HAL
open science

Improving Automated Symbolic Analysis of Ballot Secrecy for E-Voting Protocols: A Method Based on Sufficient Conditions

Lucca Hirschi, Cas Cremers

To cite this version:

Lucca Hirschi, Cas Cremers. Improving Automated Symbolic Analysis of Ballot Secrecy for E-Voting Protocols: A Method Based on Sufficient Conditions. EuroS&P 2019 - 4th IEEE European Symposium on Security and Privacy, Jun 2019, Stockholm, Sweden. pp.635-650, 10.1109/EuroSP.2019.00052 . hal-02368857

HAL Id: hal-02368857

https://inria.hal.science/hal-02368857v1

Submitted on 26 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Improving Automated Symbolic Analysis of Ballot Secrecy for E-voting Protocols: A Method Based on Sufficient Conditions

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

Cas Cremers
CISPA Helmholtz Center (i.G.)
Saarbruecken, Germany
cremers@cispa.saarland

***Abstract*—We advance the state-of-the-art in automated symbolic analysis of ballot secrecy for e-voting protocols by proposing a method based on analysing three conditions that together imply ballot secrecy.**

Our approach has two main advantages over existing automated approaches. The first is a substantial expansion of the class of protocols and threat models that can be automatically analysed: our approach can systematically deal with (a) honest authorities present in different phases, (b) threat models in which no dishonest voters occur, and (c) protocols whose ballot secrecy depends on fresh data coming from other phases. The second advantage is that our approach can significantly improve verification efficiency, as the individual conditions are often simpler to verify. E.g., for the LEE protocol, we obtain a speedup of over two orders of magnitude.

We show the scope and effectiveness of our approach using ProVerif in several case studies, including the FOO, LEE, JCJ, and Belenios protocols.

I. Introduction

There have been substantial advances during the last years in the field of e-voting protocols. Many new approaches have been developed, and the relevant security properties have become better understood and agreed upon [1]–[4]. One of the main properties is that voters' votes remain private, which is known as *ballot secrecy*. Designing protocols that achieve this has proven subtle: many vulnerabilities have been found in previously proposed protocols [3], [5], motivating the need for improved analysis techniques to support the development of e-voting systems. Unfortunately, the complexity of e-voting systems makes *computational proofs* hard, e.g., the computational proof of Helios from [6] required one person-year.

For classical security protocols, there is mature tool support in the *symbolic model* [7]–[10], which enables detecting many flaws during the protocol design phase, or later, as new threat models are considered. Verification in this more abstract model allows for a high level of automation. This notably enables security analyses exploring various threat models in order to provide more fine-grained guarantees (see *e.g.* [11]–[13]). However, these tools traditionally did not handle e-voting protocols [14]. Recently, new symbolic methods have been proposed [15]–[19] to analyse e-voting protocols. However, the applicability of these methods is still extremely limited both in the type of protocols that they can deal with and the type of security properties (including threat models) that they analyse (as acknowledged by [14], [17], [20]).

The reasons for these limitations interact in a complex way with existing approaches. One of the main reason though is that ballot secrecy is a behavioural equivalence-based property which is notoriously more difficult to analyse than the more classical reachability properties. Two effective tools that can prove such equivalence properties for an unbounded number of sessions are ProVerif [9] and Tamarin [7]. These tools can deal with many typical primitives that are used in e-voting protocols [15], [16], [18], [21]. However, they check for an abstraction of equivalence (*i.e. diff-equivalence*) that is rarely met by typical encodings of e-voting protocols and ballot secrecy. Thus, in most cases, the analysis results in a *spurious attack* (i.e., an attack that is an artefact of the abstraction and not a real attack on the protocol), and no conclusion can be drawn about the protocol.

Despite recent efforts to improve the accuracy of the equivalence being checked (*e.g.*, the swapping technique [16], [18] and the small-attack property [17]), this still effectively limits the class of e-voting protocols and the threat models to which existing tools can be successfully applied. More precisely, we have identified the following limitations from analysing several case studies and threat models:

(a) Spurious attacks when *honest authorities are present in different phases* of the voting process. For many threat models, this excludes modelling a registrar that distributes credentials in a registration phase and then commits credentials of eligible voters to the ballot box in a later phase, as in JCJ [22] and Belenios [23].
(b) Spurious attacks with ProVerif when ballot secrecy notably relies on the *freshness of some data coming from previous phases*. For example, such data can be credentials created

This work was conducