $j > p$, ϕ_{L} and ϕ_{R} as well as $\overline{\phi}_{\mathsf{L}}$, $\overline{\phi}_{\mathsf{R}}$ contain the credentials $\langle cr_j, pcr_j\rangle$ in w_j, and thus $R(\mathsf{w}_j) = \mathsf{w}_j$ works.

- For all w'_j present in $\mathsf{dom}(\phi_{\mathsf{L}})$, storing all ballots output during phase 2:
  - if $j < i+1$, according to the processes, ϕ_{L} and ϕ_{R} contain in w'_j the ballot $\mathsf{Vote}(pk, id_j, cr_j, v_1, r_1)$, where $(v_0, v_1) = \mathsf{instr}(j)$, and r_1 is the nonce generated by the voter. Thus $R(\mathsf{w}'_j) = B_j^1$ is adequate.
  - if $j = i+1$, according to the processes, ϕ_{L} as well as $\overline{\phi}_{\mathsf{L}}$ contain in w'_j the ballot $\mathsf{Vote}(pk, id_j, cr_j, v_0, r_0)$, while ϕ_{R} and $\overline{\phi}_{\mathsf{R}}$ contain the ballot $\mathsf{Vote}(pk, id_j, cr_j, v_1, r_1)$, where $(v_0, v_1) = \mathsf{instr}(j)$, and r_0, r_1 the random values used. Thus $R(\mathsf{w}'_j) = \mathsf{w}'_j$ is appropriate.
  - if $i+1 < j \leqslant p$, according to the processes, ϕ_{L} and ϕ_{R} contain $\mathsf{Vote}(pk, id_j, cr_j, v_0, r_0)$ in w'_j, where $(v_0, v_1) = \mathsf{instr}(j)$ and r_0 is the nonce generated by the voter. Thus $R(\mathsf{w}'_j) = B_j^0$ is adequate.

* if $j > p$, according to the processes, ϕ_{L}, ϕ_{R}, $\overline{\phi}_{\mathsf{L}}$, $\overline{\phi}_{\mathsf{R}}$ each contain in w'_j the ballot received as an input from the attacker earlier by voter j's process. As explained earlier, the recipe used in $\overline{\mathsf{tr}}$ to construct that input is such that this ballot verifies $\overline{\phi}_{\mathsf{L}}(\mathsf{w}'_j)\sigma = \phi_{\mathsf{L}}(\mathsf{w}'_j)$ and $\overline{\phi}_{\mathsf{R}}(\mathsf{w}'_j)\sigma = \phi_{\mathsf{R}}(\mathsf{w}'_j)$. Hence, picking $R(\mathsf{w}'_j) = \mathsf{w}'_j$ satisfies (1).

- Finally, the only remaining variable is $\mathsf{w}_{\mathsf{tall}}$, storing the result output in phase 3. Our argument is that the tally actually outputs the same result in the execution of tr in P_i and $\overline{\mathsf{tr}}$ in Q_{L}, and similarly for P_{i+1} and Q_{R}. Indeed, consider the inputs received by Tally on the private channel containing the internal state. In P_i and tr, these are the "left" ballots computed by all honest voters, and the dishonest ballots. In Q_{L} and $\overline{\mathsf{tr}}$, they are

  * the left ballot of voter $i+1$
  * the ballots given as input to dishonest voters $j \in [\![1, p]\!]$ computed using B^0_j, which, as explained earlier, are the left ballots of the original honest voters where r_0 is replaced with r'_0
  * the ballots given as input to dishonest voters $j > p$, computed using RR_j, which, as explained earlier, are computed in the same way as the ballots of the original dishonest voters, from the list of honest ballots where all random values r are replaced with the corresponding $\tilde{r}$.

  Hence, the randomness-independence property (Lemma 4) applies, and guarantees that tallying the ballots in P_i with tr, and in Q_{L} with $\overline{\mathsf{tr}}$ produces the same result. The same argument applies to P_{i+1} and Q_{R}. Thus, $R(\mathsf{w}_{\mathsf{tall}}) = \mathsf{w}_{\mathsf{tall}}$ satisfies (1).

Using property (1), we can now conclude the proof. Indeed, we know that $Q_{\mathsf{L}} \approx_t Q_{\mathsf{R}}$, which, applied to $\overline{\mathsf{tr}}$, implies that that $\overline{\phi}_{\mathsf{L}} \sim \overline{\phi}_{\mathsf{R}}$. Since R is a frame of recipes, it follows immediately from the definition of static equivalence that $R\overline{\phi}_{\mathsf{L}} \sim R\overline{\phi}_{\mathsf{R}}$.

On the other hand, tr was obtained as a non-equivalence witness for P_i and P_{i+1}, meaning that $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$. Thus there exist recipes M, N such that $M\phi_{\mathsf{L}} = N\phi_{\mathsf{L}}$ and $M\phi_{\mathsf{R}} \neq N\phi_{\mathsf{R}}$, *i.e.*

$$M\big((R\overline{\phi}_{\mathsf{L}})\sigma\big) = N\big((R\overline{\phi}_{\mathsf{L}})\sigma\big) \quad \text{and} \quad M\big((R\overline{\phi}_{\mathsf{R}})\sigma\big) \neq N\big((R\overline{\phi}_{\mathsf{R}})\sigma\big).$$

Since none of the public names r' appear in ϕ_{L} or ϕ_{R}, we may always w.l.o.g. choose M and N that do not contain these names either. We then have

$$\big(M(R\overline{\phi}_{\mathsf{L}})\big)\sigma = \big(N(R\overline{\phi}_{\mathsf{L}})\big)\sigma \quad \text{and} \quad \big(M(R\overline{\phi}_{\mathsf{R}})\sigma\big) \neq \big(N(R\overline{\phi}_{\mathsf{R}})\sigma\big).$$

Since σ is a bijective renaming, this means

$$M(R\overline{\phi}_{\mathsf{L}}) = N(R\overline{\phi}_{\mathsf{L}}) \quad \text{and} \quad M(R\overline{\phi}_{\mathsf{R}}) \neq N(R\overline{\phi}_{\mathsf{R}}),$$

i.e. $MR \stackrel{?}{=} NR$ is a test distinguishing $\overline{\phi}_{\mathsf{L}}$ and $\overline{\phi}_{\mathsf{R}}$. This contradicts the fact $Q_{\mathsf{L}} \approx_t Q_{\mathsf{R}}$. Therefore, our assumption was false, *i.e.* $P_i \approx_t P_{i+1}$, which concludes the proof. □

4.4. Bounding the number of dishonest voters

This second reduction result allows one to bound the number of dishonest voters when considering BPRIV. More precisely, we consider a unique honest voter, and we show that k dishonest voters are sufficient to mount an attack against vote privacy (if such an attack exists). Here, we reduce the number of voters from n to $k+1$ (k dishonest voters plus one honest voter), and the resulting bound depends on the counting function.

Proposition 3. *Let $\mathcal{V}$ be a voting scheme whose associated counting function is k-bounded for $k \geqslant 1$. If $\mathcal{V}$ does not satisfy $\mathsf{BPRIV}(1,n)$ for some $n \geqslant 0$, then $\mathcal{V}$ does not satisfy $\mathsf{BPRIV}(1,k)$ Moreover, in that case there exists a witness of this attack where no more than k ballots reached the ballot box.*

Roughly, if $\mathsf{BPRIV}(1,n-1)$ does not hold, the difference appears either (i) when the honest voter outputs her ballot, or (ii) when outputting the result. Indeed, the behaviour of a dishonest voter who simply outputs the message he received does not help to mount an attack. Moreover, the only test that a dishonest voter performs is a public test from which the attacker will not infer anything. In case (i), no dishonest voters are even needed, and the claim holds.

In case (ii), we know that that the public terms representing the final result are different on both sides. We apply our k-boundedness hypothesis, and we know that a difference is still there when considering k voters (or even less). Removing the corresponding actions performed by dishonest voters, the trace still corresponds to an execution assuming that the validity tests do not depend on the the other ballots on the bulletin board. Hence, we have a witness of non-equivalence with at most k ballots, and thus at most k dishonest voters.

We now give a detailed proof of Proposition 3.

Proof. First, relying on Lemma 3, we know that the processes under study are action-deterministic, and therefore, thanks to Proposition 1, we can assume that a witness of an attack of minimal length has some specific shape. Following the notation introduced in Section 3, we consider $n+1$ distinct voters $\overrightarrow{v_0},\ldots,\overrightarrow{v_n}$, and we consider a witness tr of non-equivalence of minimal length. We know that:

- $\mathsf{Election}^{\mathsf{L}}_{1,n}(\overrightarrow{v_0},\overrightarrow{v_1},\ldots,\overrightarrow{v_n}) \overset{\mathsf{tr}}{\Longrightarrow} (i_{\mathsf{L}};\mathcal{P}_{\mathsf{L}};\phi_{\mathsf{L}};M_{\mathsf{L}})$ for some $(i_{\mathsf{L}};\mathcal{P}_{\mathsf{L}};\phi_{\mathsf{L}};M_{\mathsf{L}})$;
- $\mathsf{Election}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0},\overrightarrow{v_1},\ldots,\overrightarrow{v_n}) \overset{\mathsf{tr}}{\Longrightarrow} (i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$ for some $(i_{\mathsf{R}};\mathcal{P}_{\mathsf{R}};\phi_{\mathsf{R}};M_{\mathsf{R}})$;
- $i_{\mathsf{L}} = i_{\mathsf{R}}$, $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$, and tr is Σ_{err}-free.

We are going to show that this minimal witness tr is also a witness of the following non-equivalence: $\mathsf{Election}^{\mathsf{L}}_{1,k}(\overrightarrow{v_0},\overrightarrow{v_1},\ldots,\overrightarrow{v_k}) \not\approx_t \mathsf{Election}^{\mathsf{R}}_{1,k}(\overrightarrow{v_0},\overrightarrow{v_1},\ldots,\overrightarrow{v_k})$.

In the following, we will distinguish cases depending on the form of tr. Due to the form of the processes, we can assume w.l.o.g. that tr is a prefix of:

$$\mathsf{out}(ch,\mathsf{w}_0).\mathsf{phase}\,1.\mathsf{out}(c_{i_1},\mathsf{w}_{i_1}).\ldots.\mathsf{out}(c_{i_p},\mathsf{w}_{i_p}).\mathsf{phase}\,2.\mathsf{tr}_{\mathsf{cast}}.\mathsf{phase}\,3.\mathsf{out}(c_{res},\mathsf{w}_{\mathsf{tall}})$$

Case 1: tr only contains actions from phase 0 and phase 1. In such a case, tr cannot be a witness of non-equivalence. Indeed, the frames on both sides are necessarily in static equivalence.

Case 2: tr contains actions from phases 0, 1, and 2 (but no action from phase 3). We distinguish two cases.

- We first consider the case where some actions in phase 2 are performed by a dishonest voter id_j, *i.e.* there is $\mathsf{in}(c_j, R_j) \in \mathsf{tr}$ and possibly $\mathsf{out}(c_j, \mathsf{w}_j) \in \mathsf{tr}$, and $\mathsf{append}(c_j)$ as well. Then, we consider $\mathsf{tr}' = \overline{\mathsf{tr}}\{\mathsf{w}_j \mapsto R_j\}$ where $\overline{\mathsf{tr}}$ is tr in which the input, output, and append actions performed during phase 2 on channel c_j have been removed. The resulting trace tr' is smaller than tr. To conclude, it remains to show that tr' is a witness of non equivalence, thus contradicting the minimality of the witness tr.
  It is easy to see that this trace tr' still passes in $\mathsf{Election}^{\mathsf{L}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$. Note that the action $\mathsf{append}(c_j)$ has no impact since the tallying phase has not been executed. The frame ϕ'_{L} resulting from this new execution tr' is such that $\phi_{\mathsf{L}} = \phi'_{\mathsf{L}} \cup \{\mathsf{w}_j \mapsto b^0_{\mathsf{L}}\}$ where $b^0_{\mathsf{L}} = R_j\phi'_{\mathsf{L}}$ and R_j is the recipe mentioned above such that $vars(R_j) \subseteq \mathsf{dom}(\phi'_{\mathsf{L}})$.
  Similarly to the reasoning performed on the left side, this trace tr' also passes in $\mathsf{Election}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$ (since tr passes too). Moreover, the frame ϕ'_{R} resulting from this execution tr' is such that $\phi_{\mathsf{R}} = \phi'_{\mathsf{R}} \cup \{\mathsf{w}_j \mapsto b^0_{\mathsf{R}}\}$ where $b^0_{\mathsf{R}} = R_j\phi'_{\mathsf{R}}\downarrow$ considering the exact same recipe R_j as the one mentioned above. We know that $\phi'_{\mathsf{L}} \sim \phi'_{\mathsf{R}}$ implies that $\phi_{\mathsf{L}} \sim \phi_{\mathsf{R}}$, and thus since $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$, we deduce that $\phi'_{\mathsf{L}} \not\sim \phi'_{\mathsf{R}}$. This allows us to conclude that tr' is a witness of non-inclusion, and this leads to a contradiction as tr' is smaller than tr.
- We now assume that there is no input/output/append action performed by a dishonest voter during the casting phase (phase 2). In such a case, we have that either $\mathsf{tr}_{\mathsf{cast}} = \mathsf{in}(c_0, R_0).\mathsf{out}(c_0, \mathsf{w}_0).\mathsf{append}(c_0)$ or $\mathsf{tr}_{\mathsf{cast}} = \mathsf{in}(c_0, R_0).\mathsf{out}(c_0, \mathsf{w}_0)$ or $\mathsf{tr}_{\mathsf{cast}} = \mathsf{in}(c_0, R_0)$. Note that actually the first and the last case are impossible since the input and the append actions do not modify the frame, and thus are not necessary to obtain a witness of non-equivalence (of the shape mentioned above) leading a contradiction regarding minimality.
  In case phase 1 contains an output on c_i with $i > 0$, *i.e.* $\mathsf{out}(c_i, \mathsf{w}_i)$ occurs in phase 1, and $\mathsf{w}_i\phi_{\mathsf{L}} = \langle cr_i, \mathsf{Pub}(cr_i, u_i)\rangle$, we consider $\mathsf{tr}' = \overline{\mathsf{tr}}\{\mathsf{w}_i \mapsto \langle cr'_i, \mathsf{Pub}(cr'_i, u'_i)\rangle\}$, where $\overline{\mathsf{tr}}$ is tr in which this output has been removed, and cr'_i and u'_i are fresh public constants. Then tr' passes in $\mathsf{Election}^{\mathsf{L}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$ and also in $\mathsf{Election}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$. Indeed, cr_i and u_i do not occur anymore in the remaining process to be executed since DVoter is not executed for id_j.
  This trace tr' leads to the frames ϕ'_{L} (on the left) and ϕ'_{R} (on the right) such that $\phi_{\mathsf{X}} = \phi'_{\mathsf{X}}\{cr'_i \mapsto cr_i\}\{u'_i \mapsto u_i\} \cup \{\mathsf{w}_i \mapsto \langle cr_i, \mathsf{Pub}(cr_i, u_i)\rangle\}$ for $\mathsf{X} \in \{\mathsf{L}, \mathsf{R}\}$. Since, we know that $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$, we conclude that $\phi'_{\mathsf{L}} \not\sim \phi'_{\mathsf{R}}$, which proves this case. Note that, in case the distinguishing test relies on w_i, we can easily reconstruct the corresponding term $\langle cr'_i, \mathsf{Pub}(cr'_i, u'_i)\rangle$ to obtain a witness of $\phi'_{\mathsf{L}} \not\sim \phi'_{\mathsf{R}}$.
  Otherwise (no output on c_i with $i > 0$ during phase 1), the trace tr also passes starting from $\mathsf{Election}^{\mathsf{L}}_{1,k}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_k})$, or from $\mathsf{Election}^{\mathsf{R}}_{1,k}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_k})$, and the resulting frames are the same as those obtained when starting the executions from $\mathsf{Election}^{\mathsf{L}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$, and $\mathsf{Election}^{\mathsf{R}}_{1,k}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$. Therefore, tr is a witness of non-equivalence for $\mathsf{Election}^{\mathsf{L}}_{1,k}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_k}) \not\approx_t \mathsf{Election}^{\mathsf{R}}_{1,k}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_k})$ contradicting our main hypothesis.

Case 3: tr contains actions from phase 3 (actually only one). We distinguish three cases.

- If during phase 2, some action occurs on channel c_i with $i > 0$ – $\mathsf{in}(c_i, R)$, and $\mathsf{out}(c_i, \mathsf{w})$ but not the $\mathsf{append}(c_i)$ one – then we can consider $\mathsf{tr}' = \overline{\mathsf{tr}}\{\mathsf{w} \mapsto R\}$ where $\overline{\mathsf{tr}}$ is equal to tr without these actions (input and output) on channel c_i, and we can show that this trace tr' is a witness of non-equivalence obtaining a contradiction regarding the minimality of tr.
- Otherwise, if phase 1 contains an action of the form $\mathsf{out}(c_i, \mathsf{w}_i)$ corresponding to the output of a credential of a dishonest voter id_i (*i.e.* $i > 0$), whereas there is no $\mathsf{in}(c_i, R_i)$ during phase 2 for this particular (dishonest) voter, then we consider the trace tr' which is equal to tr without this output $\mathsf{out}(c_i, \mathsf{w}_i)$, and we also replace the occurrences of w_i in tr by $\langle cr'_i, \mathsf{Pub}(cr'_i, u'_i)\rangle$ where cr'_i and u'_i are fresh public constants. As before, we conclude that tr' is a smaller witness.
- We now consider the case of a trace tr that is composed of phase 1 during which only dishonest voters who cast their ballot (action append) participate to phase 1, then phase 2, and then phase 3 containing the output on channel c_{res}. We also know that the last output (the one on c_{res}) is needed to get a witness of non-equivalence, and that $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$ where ϕ_{L} and ϕ_{R} are the two resulting frames. Thus, the test distinguishing these two frames relies on $\mathsf{w}_{\mathsf{tall}}$ (the message output on c_{res}). Actually, relying on Lemma 1, we have $\mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{L}} \neq_{\mathsf{E}} \mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{R}}$. Moreover, we know that $\mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{L}} = \mathsf{count}(\mathsf{extract}(\mathsf{BB}_{\mathsf{L}}))$ and $\mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{R}} = \mathsf{count}(\mathsf{extract}(\mathsf{BB}_{\mathsf{R}}))$ where BB_{L} (resp. BB_{R}) is the bulletin board (*i.e.* the content of the memory cell m_{bb}) resulting from trace tr on the left (resp. on the right).

  If at most k voters voted (*i.e.* cast their vote - action append), then, as we know that only the dishonest voters who cast a vote output their credential during the initialisation phase, we can deduce that this witness tr is also a witness of $\mathsf{Election}^{\mathsf{L}}_{1,k}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_k}) \not\approx_t \mathsf{Election}^{\mathsf{R}}_{1,k}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_k})$.

  Otherwise, we know that n' voters with $n' > k$ have cast their vote. Thanks to our k-bounded hypothesis, we know that there exists $k' \leqslant k$, and $0 \leqslant i_1 < \ldots < i_{k'} \leqslant n$ such that counting the votes of $id_{i_1}, \ldots, id_{i_{k'}}$ still leads to a difference in the result.

  In the trace tr, we know that there are actions $\mathsf{append}(c_{i_1}), \ldots, \mathsf{append}(c_{i_{k'}})$ corresponding to the append actions of these voters $id_{i_1}, \ldots, id_{i_{k'}}$. We consider tr' obtained from tr by removing all these actions. It is easy to see that this smaller trace tr' still passes in $\mathsf{Election}^{\mathsf{L}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$ and in $\mathsf{Election}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$. The resulting bulletin board $\mathsf{BB}'_{\mathsf{L}}$ (resp. $\mathsf{BB}'_{\mathsf{R}}$) contain less ballots than before, and these ballots have been chosen such that:

  $$\mathsf{count}(\mathsf{extract}(\mathsf{BB}'_{\mathsf{L}})) \neq \mathsf{count}(\mathsf{extract}(\mathsf{BB}'_{\mathsf{R}}))$$

  Therefore, the resulting frames ϕ'_{L} and ϕ'_{R} are almost the same as ϕ_{L} and ϕ_{R}, except for the result output during the tallying phase, which we know are different public terms. As our processes are action-deterministic (Lemma 3), there is no other choice to obtain another frame, and thus tr' is a smaller witness of $\mathsf{Election}^{\mathsf{L}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n}) \not\approx_t \mathsf{Election}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$, leading again to a contradiction.

Hence the result. □

5. Dealing with revoting

We now consider the case where re-voting is allowed. We first adapt the BPRIV definition to this setting (see Section 5.1) before stating and discussing our reduction result in Section 5.2 and Section 5.3.

5.1. Modelling BPRIV with re-voting

The processes HVoter, DVoter, and Tally are left unchanged. Only the main Election processes, and the consistency assumption change. The tallying now takes into account a revote policy, indicating how to proceed when a voter casts multiple votes. A revote policy is a function:

$$\mathsf{policy} : (\Sigma_0 \times \mathcal{N}_{\mathsf{priv}} \times \mathsf{Votes})\ \mathsf{list} \to \mathsf{Votes}\ \mathsf{list}.$$

This policy function replaces lst in the strong consistency assumption (Section 3.3). We consider here the two most common revote policies. The last and first policies, that select resp. the last or the first vote from each voter.

We reuse the notations from Section 3.2, and we introduce in addition $\overrightarrow{w_i} = (d_i, id_i, cr_i, pcr_i)$ for each $i \in \{1, \ldots, n\}$ where d_i are different private channel names. The privacy property $\mathsf{BPRIVR}(p, n-p)$ is written as follows:

$$\mathsf{ElectionRevote}^{\mathsf{L}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}) \approx_t \mathsf{ElectionRevote}^{\mathsf{R}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$$

where $\mathsf{ElectionRevote}^{\mathsf{X}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}) = \mathsf{new}\ sk.\ m_{\mathsf{bb}} := \mathsf{nil}.\ \mathsf{out}(ch, \mathsf{pk}(sk)).$

$$\begin{array}{ll}
(& \mathsf{phase}\ 1.\ \mathsf{out}(c_1, pcr_1).\ \mathsf{phase}\ 2.\ !\ \mathsf{new}\ d_1.\ \mathsf{out}(c_1, d_1).\ \mathsf{HVoter}^{\mathsf{X}}(\overrightarrow{v_1}, \mathsf{pk}(sk)) \\
& \mid \ldots \\
& \mid \mathsf{phase}\ 1.\ \mathsf{out}(c_p, pcr_p).\ \mathsf{phase}\ 2.\ !\ \mathsf{new}\ d_p.\ \mathsf{out}(c_p, d_p).\ \mathsf{HVoter}^{\mathsf{X}}(\overrightarrow{w_p}, \mathsf{pk}(sk)) \\
& \mid \mathsf{phase}\ 1.\ \mathsf{out}(c_{p+1}, pcr_{p+1}).\ \mathsf{phase}\ 2.\ !\ \mathsf{new}\ d_{p+1}.\ \mathsf{out}(c_{p+1}, d_{p+1}).\mathsf{DVoter}(\overrightarrow{w_{p+1}}, \mathsf{pk}(sk)) \\
& \mid \ldots \\
& \mid \mathsf{phase}\ 1.\ \mathsf{out}(c_n, pcr_n).\ \mathsf{phase}\ 2.\ !\ \mathsf{new}\ d_n.\ \mathsf{out}(c_n, d_n).\ \mathsf{DVoter}(\overrightarrow{w_n}, \mathsf{pk}(sk)) \\
& \mid \mathsf{phase}\ 3.\ \mathsf{Tally}(sk)\ )
\end{array}$$

with $ch \in \mathcal{C}\mathsf{h}_{\mathsf{pub}}$, $\mathsf{X} \in \{\mathsf{L}, \mathsf{R}\}$.

Note that a replication operator has been added in front of the voter processes to model the fact that revote is now possible.

5.2. Reduction result with re-voting

We are now able to state our reduction result when considering re-voting.

Theorem 2. *Let $\mathcal{V}$ be a voting scheme whose associated counting function is k-bounded for some $k \geqslant 1$, and p, n be two integers such that $1 \leqslant p \leqslant n$. If $\mathcal{V}$ does not satisfy $\mathsf{BPRIVR}(p, n-p)$, then $\mathcal{V}$ does not satisfy $\mathsf{BPRIVR}(1, k)$. Moreover, in that case there exists a witness of this attack where no more than k ballots reached the ballot box (each from a different voter).*

The proof of this Theorem follows the same lines as the one when re-vote is not allowed, and is composed of two main reduction steps. Before performing these two reduction steps, we may note that our election processes are still action-deterministic. Actually, the construction $\mathsf{new}\ d.\mathsf{out}(c,d).P$ is there for that, and Proposition 1 charaterizing the form of a minimal attack trace is still valid for these election processes where re-vote is allowed. Rather than redoing the proof completely, we highlight the differences with the "no revote" case for these two steps.

Step 1: Reducing the number of honest voters to 1. We show that if $\mathsf{BPRIVR}(1, n-1)$ holds, then so does $\mathsf{BPRIV}(p, n-p)$. The proof for this step has the same structure as the one for Proposition 2. The only difference, essentially, is that instead of each honest voter only submitting one ballot, which we have to simulate for a dishonest voter, they may submit any number of ballots. Thanks to the actions $\mathsf{sess}(c_j, d)$ added to the trace, we know however which voter each ballot belongs to. Using this information, we can simulate the honest ballots, just as in the previous proof. As in the "no revote" proof, we define intermediate processes P_i for $i \in \{0, \ldots, p\}$), and we assume by contradiction that there exists i_0 such that $P_{i_0} \not\approx_t P_{i_0+1}$. We consider a minimal trace tr witnessing $P_i \not\approx_t P_{i+1}$, with associated frames ϕ_L, ϕ_R. Its shape is slightly different from the one in the previous proof, because of the $\mathsf{sess}(c_j, d)$ actions added whenever voter j is replicated for a new session. However the ideas are the same.

Step 2: Reducing the number of dishonest voters to k. Again, the shape of the witness of non-equivalence that we consider is a bit different from the one used in Proposition 3 as we now have $\mathsf{sess}(c_j, d)$ actions that will occur. Nevertheless the reasoning remains the same. We only focus on the case where tr contains actions from phase 3 (actually only one), and we distinguish 3 cases:

- If, during phase 2, some actions (*e.g.* $\mathsf{sess}(c_i, d)$, $\mathsf{in}(d, R)$, $\mathsf{out}(d, \mathsf{w})$) occur on channel c_i (with $i > 0$) but not the corresponding $\mathsf{append}(d)$ actions, then we can consider $\mathsf{tr}' = \overline{\mathsf{tr}}\{\mathsf{w} \mapsto R\}$ where $\overline{\mathsf{tr}}$ is equal to tr without these actions, and we can show that tr' is a witness of non-equivalence obtaining a contradiction regarding the minimality of tr.
- Now, in case phase 1 contains an action of the form $\mathsf{out}(c_i, \mathsf{w}_i)$ with $i > 0$, whereas there is no $\mathsf{sess}(c_i, d)$ in phase 2, then we can consider the trace tr' which is equal to tr without this output $\mathsf{out}(c_i, \mathsf{w}_i)$, and where the occurrences of w_i are replaced with $\langle cr'_i, \mathsf{Pub}(cr'_i, u'_i)\rangle$ for fresh public constants cr'_i and u'_i. As before, we conclude that tr' is a smaller witness.
- We now consider the case of a trace tr composed of a phase 1 (only voters who outputs a ballot participate to this phase 1), then a phase 2, and then the output of the result during phase 3. We have $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$ where ϕ_{L} and ϕ_{R} are the two resulting frames, and in fact, relying on Lemma 1, we have $\mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{L}} \neq_{\mathsf{E}} \mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{R}}$. Moreover, we know that:

  $$\mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{L}} = \mathsf{count}(\mathsf{policy}(\mathsf{extract}(\mathsf{BB}_{\mathsf{L}}))) \text{ and } \mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{R}} = \mathsf{count}(\mathsf{policy}(\mathsf{extract}(\mathsf{BB}_{\mathsf{R}})))$$

  where BB_{L} (resp. BB_{R}) is the bulletin board (*i.e.* the content of the memory cell m_{bb}) resulting from the trace tr on the left (resp. on the right).
  If at most k distinct voters cast their vote (action append), then we know that only these dishonest voters have output their credential during the initialisation phase, and thus this witness is also a witness of

  $$\mathsf{ElectionRevote}^{\mathsf{L}}_{1,k}(\vec{v_0}, \vec{v_1}, \ldots, \vec{v_k}) \not\approx_t \mathsf{ElectionRevote}^{\mathsf{R}}_{1,k}(\vec{v_0}, \vec{v_1}, \ldots, \vec{v_k}).$$

Moreover, this witness satisfies our requirements, which concludes the proof for this case.
Otherwise, we know that $n' > k$ votes with $n' > k$ have been cast (possibly by the same voter), *i.e.* that $\mathsf{BB}_{\mathsf{L}} = [b_1^{\mathsf{L}}, \ldots, b_{n'}^{\mathsf{L}}]$ and $\mathsf{BB}_{\mathsf{R}} = [b_1^{\mathsf{R}}, \ldots, b_{n'}^{\mathsf{R}}]$. Moreover, we know that for each pair of ballots $(b_j^{\mathsf{L}}, b_j^{\mathsf{R}})$, there exists id, cr, v^{L}, and v^{R} such that: $\mathsf{extract}(b_j^{\mathsf{L}}) = (id, cr, v^{\mathsf{L}})$ and $\mathsf{extract}(b_j^{\mathsf{R}}) = (id, cr, v^{\mathsf{R}})$. In case a voter cast more than one ballot, then we know that only one has been taken into account due to the revote policy, and thus there is i_0 such that $b_{i_0}^{\mathsf{L}}$ and $b_{i_0}^{\mathsf{R}}$ do not influence the result (since it has been removed by the revote policy). Therefore, we can remove the corresponding $\mathsf{append}(d)$ action, and we obtain a smaller trace tr' leading to the exact same frames, and same result.
Otherwise, each voter has voted only once, but $n' > k$. Therefore the policy will consider all ballots to compute the result. Thanks to our k-bounded hypothesis, we know that there exists $k' \leqslant k$, and $0 \leqslant i_1 < \ldots < i_{k'} \leqslant n$ such that

$$\mathsf{count}(\mathsf{extract}([b_{i_1}^{\mathsf{L}}, \ldots, b_{i_{k'}}^{\mathsf{L}}])) \neq \mathsf{count}(\mathsf{extract}([b_{i_1}^{\mathsf{R}}, \ldots, b_{i_{k'}}^{\mathsf{R}}]))$$

Note that, since each voter only votes once, this implies that

$$\mathsf{count}(\mathsf{policy}(\mathsf{extract}([b_{i_1}^{\mathsf{L}}, \ldots, b_{i_{k'}}^{\mathsf{L}}]))) \neq \mathsf{count}(\mathsf{policy}(\mathsf{extract}([b_{i_1}^{\mathsf{R}}, \ldots, b_{i_{k'}}^{\mathsf{R}}]))).$$

We now consider tr' which is tr without the actions $\mathsf{append}(d)$ corresponding to all the ballots that have been removed. Note that, if we want to remove the i_0^{th} ballot from the bulletin board, this corresponds to removing the i_0^{th} append actions from the trace tr. The resulting trace tr' is smaller than tr, and leads to the exact same frames, except for their last element corresponding to the output of the result. We have ensured a difference is maintained between the two sides, and thus tr' is still a witness of non-equivalence, which concludes the proof.

5.3. Discussion

Even after applying our reduction result, we may note that replication operators are still there, and thus establishing such an equivalence property (even when $p = 1$, and $k = 1$) is not trivial. Traces of unbounded length still must be considered. However, as we are able to establish that, in a minimal attack trace, at most k ballots reached the ballot box (each by a different voter), we can easily remove the replication operator in front of a dishonest voter. This reasoning does not apply for the honest voter, as the output she performed may be useful to mount an attack (contrary to the output of a dishonest voter who outputs a term known by the attacker). This has been overlooked in the reduction result presented in [6]. The security analysis of Helios with revote has been done without considering this replication operator, leading to erroneous security analysis.

6. Extension to the case of a dishonest ballot box

We now consider the case where the ballot box is no longer trusted. As in the previous section, we first adapt the BPRIV definition to this setting before stating and proving our reduction result.

6.1. *Symbolic* BPRIV *with a dishonest ballot box*

The symbolic definition we propose in Section 3.2, based on the original game-based formulation of [11], considers a setting where the ballot box is trusted. Indeed, it does not give the attacker complete control over the contents of the ballot box: the attacker cannot arbitrarily write to the ballot box, but rather can only see and block honest ballots, and cast ballots in the name of dishonest voters only.

In [12], an extension of BPRIV that features a fully dishonest ballot box is introduced: the attacker can arbitrarily choose the content of the ballot box. This creates additional difficulties compared to the honest ballot box case. In BPRIV, on the left-hand side, the attacker is shown the "real" ballots, *i.e.* the ones that will be tallied. On the right-hand side, he sees a "fake" ballot for each honest voter, while using the corresponding "real" ballot when tallying. Adapting naïvely this behaviour to the dishonest ballot box setting produces an unsatisfiable definition. Indeed, if the attacker can modify arbitrarily the "fake" ballots he received before sending them to be tallied, simply using (on the right-hand side) the unmodified "real" ballots to compute the election result would let him trivially distinguish the two sides. Instead, [12] proposes to observe how the attacker modified the "fake" ballots, and to apply the same modifications to the "real" ballots before tallying. This leads [12] to an extension of BPRIV, in the form of a cryptographic game, that relies on a so-called *recovery algorithm*, which performs the operation of finding out what modifications the attacker did on the "fake" ballots.

We propose here an adaptation of our symbolic BPRIV definition, that incorporates this idea. To simplify the presentation, we restrict ourselves to the case where voters do not revote.

Voter processes. We update the $\mathsf{HVoter}^{\mathsf{L}}$ and $\mathsf{HVoter}^{\mathsf{R}}$ processes as follows.

$$
\begin{array}{l}
\mathsf{HVoter}^{\mathsf{L}}(c, id, cr, pcr, pk) = \\
\quad \mathsf{in}(c, z). \\
\quad \mathsf{let}\ (v^0, v^1) = (\mathsf{proj}_1^2(z), \mathsf{proj}_2^2(z))\ \mathsf{in} \\
\quad \mathsf{if}\ v^0, v^1 \in \mathsf{Votes}\ \mathsf{then} \\
\qquad \mathsf{new}\ r^0.\ \mathsf{new}\ r^1. \\
\qquad \mathsf{let}\ b^0 = \mathsf{Vote}(pk, id, cr, v^0, r^0)\ \mathsf{in} \\
\qquad \mathsf{let}\ b^1 = \mathsf{Vote}(pk, id, cr, v^1, r^1)\ \mathsf{in} \\
\qquad \mathsf{if}\ \mathsf{Valid}(id, pcr, b^0, pk) = \mathsf{true} \\
\qquad\quad \mathsf{then}\ \mathsf{append}(c, \langle id, \bot, \bot \rangle_3, m_{\mathsf{bb}}^{\mathsf{ref}}). \\
\qquad\qquad\quad \mathsf{out}(c, b^0) \\
\qquad\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{invalid}}) \\
\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{vote}})
\end{array}
$$

$$
\begin{array}{l}
\mathsf{HVoter}^{\mathsf{R}}(c, id, cr, pcr, pk) = \\
\quad \mathsf{in}(c, z). \\
\quad \mathsf{let}\ (v^0, v^1) = (\mathsf{proj}_1^2(z), \mathsf{proj}_2^2(z))\ \mathsf{in} \\
\quad \mathsf{if}\ v^0, v^1 \in \mathsf{Votes}\ \mathsf{then} \\
\qquad \mathsf{new}\ r^0.\ \mathsf{new}\ r^1. \\
\qquad \mathsf{let}\ b^0 = \mathsf{Vote}(pk, id, cr, v^0, r^0)\ \mathsf{in} \\
\qquad \mathsf{let}\ b^1 = \mathsf{Vote}(pk, id, cr, v^1, r^1)\ \mathsf{in} \\
\qquad \mathsf{if}\ \mathsf{Valid}(id, pcr, b^1, pk) = \mathsf{true} \\
\qquad\quad \mathsf{then}\ \mathsf{append}(c, \langle id, b^0, b^1 \rangle_3, m_{\mathsf{bb}}^{\mathsf{ref}}). \\
\qquad\qquad\quad \mathsf{out}(c, b^1) \\
\qquad\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{invalid}}) \\
\quad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{vote}})
\end{array}
$$

They are very similar to the honest ballot box case, except that the m_{bb} list, which was used to store the list of "real" ballots to be tallied, is replaced with a list $m_{\mathsf{bb}}^{\mathsf{ref}}$, that stores no information on the left-hand side, and the correspondence between "real" and "fake" ballots on the right-hand side. Note that ballots are added to $m_{\mathsf{bb}}^{\mathsf{ref}}$ before being publicly output: indeed, we wish $m_{\mathsf{bb}}^{\mathsf{ref}}$ to store the list of all generated ballots, even ones the attacker may block later by choosing not to add them to the ballot box he will compute.

Since we give the attacker complete control over the ballot box, there is no longer need for a DVoter process to cast ballots in the name of dishonest voters.

Validity check. The ballot box to be tallied will be input directly from the attacker. Since the attacker has direct write access to the ballot box, we first check he has not added invalid ballots to it, and that only one ballot has been submitted per voter (as we exclude revote here). To do so, we will use the $\mathsf{Valid}(id, pcr, b, pk)$ recipe. To make the processes more legible, we ask that the attacker provides for each ballot the identity of the voter allegedly casting it. This information is public, so asking that does not restrict the attacker. We thus consider a computation $C_{\mathsf{Valid}}(bb, creds, pk)$, that takes as parameters a list bb of pairs (id, b) of an identity and a ballot, a list $creds$ of pairs (id, pcr) of all identities and public credentials, and the public election key pk. The computation C_{Valid} iterates through list bb, and for each element (id, b) of bb (an element not of that form causes it to fail), goes through $creds$ to:

i) check that id is indeed the identity of an actual voter,
ii) retrieve the associated pcr,
iii) check that $\mathsf{Valid}(id, pcr, b, pk) = \mathsf{true}$.

In addition, for each (id, b) in bb, it checks that no other (id, b') (with the same id) is present in bb. We do not extensively write the process C_{Valid} here – we have already shown earlier how all the operations it performs (iteration on lists, comparisons) can be implemented as computation processes.

Recovery. Before actually performing the tally, the recovery operation must be performed on the right-hand side. Depending on the way this recovery is done, the resulting definition expresses stronger or weaker guarantees. For this reason, the computational BPRIVD notion from [12] is defined parametrically w.r.t. the recovery algorithm.

In order to keep our symbolic definition similarly generic, we simply assume a computation $C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})$. It takes as parameter a list bb of pairs $\langle id, b\rangle$ of identities and ballots (to be instantiated with the list produced by the attacker), and a list bb_{ref} of tuples $\langle id, b^0, b^1\rangle_3$ (where b^0, b^1 are ballots or $\perp$), and computes the list bb_{tal} to be tallied. The recovery and tallying will then be performed in the following process:

$$
\begin{array}{l}
\mathsf{TallyRecover}(c, sk) = \mathsf{in}(c, bb).\\
\qquad\qquad \mathsf{if}\ C_{\mathsf{Valid}}(bb, \mathsf{pk}(sk)) = \mathsf{true}\ \mathsf{then}\\
\qquad\qquad\quad \mathsf{read}\ m^{\mathsf{ref}}_{\mathsf{bb}}\ \mathsf{as}\ bb_{\mathsf{ref}}.\\
\qquad\qquad\quad \mathsf{let}\ bb_{\mathsf{tal}} = C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})\ \mathsf{in}\\
\qquad\qquad\quad \mathsf{let}\ res = C_{\mathsf{Tally}}(bb_{\mathsf{tal}}, sk)\ \mathsf{in}\\
\qquad\qquad\qquad \mathsf{out}(c_r, res)\\
\qquad\qquad \mathsf{else}\ \mathsf{out}(c, \mathsf{err}_{\mathsf{vote}}).
\end{array}
$$

Example 11. *A typical recovery algorithm consists in going through the list submitted by the attacker, and, for each ballot, checking if it is equal to a "fake" ballot generated by an honest voter. If so, the ballot is replaced with the corresponding "real" ballot for that voter, otherwise it is left*

unchanged. That algorithm, as discussed in [12], leads to a property expressing that the attacker cannot modify honest ballots, but only choose to include them or to remove them – essentially, the ballots must be non-malleable.

We can encode it as the following recovery computation $C^0_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})$[2].

$$
\begin{array}{l}
C^0_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}}) = \\
\quad \mathsf{new}\ c.\ \Big(\quad \mathsf{out}(c, \langle bb, \mathsf{nil}\rangle) \\
\qquad\qquad \mid\ \mathsf{in}(c, \langle \mathsf{nil}, bb_{\mathsf{tal}}\rangle).\ \mathsf{out}(c_b, \mathsf{rev}(bb_{\mathsf{tal}})) \\
\qquad\qquad \mid\ !\ \mathsf{in}(c, \langle \langle id, b\rangle_3 :: l, bb_{\mathsf{tal}}\rangle). \\
\qquad\qquad\quad \mathsf{new}\ c'.\ (\\
\qquad\qquad\qquad \mathsf{out}(c', bb_{\mathsf{ref}}). \\
\qquad\qquad\qquad \mid\ \mathsf{in}(c', \mathsf{nil}).\ \mathsf{out}(c, \langle l, b :: bb_{\mathsf{tal}}\rangle) \\
\qquad\qquad\qquad \mid\ !\ \mathsf{in}(c', \langle id', b_0, b_1\rangle_3 :: ll). \\
\qquad\qquad\qquad\quad \mathsf{if}\ b_1 = b\ \wedge\ \langle b_0, b_1\rangle \neq \langle \bot, \bot\rangle\ \mathsf{then}\ \mathsf{out}(c, \langle l, b_0 :: bb_{\mathsf{tal}}\rangle) \\
\qquad\qquad\qquad\quad \mathsf{else}\ \mathsf{out}(c', ll)) \Big)
\end{array}
$$

As another example, we could encode a variant of that algorithm, useful e.g. *in the case of the original Helios protocol, where instead of comparing the entire ballot* b *to ballots in* bb_{ref}*, only the ciphertext* $\mathsf{proj}_2(b)$ *is compared. (Recall that in our model of Helios, ballots are* $(id, ciphertext)$ *– see Example 7). This would express a weaker property overall, as discussed in [12]: the ciphertexts are non-malleable, but the identity included in the ballot is – which makes for a very weak notion of privacy, but accurately characterises the level of security provided by the unpatched Helios.*

Assumptions on the recovery computation. Verification tools tend to reason more easily on processes with a similar structure: for this reason, we include the recovery computation on both sides of the equivalence. On the left-hand side, bb_{ref} will only contain $\langle id, \bot, \bot\rangle_3$ elements, and the recovery should not change the provided ballot box. Hence, we require that $\bot$ values are ignored: $C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})$ and $C_{\mathsf{Rec}}(bb, bb'_{\mathsf{ref}})$ should return the same result if bb_{ref} and bb'_{ref} are equal up to $\langle id, \bot, \bot\rangle_3$ elements, and $C_{\mathsf{Rec}}(bb, \mathsf{nil})$ must return bb.

Moreover, we will need for our reduction result to assume that C_{Rec} has the following property, which we call *partial recovery*. Consider a list bb of terms, and a list

$$bb_{\mathsf{ref}} = [(id_1, b^0_1, b^1_1); \ldots; (id_n, b^0_n, b^1_n)]$$

containing n honestly generated ballots for distinct voters id_i (*i.e.*, computed by the Vote recipe, with fresh randoms each time). Then we assume

- for any permutation bb'_{ref} of bb_{ref}, $C_{\mathsf{Rec}}(bb, bb'_{\mathsf{ref}}) = C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})$;
- for any partition $bb_{\mathsf{ref}} = bb_{\mathsf{ref}1} @ bb_{\mathsf{ref}2}$, $C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}}) = C_{\mathsf{Rec}}(C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}1}), bb_{\mathsf{ref}2})$.

[2]To keep the order of the list unchanged, which is required later on, we write C^0_{Rec} using a rev computation that reverses a list, which is straightforward to construct.

Intuitively, that property means that the recovery operation does not depend on the order in which ballots are cast, and can be computed piecewise on a partition of the ballots.

In addition, we assume that C_{Rec} is *computable by recipes*, in the sense that for any integers l_1, l_2, there exists a recipe R_{l_1,l_2} with only two variables x_{bb}, x_{ref}, such that for all bb of length l_1 and bb_{ref} of length l_2, the result returned by the computation $C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})$ is equal to $R_{l_1,l_2}[x_{bb} \mapsto bb, x_{\mathsf{ref}} \mapsto bb_{\mathsf{ref}}]$. The recipe R_{l_1,l_2} may of course depend on the length of the lists – typically, if C_{Rec} iterates through the list bb, it will likely use $\mathsf{hd}(\mathsf{tl}^n(x_{bb}))$ to access the n-th

Finally, we add another, more restrictive assumption on the recovery computation.

- C_{Rec} preserves the length of the list of ballots it is given: for all bb, bb_{ref}, $C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})$ has the same length as bb;
- C_{Rec} is stable by sub-list: for any bb, bb_{ref}, any sequence s of distinct integers in $\{1, \ldots, \mathsf{length}(bb)\}$, if we denote "$\cdot|_s$" the operation of keeping in a list only the elements at the positions indicated by the indices in s, then $C_{\mathsf{Rec}}(bb|_s, bb_{\mathsf{ref}}) = C_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})|_s$.

These assumptions, up to the last one, do not seem overly restrictive, and hold for all recovery algorithms considered in [12]. The final assumption on the other hand is much more restrictive: it typically prevents the recovery computation from adding back to the ballot box some ballots that may have been removed by the attacker. [12] makes use of such recovery when modelling variants of the property that express additional verifiability guarantees – basically, by having the recovery add all ballots from the voters who check their vote. We forbid such recovery computations here, meaning that our reduction result does not hold when considering verifiability guarantees. These typically require that a subset of honest voters perform verifications, and get additional assurance that their vote is counted. It is not particularly surprising that these guarantees cannot be captured by only one honest voter.

Example 12. *The computation $C^0_{\mathsf{Rec}}(bb, bb_{\mathsf{ref}})$ given in Example 11 satisfies these assumptions. That is clear from its construction for most of them: the partial recovery, ignoring $\perp$ values, length preservation and stability by sublists. The condition that it is computable by recipes is less obvious. In practical examples, such as this one, it is convenient, in order to establish that property, to use an equational theory that includes the term-level if-then-else* ite *equation introduced in section 2.1. Indeed, with that construction, it becomes quite clear how to construct a (very large) recipe that compares each element of bb with each element of bb_{ref} (for fixed-size lists), and keeps the appropriate ballots. Note that, when the processes we consider do not rely on the term-level* ite *equation (but only the recipes), adding it does not actually give more distinguishing power to the attacker. Indeed, the attacker could only gain power by using it to produce a recipe that takes different branches on either side, but in that case, only considering the condition of the construction would already produce a boolean that distinguishes the two sides.*

BPRIVD *against a dishonest ballot box.* Overall, the BPRIVD property against a dishonest ballot box is as follows.

Definition 9. *A voting scheme is* BPRIVD *for p honest voters and $n-p$ dishonest voters, written* $\mathsf{BPRIVD}(p, n-p)$*, if* $\mathsf{ElectionD}^{\mathsf{L}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}) \approx_t \mathsf{ElectionD}^{\mathsf{R}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n})$ *where*

$$
\begin{aligned}
\mathsf{ElectionD}^{\mathsf{X}}_{p,n-p}(\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}) = \mathsf{new}\ sk.\ m_{\mathsf{bb}}^{\mathsf{ref}} := \mathsf{nil}.\ \mathsf{out}(ch, \mathsf{pk}(sk)).\\
(\ \mathsf{phase}\ 1.\ \mathsf{out}(c_1, pcr_1).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{X}}(\overrightarrow{v_1}, \mathsf{pk}(sk))\\
\mid \ldots\\
\mid \mathsf{phase}\ 1.\ \mathsf{out}(c_p, pcr_p).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{X}}(\overrightarrow{v_p}, \mathsf{pk}(sk))\\
\mid \mathsf{phase}\ 1.\ \mathsf{out}(c_{p+1}, \langle cr_{p+1}, pcr_{p+1}\rangle)\\
\mid \ldots\\
\mid \mathsf{phase}\ 1.\ \mathsf{out}(c_n, \langle cr_n, pcr_n\rangle)\\
\mid \mathsf{phase}\ 3.\ \mathsf{TallyRecover}(ch, sk)\ )
\end{aligned}
$$

with $ch \in \mathcal{C}\mathsf{h}_{\mathsf{pub}}$, $\mathsf{X} \in \{\mathsf{L}, \mathsf{R}\}$.

6.2. Reduction result with dishonest ballot box

We are now able to state our reduction result when considering a dishonest ballot box.

Theorem 3. *Let $\mathcal{V}$ be a voting scheme whose associated counting function is k-bounded for some $k \geqslant 1$, and p, n be two integers such that $1 \leqslant p \leqslant n$. If $\mathcal{V}$ does not satisfy* $\mathsf{BPRIVD}(p, n-p)$*, then $\mathcal{V}$ does not satisfy* $\mathsf{BPRIVD}(1, k)$*. Moreover, in that case there exists a witness of this attack where no more than k ballots are present in the ballot box computed by the attacker (each from a different voter).*

The proof of this result follows the same steps as the one for the honest ballot box, with the same two main reductions. Note first that the election processes we defined for the dishonest ballot box case are still action-deterministic. In fact, Proposition 1 still holds for these new election processes. Rather than re-doing the entire reduction proof, we detail here only the differences with the honest ballot box case.

Step 1: Reducing the number of honest voters to 1. We first show that, assuming $\mathsf{BPRIVD}(1, n-1)$ holds, then $\mathsf{BPRIVD}(p, n-p)$ also does. The main idea for this proof is similar to the one for Proposition 2: we define hybrid election processes P_i, where the first i honest voters behave like $\mathsf{HVoter}^{\mathsf{R}}$, while the other $p-i$ behave like $\mathsf{HVoter}^{\mathsf{L}}$, going gradually from $\mathsf{ElectionD}^{\mathsf{L}}_{p,n-p}$ to $\mathsf{ElectionD}^{\mathsf{R}}_{p,n-p}$ as i increases. Formally, fixing n distinct voters $\overrightarrow{v_1}, \ldots, \overrightarrow{v_n}$, with

$\overrightarrow{v_i} = (c_i, id_i, cr_i, pcr_i)$, $pcr_i = \mathsf{Pub}(cr_i, u_i)$ for all i, and $p \in \{1, \ldots, n\}$, we define:

$$
\begin{array}{l}
P_i = \mathsf{new}\ sk.\ m_{\mathsf{bb}}^{\mathsf{ref}} := \mathsf{nil}.\ \mathsf{out}(ch, \mathsf{pk}(sk)). \\
\quad (\ \mathsf{phase}\ 1.\mathsf{out}(c_1, pcr_1).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{R}}(\overrightarrow{v_1}, \mathsf{pk}(sk)) \\
\quad \mid \ldots \\
\quad \mid \mathsf{phase}\ 1.\mathsf{out}(c_i, pcr_i).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{R}}(\overrightarrow{v_i}, \mathsf{pk}(sk)) \\
\quad \mid \mathsf{phase}\ 1.\mathsf{out}(c_{i+1}, pcr_{i+1}).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{L}}(\overrightarrow{v_{i+1}}, \mathsf{pk}(sk)) \\
\quad \mid \ldots \\
\quad \mid \mathsf{phase}\ 1.\mathsf{out}(c_p, pcr_p).\ \mathsf{phase}\ 2.\ \mathsf{HVoter}^{\mathsf{L}}(\overrightarrow{v_p}, \mathsf{pk}(sk)) \\
\quad \mid \mathsf{phase}\ 1.\mathsf{out}(c_{p+1}, \langle cr_{p+1}, pcr_{p+1} \rangle) \\
\quad \mid \ldots \\
\quad \mid \mathsf{phase}\ 1.\mathsf{out}(c_n, \langle cr_n, pcr_n \rangle) \\
\quad \mid \mathsf{phase}\ 3.\ \mathsf{TallyRecover}(ch, sk)\ )
\end{array}
$$

We then show that for all $i \in \{0, \ldots, p-1\}$, we have that $P_i \approx_t P_{i+1}$. Since the two extreme processes P_0, P_p are in fact $\mathsf{ElectionD}^{\mathsf{L}}_{p,n-p}$ and $\mathsf{ElectionD}^{\mathsf{R}}_{p,n-p}$, this will prove by transitivity that $\mathsf{BPRIVD}(p, n-p)$ holds. Let $i \in \{0, \ldots, p-1\}$, and $Q_{\mathsf{X}} = \mathsf{ElectionD}^{\mathsf{X}}_{1,n-1}(\overrightarrow{v_{i+1}}, \overrightarrow{v_1}, \ldots, \overrightarrow{v_i}, \overrightarrow{v_{i+2}}, \ldots, \overrightarrow{v_n})$ for $\mathsf{X} = \mathsf{L}, \mathsf{R}$. By assumption, $\mathsf{BPRIVD}(1, n-1)$ holds, and thus $Q_{\mathsf{L}} \approx_t Q_{\mathsf{R}}$. To prove that $P_i \approx_t P_{i+1}$, we will show that from a (minimal) witness tr of non-equivalence of these two processes, we can construct a trace $\overline{\mathsf{tr}}$ showing the non-equivalence of Q_{L} and Q_{R}. The construction of $\overline{\mathsf{tr}}$ is quite similar to the one in the proof of Proposition 2, with some added difficulties. In the honest case, we had to show that the attacker can use dishonest voters to simulate the behaviour of the honest ones, and cast the appropriate ballots to obtain the same election result in Q_{X} as in P_i, P_{i+1}. In the case of the dishonest ballot box, this part of the proof is easier: the attacker can reconstruct the ballots from honest voters, except $\overrightarrow{v_{i+1}}$, in the same way as in the honest case, and simply use these in the recipe that constructs the final ballot box bb. The difficulty comes next: in $\mathsf{ElectionD}^{\mathsf{X}}(1, n-1)$, for all these simulated honest voters, no binding will be added to $m_{\mathsf{bb}}^{\mathsf{ref}}$. Thus, the recovered ballot box bb_{tal}, obtained after the recovery computation, will differ from the one obtained in P_i. To avoid this issue, the attacker needs to apply part of the recovery computation himself before submitting the ballot box.

From Proposition 1, tr is Σ_{err}-free, is executable in P_i, P_{i+1}, and produces frames ϕ_{L}, ϕ_{R} such that $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$. From the form of the processes, we may w.l.o.g. assume that tr is (a prefix of)

$$\mathsf{out}(ch, \mathsf{w}_0).\mathsf{phase}\,1.\mathsf{out}(c_{i_1}, \mathsf{w}_{i_1}).\ldots.\mathsf{out}(c_{i_p}, \mathsf{w}_{i_p}).\mathsf{phase}\,2.\mathsf{tr}_{\mathsf{cast}}.\mathsf{phase}\,3.\mathsf{in}(ch, R_{bb}).\ \mathsf{out}(c_{res}, \mathsf{w}_{\mathsf{tall}})$$

where $\mathsf{tr}_{\mathsf{cast}}$ only contains at most one input (recipe R_i) then one output (frame variable w'_i) on each channel c_i, and no operation on other channels.

We can define for each $j \in \{1, \ldots, p\}$ with $j \neq i+1$ recipes B^0_j, B^1_j, that compute the ballots b_0, and b_1 produced by the honest voter $\overrightarrow{v_j}$. We omit the details of that construction, as they are similar to the honest ballot box case. Note that Lemma 4, regarding randomness independence, extends to the C_{Rec} computation, and allows us to rename all private names with fresh public names, as in the previous proof – we will omit the details of that renaming from now on, for better legibility.

Just as in the previous proof, we let $\overline{\mathsf{tr}}$ be the trace containing the same actions as tr, except that in $\mathsf{tr}_{\mathsf{cast}}$, all inputs on c_j with $j \neq i+1$ are removed, and the recipe R_{bb} used by the attacker to compute the ballot box for the input in phase 3 is replaced with a new recipe $\overline{R_{bb}}$ which we will define shortly. Note, first, that regardless of how $\overline{R_{bb}}$ is defined, $\overline{\mathsf{tr}}$ is executable in Q_{L}, Q_{R} up to the input in phase 3, and produces at that stage frames $\overline{\phi}'_{\mathsf{L}}$, $\overline{\phi}'_{\mathsf{R}}$.

We first let $R'_{bb} = R_{bb}\{\mathsf{w}'_j \mapsto B^1_j\}_{1 \leqslant j \leqslant i}\{\mathsf{w}'_j \mapsto B^0_k\}_{i+1 \leqslant j \leqslant p}$. Intuitively, R'_{bb} represents the same computation as R_{bb}, except that all ballots produced by the honest voters $\overrightarrow{v_j}$, with $j \neq i+1$, are replaced with the appropriate recipe (recall that these voters vote the same way in P_i, P_{i+1}). Just as in the honest case proof, it can be shown that the recipes B^0_j, B^1_j produce the expected ballots when applied to both $\overline{\phi}'_{\mathsf{L}}$ and $\overline{\phi}'_{\mathsf{R}}$, and thus R'_{bb}, applied to these frames, produces the same ballot boxes bb_{L}, bb_{R} that R_{bb} computes on ϕ_{L}, ϕ_{R}.

Our goal is to choose $\overline{R_{bb}}$ in such a way that the ballot boxes $\overline{bb}_{\mathsf{L}}$, $\overline{bb}_{\mathsf{R}}$ it produces on $\overline{\phi}'_{\mathsf{L}}$, $\overline{\phi}'_{\mathsf{R}}$ will lead, when given to the $\mathsf{TallyRecover}$ process, to the same result being output in Q_{X} as the one for bb_{L}, bb_{R} in P_i, P_{i+1}. The $\mathsf{TallyRecover}$ part in these processes differs only in the C_{Rec} computation. Indeed, in P_i, P_{i+1}, that computation operates on a list $bb_{\mathsf{ref},\mathsf{L}}$, $bb_{\mathsf{ref},\mathsf{R}}$ (read from $m^{\mathsf{ref}}_{\mathsf{bb}}$), containing all honest ballots from $\overrightarrow{v_1}, \ldots, \overrightarrow{v_p}$. In Q_{L} (resp. Q_{R}), the list $\overline{bb}_{\mathsf{ref},\mathsf{L}}$ (resp. $\overline{bb}_{\mathsf{ref},\mathsf{R}}$) only contains the honest ballot for voter $\overrightarrow{v_{i+1}}$. Let bb'_{ref} be the list obtained from $bb_{\mathsf{ref},\mathsf{L}}, bb_{\mathsf{ref},\mathsf{R}}$ by removing the ballot for $\overrightarrow{v_{i+1}}$. Note that by construction of P_i, P_{i+1}, that list only contains ballots from voters that behave the same in both processes, and is thus the same starting from $bb_{\mathsf{ref},\mathsf{L}}$ or $bb_{\mathsf{ref},\mathsf{R}}$. By the partial recovery assumption, the recovery computation can be performed first on $\overline{bb}_{\mathsf{ref},\mathsf{X}}$, (for $\mathsf{X} = \mathsf{L}, \mathsf{R}$) and then on bb'_{ref}:

$$C_{\mathsf{Rec}}(bb_{\mathsf{X}}, bb_{\mathsf{ref},\mathsf{X}}) = C_{\mathsf{Rec}}(C_{\mathsf{Rec}}(bb_{\mathsf{X}}, bb'_{\mathsf{ref}}), \overline{bb}_{\mathsf{ref},\mathsf{X}}).$$

Thus, our goal is achieved if we find a $\overline{R_{bb}}$ that produces the list $C_{\mathsf{Rec}}(bb_{\mathsf{X}}, bb'_{\mathsf{ref}})$ when applied to $\overline{\phi}_{\mathsf{X}}$. By assumption, C_{Rec} is computable by recipes, and thus there exists a recipe R with two variables x_{bb}, x_{ref} such that $C_{\mathsf{Rec}}(bb_{\mathsf{X}}, bb'_{\mathsf{ref}}) = R[x_{bb} \mapsto bb_{\mathsf{X}}, x_{\mathsf{ref}} \mapsto bb'_{\mathsf{ref}}]$ for $\mathsf{X} = \mathsf{L}, \mathsf{R}$. We have already constructed a recipe R'_{bb} producing bb_{X} from $\overline{\phi}_{\mathsf{X}}$. Using the recipes B^0_j, B^1_j, and variable w'_{i+1}, we can easily construct a recipe R_2 that produces bb'_{ref} from $\overline{\phi}_{\mathsf{X}}$. Choosing $\overline{R_{bb}} = R[x_{bb} \mapsto R'_{bb}, x_{\mathsf{ref}} \mapsto R_2]$ then achieves our goal. The trace $\overline{\mathsf{tr}}$ obtained using $\overline{R_{bb}}$ as input in phase 3 is executable in Q_{L}, Q_{R}, and produces frames $\overline{\phi}_{\mathsf{L}}$, $\overline{\phi}_{\mathsf{R}}$. Moreover it leads to the same result being output in $\overline{\phi}_{\mathsf{L}}$ (resp. $\overline{\phi}_{\mathsf{R}}$) as in ϕ_{L} (resp. ϕ_{R}).

The end of the proof is then just as in the honest ballot box case: we show that ϕ_{L} and ϕ_{R} can be reconstructed from $\overline{\phi}_{\mathsf{L}}$, $\overline{\phi}_{\mathsf{R}}$ using the same recipe, and thus deduce that since $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$, we have $\overline{\phi}_{\mathsf{L}} \not\sim \overline{\phi}_{\mathsf{R}}$, meaning that $\overline{\mathsf{tr}}$ is indeed a witness of non-equivalence of Q_{L}, Q_{R}.

Step 2: Reducing the number of dishonest voters to k. The second step of the proof is to show that for a k-bounded counting function, if $\mathsf{BPRIVD}(1, n)$ does not hold, then neither does $\mathsf{BPRIVD}(1, k)$. We in fact show that there is then a witness of an attack against $\mathsf{BPRIVD}(1, k)$ where the ballot box submitted for tallying by the attacker contains at most k ballots.

The proof is very similar to the one for Proposition 3, in the case of an honest ballot box.

Consider a minimal-length attack trace tr on $\mathsf{BPRIVD}(1,n)$, *i.e.* one that distinguishes $\mathsf{ElectionD}^{\mathsf{L}}_{1,n}(\overrightarrow{v_0},\ldots,\overrightarrow{v_n})$ and $\mathsf{ElectionD}^{\mathsf{R}}_{1,n}(\overrightarrow{v_0},\ldots,\overrightarrow{v_n})$ for some distinct voters $\overrightarrow{v_0},\ldots,\overrightarrow{v_n}$. By Proposition 1, we know that the sequence of actions tr is Σ_{err}-free, is executable (and reaches the same phase) in both elections processes, where it produces respectively frames ϕ_{L}, ϕ_{R} such that $\phi_{\mathsf{L}} \not\sim \phi_{\mathsf{R}}$. Moreover, from the form of the processes, we may assume w.l.o.g. that tr is a prefix of

$$\mathsf{out}(ch,\mathsf{w}_0).\mathsf{phase}\,1.\mathsf{out}(c_{i_1},\mathsf{w}_{i_1}).\ldots.\mathsf{out}(c_{i_p},\mathsf{w}_{i_p}).\mathsf{phase}\,2.\mathsf{tr}_{\mathsf{cast}}.\mathsf{phase}\,3.\mathsf{in}(ch,R_{bb}).\ \mathsf{out}(c_{res},\mathsf{w}_{\mathsf{tall}})$$

where $\mathsf{tr}_{\mathsf{cast}}$ only contains at most one input (recipe R_0), then one append, then one output (frame variable w'_0) on channel c_0, and no operation on other channels.

We will now show that tr is in fact also a witness that $\mathsf{ElectionD}^{\mathsf{L}}_{1,k}(\overrightarrow{v_0},\ldots,\overrightarrow{v_k}) \not\approx_t \mathsf{ElectionD}^{\mathsf{R}}_{1,k}(\overrightarrow{v_0},\ldots,\overrightarrow{v_k})$, which will conclude the proof.

As in the case of the honest ballot box, we can distinguish three cases, depending on which point of the execution tr reaches.

Case 1: tr only contains actions in phases 0 and 1. That is not possible: all messages output in these phases are the same on both sides, and thus tr could not be a witness of non-equivalence.

Case 2: tr contains actions in phases 0, 1, 2, and potentially the input in phase 3, but not the output in phase 3. In other words, it reaches the casting phase, and the attacker maybe submits a ballot box, but the execution stops before the result is received. We can immediately discard the case where the input in phase 3 is performed: if that were so, then stopping the execution just before that input would still be an attack witness, since the input itself does not gain the attacker any new information. Hence, tr would not be minimal. In addition, $\mathsf{tr}_{\mathsf{cast}}$ necessarily contains an output on c_0: otherwise, simply removing it and stopping tr at the end of phase 1 would yield a shorter attack trace, as inputs and appends do not modify the frame. Thus, $\mathsf{tr}_{\mathsf{cast}} = \mathsf{in}(c_0,R_0).\ \mathsf{append}(c_0).\ \mathsf{out}(c_0,\mathsf{w}'_0)$. We then conclude by the same argument as in the honest ballot box case. In summary, any dishonest voters outputting their credentials in phase 1 are not needed to construct the frame, as the attacker could obtain the same frame (up to renaming) by using a credential he generated himself. Removing them would produce a shorter attack trace, hence by minimality there are none in tr. Thus the attack only involves one honest voter, and tr is already an attack trace witnessing $\mathsf{ElectionD}^{\mathsf{L}}_{1,k}(\overrightarrow{v_0},\ldots,\overrightarrow{v_k}) \not\approx_t \mathsf{ElectionD}^{\mathsf{R}}_{1,k}(\overrightarrow{v_0},\ldots,\overrightarrow{v_k})$.

Case 3: tr contains all actions from phase 3, *i.e.* reaches the final output of the election result. Let us call R_{bb} the recipe provided by the attacker in phase 3, and $bb_{\mathsf{L}} = R_{bb}\phi_{\mathsf{L}}$, $bb_{\mathsf{R}} = R_{bb}\phi_{\mathsf{R}}$ the ballot boxes submitted on the left and right side. Let also $bb_{\mathsf{ref,L}}$, $bb_{\mathsf{ref,R}}$ be the lists recording the (only) honest ballot on the left and on the right. With these notations, the election results output on each side are $\mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{L}} = \mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{L}},bb_{\mathsf{ref,L}}))$ and $\mathsf{w}_{\mathsf{tall}}\phi_{\mathsf{R}} = \mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{R}},bb_{\mathsf{ref,R}})))$.

By construction of the election process, both ballot boxes pass the validity checks of C_{Valid}. Thus, bb_{L} and bb_{R} are two lists of pairs $\langle id,b\rangle$, where id is one of the voters' identities, and b a valid ballot for the credential output for id in phase 1. Moreover, the C_{Valid} computation ensures that all identities in each list are distinct.

By the same reasoning as in the honest ballot box case, the static non-equivalence of ϕ_{L} and ϕ_{R} necessarily comes from the result of the election: $\mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{L}},bb_{\mathsf{ref,L}})) \neq \mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{R}},bb_{\mathsf{ref,R}})))$.

The two lists bb_{L}, bb_{R} have the same length – otherwise, checking their length instead of submitting them as input would let the attacker obtain a smaller non-equivalence witness. Thus, by assumption on C_{Rec}, so do $C_{\mathsf{Rec}}(bb_{\mathsf{L}}, bb_{\mathsf{ref,L}})$ and $C_{\mathsf{Rec}}(bb_{\mathsf{R}}, bb_{\mathsf{ref,R}})$. Thus, by the k-boundedness assumption on count, there exists a sequence s of $k' \leqslant k$ indices, such that $\mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{L}}, bb_{\mathsf{ref,L}})|_s) \neq \mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{R}}, bb_{\mathsf{ref,R}})|_s))$ (recall that "$l|_s$" denotes the list obtained by keeping only the elements of indices in s of a list l).

Then, by assumption on C_{Rec}, we get that keeping only the ballots pointed by s in the ballot box submitted by the attacker leads to a different result: $\mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{L}}|_s, bb_{\mathsf{ref,L}})) \neq \mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{R}}|_s, bb_{\mathsf{ref,R}})))$. Let R'_{bb} denote the recipe that constructs the list of elements of R_{bb} pointed by indices in s.

These indices point to k' elements of bb_{L}, bb_{R}, each of the form (id, b), where the ids are distinct voter identities (in $\overrightarrow{v_0}, \ldots, \overrightarrow{v_n}$). Moreover, for each index i, the ith elements of bb_{L} and bb_{R} necessarily contain the same id – otherwise, again, as the identities are public values, the attacker could build a shorter attack witness by simply looking at that element. Consequently, the ballots in $R'_{bb}\phi_{\mathsf{L}}$ and $R'_{bb}\phi_{\mathsf{R}}$ are recorded for the same k' distinct voter identities.

Consider the trace $\overline{\mathsf{tr}}$, obtained from tr by keeping in phase 1 only the actions related to those k' voters, and replacing R_{bb} with R'_{bb} – using public names instead of w_i variables for removed voters, if they were used in that recipe.

Consider also the $\mathsf{ElectionD}^{\mathsf{L}}_{1,k}$ and $\mathsf{ElectionD}^{\mathsf{R}}_{1,k}$ processes featuring the k' voters we kept, plus $\overrightarrow{v_0}$ (if that voter was not already one of the k'), and $k - k'$ additional dishonest voters (that are unused in $\overline{\mathsf{tr}}$). $\overline{\mathsf{tr}}$ can be executed in those processes. That is clear up to the input in phase 3. That input is instantiated with $bb_{\mathsf{L}}|_s$ on the left, and $bb_{\mathsf{R}}|_s$ on the right. The validity test is then performed on that ballot box, and succeeds – indeed, it succeeded on bb_{L} and bb_{R} in tr, and by construction the removal of some of the dishonest voters cannot make it fail. The final result election output when executed $\overline{\mathsf{tr}}$ is $\mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{L}}|_s, bb_{\mathsf{ref,L}}))$ on the left, and $\mathsf{count}(\mathsf{extract}(C_{\mathsf{Rec}}(bb_{\mathsf{R}}|_s, bb_{\mathsf{ref,R}}))$ on the right – two different public values. $\overline{\mathsf{tr}}$ is therefore an attack witness on $\mathsf{BPRIVD}(1, k)$, in which the submitted ballot box has length $k' \leqslant k$, which concludes the proof.

7. Applications and case studies

To illustrate the generality of our result, and to showcase how useful it can be in practice, we apply it to several well-known voting protocols from the literature considering different counting functions. In this section, we first present the counting functions, as well as the e-voting protocols that we consider for our analysis. Then, we discuss the results we obtained relying on the Proverif tool. Note that the analysis performed using Proverif is only render possible thanks to our reduction results that allows one to obtain a bound on the number of voters and on the number of ballots that reach the ballot box.

7.1. Counting functions under study

We apply our results on several case studies considering different counting functions. We have already introduced some classical counting functions in Section 4.1, namely multiset, sum, and majority, and we have shown that they are 1-bounded. We now add an example of a more involved counting functions: Single Transferable Vote (STV), used *e.g.* in the Australian legislative elections, for which we establish that it is 5-bounded when considering 3 candidates for 1 seat.

Single Transferable Vote (STV) is a system where each voter casts a single ballot containing a total ordering of all candidates. A vote goes to the voter's first choice. If that choice is later eliminated, instead of being thrown away, the vote is transferred to her second choice, and so on. In each round, the least popular candidate is eliminated. His votes are transferred based on voters' subsequent choices. The process is repeated until one candidate remains, who is declared the winner. We assume a total order $\prec$ on candidates is picked beforehand, and is used to break ties. The STV counting function outputs a term representing the winning candidate; it is parametrised by the set of candidates and the order $\prec$. Let $\mathsf{Count}^3_{\mathsf{STV}}$ the STV function for candidates $\{a, b, c\}$ with $a \prec b \prec c$. Votes are 3-tuples: $(c_1; c_2; c_3)$ where $\{c_1, c_2, c_3\} = \{a, b, c\}$ and c_i denotes the i^{th} choice.

Example 13. *Let $v = (a; b; c)$ and $v' = (a; c; b)$. We have $v \neq v'$, however $\mathsf{Count}^3_{\mathsf{STV}}([v]) = \mathsf{Count}^3_{\mathsf{STV}}([v']) = a$. Thus, the previous reasoning to establish 1-boundedness does not apply here.*

Lemma 5. $\mathsf{Count}^3_{\mathsf{STV}}$ *is 5-bounded.*

Proof. We assume that $a \prec b \prec c$. Let $\ell = [v_1, \ldots, v_n]$ and $\ell' = [v'_1, \ldots, v'_n]$ be two lists of Votes such that $\mathsf{Count}^3_{\mathsf{STV}}(\ell) \neq \mathsf{Count}^3_{\mathsf{STV}}(\ell')$. For each $1 \leqslant i \leqslant n$, we denote $(c_{i,1}; c_{i,2}; c_{i,3})$ the vote v_i and $(c'_{i,1}; c'_{i,2}; c'_{i,3})$ the vote v'_i.

Case 1: There exists $1 \leqslant i_0 \leqslant n$ such that $v_{i_0} = (c_{i_0,1}; c_{i_0,2}; c_{i_0,3})$ and $v'_{i_0} = (c'_{i_0,1}; c'_{i_0,2}; c'_{i_0,3})$ with $c_{i_0,1} \neq c'_{i_0,1}$. In such a case, we keep this vote, and we have

$$c_{i_0,1} = \mathsf{Count}^3_{\mathsf{STV}}([v_{i_0}]) \neq \mathsf{Count}^3_{\mathsf{STV}}([v'_{i_0}]) = c'_{i_0,1}.$$

Case 2: Otherwise, for $1 \leqslant i \leqslant n$, we have $c_{i,1} = c'_{i,1}$. Thus, at the first round, the eliminated candidate is the same on both sides. Call it c_0. If c_0 does not occur as the first choice on a vote, *i.e.* $c_0 \neq c_{i,1}$ for all i (and thus $c_0 \neq c'_{i,1}$, as $c_{i,1} = c'_{i,1}$), then the eliminated candidate at the second round will be the same on both sides, and the winner as well, contradicting our hypothesis.

Hence, c_0 occurs as the first choice in some votes. Let $i_0, \ldots, i_k$ denote the indices of all such votes. We have $c_{i_j,1} = c'_{i_j,1} = c_0$ for any $j \in \{0, \ldots, k\}$. If the second choice is the same in all these votes, *i.e.* for $j \in \{0, \ldots, k\}$, we have $c_{i_j,2} = c'_{i_j,2}$, then the eliminated candidate at the second round, and thus the winner, would be the same on both sides, which contradicts our hypothesis.

Therefore, there exists $j \in \{i_0, \ldots, i_k\}$ such that $v_j = (c_0, c_1, c_2)$, $v'_j = (c_0, c_2, c_1)$ where $\{c_0, c_1, c_2\} = \{a, b, c\}$. We keep v_j, but we need more, as $\mathsf{Count}^3_{\mathsf{STV}}([v_j]) = \mathsf{Count}^3_{\mathsf{STV}}([v'_j]) = c_0$. Since c_0 is eliminated at the first round:

(1) Either $c_0 = a$ and there exist j_1, j_2 such that $c_{j_1,1} = c'_{j_1,1} = b$, and $c_{j_2,1} = c'_{j_2,1} = c$. Keeping these two votes in addition to v_j/v'_j, we get $\mathsf{Count}^3_{\mathsf{STV}}([v_j, v_{j_1}, v_{j_2}]) \neq \mathsf{Count}^3_{\mathsf{STV}}([v'_j, v'_{j_1}, v'_{j_2}])$.
(2) Or $c_0 = b$ and there exist j_1, j_2, j_3 (all distinct) such that $c_{j_1,1} = c'_{j_1,1} = a$, $c_{j_2,1} = c'_{j_2,1} = a$, and $c_{j_3,1} = c'_{j_3,1} = c$. Keeping these three votes in addition to v_j/v'_j, we have $\mathsf{Count}^3_{\mathsf{STV}}([v_j, v_{j_1}, v_{j_2}, v_{j_3}]) \neq \mathsf{Count}^3_{\mathsf{STV}}([v'_j, v'_{j_1}, v'_{j_2}, v'_{j_3}])$.
(3) Or $c_0 = c$ and there exist distinct j_1, j_2, j_3, j_4 such that $c_{j_1,1} = c'_{j_1,1} = a$, $c_{j_2,1} = c'_{j_2,1} = a$, $c_{j_3,1} = c'_{j_3,1} = b$, and $c_{j_4,1} = c'_{j_4,1} = b$. Keeping these four votes in addition to v_j/v'_j, we get $\mathsf{Count}^3_{\mathsf{STV}}([v_j, v_{j_1}, v_{j_2}, v_{j_3}, v_{j_4}]) \neq \mathsf{Count}^3_{\mathsf{STV}}([v'_j, v'_{j_1}, v'_{j_2}, v'_{j_3}, v'_{j_4}])$.

We conclude that at most 5 votes are needed to ensure the result will be different. □

In the following, we consider majority, multiset, sum, and STV (restricted to 3 candidates).

7.2. E-voting protocols under study

For our case study, we chose the following protocols: two variants of Helios [1], corresponding to its original version, subject to the attack discussed earlier, and a fixed version that includes identities in the ZKP; Belenios [28], and the related BeleniosRF [29] and BeleniosVS [20]; Civitas [30]; and Prêt-à-Voter [31, 32]. Some of the protocols (notably Helios, Belenios) can make use of homomorphic encryption, so that all encrypted votes can be summed before decryption. In our case study however, we only consider the mixnet version of these protocols, where ballots are instead mixed in a random order before decryption. Indeed, even if our reduction results apply in presence of homomorphic encryption, Proverif does not support the equations needed to define such a primitive.

Several versions of Helios. We consider several versions of the Helios protocol depending on whether the identity of the voter is part of the zero-knowledge proof (ZKP) or not [1]. The original version of Helios is the one without the identity of the voter in the ZKP, which we described in our running example. Note that this protocol is subject to a replay attack, described in [5], where the attacker submits, in the name of a dishonest voter, a copy of a honest voter's ballot, which lets him break that voter's privacy. We do indeed find this attack, even with two voters.

This attack can be mitigated in two ways. The first one consists in adding the voter's identity to the ZKP, preventing the attacker to replay it in his name. The second one is *weeding*, *i.e.* adding a mechanism allowing the server to remove duplicate ballots. However, this operation is only effective if we assume that the ballots emitted by honest voters correctly reach the ballot box. Indeed, the attacker can otherwise simply block the original ballot, and still break the privacy property without the server being able to detect the copy. In our communication model, we decided not to make such a strong assumption, and the attacker is thus able to block messages. herefore, in our framework, *weeding* is not a solution to prevent the replay attack mentioned above. We do not study this version of Helios, as the validity test with weeding does not fall in our framework described in Section 3, since we require that the validity test does not depend on the current content of the ballot box.

Note that the framework proposed in [6] makes the assumption that the ballots meant to be counted for each honest voter (typically the last one emitted), which are the ones swapped to express the privacy property, reach the ballot box. Against that weaker attacker, Helios with weeding is secure, as long as voters only vote once. It is not, though, if they revote: the replay could be performed using a previous ballot, which the attacker is allowed to block. Their security analysis correctly finds Helios with weeding secure when revote is disallowed. However, due to a misunderstanding of the reduction result, a replication operator is missing in the case of revote, which leads to the attack being missed in that case.

Several versions of Belenios. The Belenios system [28] builds on Helios and relies on an additional authority who is in charge of distributing to each voter a key pair. The signature key is given to each voter, and the list of associated public keys constitutes the list of eligible voters. Recently, a variant of Belenios, BeleniosRF, designed to ensure receipt-freeness has been proposed [29]. It is based on a cryptographic primitive called signatures on randomizable ciphertexts. We also propose an analysis of BeleniosVS that has been designed to achieve both privacy and verifiability against a dishonest voting server [20].

Table 1
Summary of our results. ✓: Proverif proves the property. ✗: Proverif finds an attack trace. ? Proverif answers "cannot be proved". ⏱: timeout (⩾ 1h). Execution times are on an Intel i7-1068NG7 CPU.

	Counting / Protocols	Multiset/Maj/Sum (2 voters/1 ballot)		Single Transferable Vote (6 voters/5 ballots)	
without revote	Helios (*id* in ZKP)	✓	⩽ 1s	✓	∼ 24 min
	Helios (ZKP without *id*)	✗	⩽ 1s	✗	∼ 27 min
	Belenios	✓	⩽ 1s	✓	∼ 27 min
	BeleniosRF	✓	∼ 3s	⏱	
	BeleniosVS	✓	∼ 3s	⏱	
	Civitas	✓	⩽ 1s	✓	∼ 39 min
	Prêt-à-Voter	✓	⩽ 1s	⏱	
revote	Helios (*id* in ZKP)	✓	⩽ 1s	✓	∼ 23 min
	Helios (ZKP without *id*)	✗	⩽ 1s	✗	∼ 42 min
	Belenios	✓	⩽ 1s	✓	∼ 23 min
dishonest ballot box	Helios (*id* in ZKP)	✓	⩽ 2s	-	-
	Helios (ZKP without *id*)	?	⩽ 3s	-	-
	Belenios	✓	⩽ 9s	-	-

Civitas [30, 33]. This system relies on anonymous credential to enforce privacy. They consist of random values encrypted with the public key of the election authority. Comparison between credentials (e.g. the one used to cast a ballot and those composing the list of legitimate voters) is done relying on plaintext equivalence tests.

Prêt-à-Voter [34]. The ballot contains a detachable list of candidate names, given in a random order (usually the left part), and their corresponding encryption in the same order on the right. Once the voter marked the candidate of his choice, he posts the encrypted part of the ballot (the right part) on the bulleting board. Then, the design of Prêt-à-Voter is similar to other voting systems.

7.3. Results

We conduct the analysis for different counting functions, using our result to bound the number of agents and ballots. We considered majority, multiset, sum, and STV (restricted to 3 candidates). In fact, in the case of 1-bounded functions, since only one ballot needs to be accepted by the ballot box, the tallying is trivial, and ends up being the same for different functions (majority, multiset, *etc.*). Thus, a single Proverif file is enough to model several counting functions as once. The encoding for modelling STV (5-bounded for 3 candidates) is more complex, with 6 voters and 5 ballots. We only model this counting function when considering an honest ballot box (with or without revote). In presence of a dishonest ballot box, the recovery process for 6 voters and 5 ballots will require several nested conditionals, and we anticipate that Proverif (or rather, the recent prototype developed to go beyond diff-equivalence, which is required to conclude on this case) will not be able to obtain results in less than 1 hour (the timeout we consider here).

We modelled the protocols briefly presented in Section 7.2 as processes satisfying our assumptions, and analysed them using Proverif. We only prove BPRIV itself with Proverif. Strong correctness only involves terms, and can easily be proved by hand. Strong consistency requires to show that the tallying process rightly computes the tally, which Proverif is not well-suited to do, as it requires 1) modelling the tally in the general case, *i.e.* with no bounds on the lengths of lists, and 2) comparing it to the abstract definition of the counting function, which Proverif cannot really manipulate. The property clearly holds though, and could be proved by hand. We considered both the cases without and with revote, for protocols that support revoting (except Civitas, which in that case uses rather complex mechanisms that do not fit our setting [3]). As mentioned earlier, when revote is allowed, our result does not get rid of the replication operator. Bounding the number of voters is still useful in that case, as it simplifies our models. More importantly, bounding the number of ballots means we can encode the ballot box as a fixed-length list, which is very helpful as Proverif does not support arbitrary length lists. We also performed case studies considering a dishonest ballot box. In that case, our Proverif models are more complex, and we rely on a recent extension of the tool that goes beyond the notion of diff-equivalence [35]. Note that this extension of Proverif is a prototype, which is currently unable to exhibit attack traces. Therefore, in case the equivalence does not hold, the result returned is always "cannot be proved".

In some cases, we made slight adjustments to the protocols, so that they fit our framework. Detailed explanations on these modelling choices can be found in the files. All model files for our case study are available at [36]. The results are presented in Table 1.

Overall, as can be seen in the table, our result allows for efficient verification of all protocols we considered. Thanks to the small bounds we establish, we get even better performance than previous work [6] in scenarios where that result applies – *i.e.* the first column, for multiset counting. In that case, some analyses took several hours/days in [6], due to the higher bounds. Our result is more general and can handle *e.g.* STV counting. On most tested protocols, performance remains acceptable in that case. However Proverif did not terminate on three files after 1h: this is likely due to the combination of the complex equational theories used by these protocols, and the theory for STV, which is itself large. As expected, we find the attack on Helios from [5].

8. Conclusion

We have proposed a symbolic version of the BPRIV vote privacy notion, and established reduction results that help us efficiently verify it on several voting protocols, with different counting functions, using automated tools. We have shown how to extend it to produce a symbolic vote privacy notion against an untrusted ballot box in the symbolic model, and proved that our reduction results still hold in that setting.

As mentioned earlier, a limitation of our definition is the modelling of the correct tallying proofs, which we abstracted away. In the computational definition, they are handled using simulators. It remains to be seen whether such techniques can be adapted to the symbolic setting, and how.

Vote privacy is considered a fundamental security property for electronic voting schemes. It is of course not the only desirable one: in particular, receipt-freeness and coercion-resistance can be seen as stronger variants of privacy, that require that an attacker should be unable to ascertain

[3] While Civitas does support revote, it uses rather complex mechanisms in that case to determine which ballot replaces which, that we chose not to model in our case study.

the voters' choice, even when they are willing, or coerced, to reveal their vote. Computational game-based definitions [30] as well as symbolic ones [3] have been proposed for these properties. They are however written in the same spirit as SWAP. Proposing formalisations in the style of BPRIV, and establishing similar reduction results to ours for these properties are open questions for future work.

References

[1] B. Adida, Helios: Web-based Open-Audit Voting, in: *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, P.C. van Oorschot, ed., USENIX Association, 2008, pp. 335–348.

[2] J.C. Benaloh and M. Yung, Distributing the Power of a Government to Enhance the Privacy of Voters (Extended Abstract), in: *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing, Calgary, Alberta, Canada, August 11-13, 1986*, J.Y. Halpern, ed., ACM, 1986, pp. 52–62.

[3] S. Delaune, S. Kremer and M.D. Ryan, Verifying Privacy-type Properties of Electronic Voting Protocols, *Journal of Computer Security* **17**(4) (2009), 435–487.

[4] M. Backes, C. Hritcu and M. Maffei, Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-Calculus, in: *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, USA, 23-25 June 2008*, IEEE Computer Society, 2008, pp. 195–209.

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

[6] M. Arapinis, V. Cortier and S. Kremer, When are three voters enough for privacy properties?, in: *Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS'16)*, LNCS, Springer, 2016.

[7] V. Cortier and C. Wiedling, A formal analysis of the Norwegian E-voting protocol, *Journal of Computer Security* **25**(15777) (2017), 21–57.

[8] V. Cortier, D. Galindo and M. Turuani, A formal analysis of the Neuchâtel e-voting protocol, in: *3rd IEEE European Symposium on Security and Privacy (Euro S&P'18)*, London, UK, 2018, pp. 430–442.

[9] D.A. Basin, S. Radomirovic and L. Schmid, Alethea: A Provably Secure Random Sample Voting Protocol, in: *31st IEEE Computer Security Foundations Symposium, (CSF'18)*, IEEE Computer Society, 2018.

[10] J. Benaloh, Verifiable secret-ballot elections, PhD thesis, Yale University, 1987.

[11] D. Bernhard, V. Cortier, D. Galindo, O. Pereira and B. Warinschi, A comprehensive analysis of game-based ballot privacy definitions, in: *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P'15)*, IEEE Computer Society Press, San Jose, CA, USA, 2015.

[12] V. Cortier, J. Lallemand and B. Warinschi, Fifty Shades of Ballot Privacy: Privacy against a Malicious Board, in: *33rd IEEE Computer Security Foundations Symposium (CSF'20)*, Boston, USA, 2020.

[13] S. Delaune and L. Hirschi, A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols, *Journal of Logical and Algebraic Methods in Programming* **87** (2017), 127–144.

[14] S. Mödersheim and L. Viganò, Alpha-Beta Privacy, *ACM Trans. Priv. Secur.* **22**(1) (2019), 7:1–7:35.

[15] B. Blanchet, An Efficient Cryptographic Protocol Verifier Based on Prolog Rules, in: *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, IEEE Computer Society, Cape Breton, Nova Scotia, Canada, 2001, pp. 82–96.

[16] B. Blanchet, Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif, *Foundations and Trends in Privacy and Security* **1**(1–2) (2016), 1–135.

[17] S. Meier, B. Schmidt, C. Cremers and D. Basin, The TAMARIN Prover for the Symbolic Analysis of Security Protocols, in: *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, LNCS, Vol. 8044, Springer, 2013, pp. 696–701.

[18] B. Blanchet, M. Abadi and C. Fournet, Automated Verification of Selected Equivalences for Security Protocols, in: *20th IEEE Symposium on Logic in Computer Science (LICS 2005)*, IEEE Computer Society, Chicago, IL, 2005, pp. 331–340.

[19] D.A. Basin, J. Dreier and R. Sasse, Automated Symbolic Proofs of Observational Equivalence, in: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, I. Ray, N. Li and C. Kruegel, eds, ACM, 2015, pp. 1144–1155.

[20] V. Cortier, A. Filipiak and J. Lallemand, BeleniosVS: Secrecy and Verifiability Against a Corrupted Voting Device, in: *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, IEEE, 2019.

[21] B. Blanchet and B. Smyth, Automated reasoning for equivalences in the applied pi calculus with barriers, *Journal of Computer Security* **26**(3) (2018), 367–422.
[22] H. Comon-Lundh and V. Cortier, Security properties: two agents are sufficient, in: *Proc. 12th European Symposium on Programming (ESOP'03)*, LNCS, Vol. 2618, Springer, Warsaw, Poland, 2003, pp. 99–113.
[23] V. Cortier, A. Dallon and S. Delaune, Bounding the number of agents, for equivalence too, in: *Proc. 5th International Conference on Principles of Security and Trust (POST'16)*, LNCS, Springer, 2016, pp. 211–232.
[24] S. Fröschle, Leakiness is Decidable for Well-Founded Protocols?, in: *Proc. 4th Conference on Principles of Security and Trust (POST'15)*, LNCS, Springer, 2015.
[25] E. D'Osualdo, L. Ong and A. Tiu, Deciding Secrecy of Security Protocols for an Unbounded Number of Sessions: The Case of Depth-Bounded Processes, in: *Proc. 30th Computer Security Foundations Symposium, (CSF'17)*, IEEE Computer Society, 2017, pp. 464–480.
[26] S. Delaune and J. Lallemand, One Vote Is Enough for Analysing Privacy, in: *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part I*, V. Atluri, R.D. Pietro, C.D. Jensen and W. Meng, eds, Lecture Notes in Computer Science, Vol. 13554, Springer, 2022, pp. 173–194. doi:10.1007/978-3-031-17140-6_9.
[27] M. Abadi and C. Fournet, Mobile values, new names, and secure communication, in: *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, C. Hankin and D. Schmidt, eds, ACM, 2001, pp. 104–115.
[28] V. Cortier, P. Gaudry and S. Glondu, Belenios: A Simple Private and Verifiable Electronic Voting System, in: *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*, LNCS, Vol. 11565, Springer, 2019, pp. 214–238.
[29] P. Chaidos, V. Cortier, G. Fuchsbauer and D. Galindo, BeleniosRF: A Non-interactive Receipt-Free Electronic Voting Scheme, in: *23rd ACM Conference on Computer and Communications Security (CCS'16)*, ACM, Vienna, Austria, 2016, pp. 1614–1625.
[30] A. Juels, D. Catalano and M. Jakobsson, Coercion-Resistant Electronic Elections, in: *Towards Trustworthy Elections, New Directions in Electronic Voting*, D. Chaum, M. Jakobsson, R.L. Rivest, P.Y.A. Ryan, J. Benaloh, M. Kutylowski and B. Adida, eds, LNCS, Vol. 6000, Springer, 2010, pp. 37–63.
[31] D. Chaum, P.Y.A. Ryan and S.A. Schneider, A Practical Voter-Verifiable Election Scheme, in: *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, S.D.C. di Vimercati, P.F. Syverson and D. Gollmann, eds, LNCS, Vol. 3679, Springer, 2005, pp. 118–139.
[32] P.Y.A. Ryan and S.A. Schneider, Prêt à Voter with Re-encryption Mixes, in: *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, D. Gollmann, J. Meier and A. Sabelfeld, eds, LNCS, Vol. 4189, Springer, 2006, pp. 313–326.
[33] M.R. Clarkson, S. Chong and A.C. Myers, Civitas: Toward a Secure Voting System, in: *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, IEEE Computer Society, 2008, pp. 354–368.
[34] P.Y.A. Ryan and S.A. Schneider, Prêt à Voter with Re-encryption Mixes, in: *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, D. Gollmann, J. Meier and A. Sabelfeld, eds, LNCS, Vol. 4189, Springer, 2006, pp. 313–326.
[35] V. Cheval and I. Rakotonirina, Indistinguishability Beyond Diff-Equivalence in ProVerif, in: *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*, IEEE, 2023, pp. 184–199. doi:10.1109/CSF57540.2023.00036.
[36] S. Delaune, J. Lallemand and A. Outrey, One Vote is Enough for Analysing Privacy, Technical Report, CNRS, 2023. https://hal.science/hal-04262499.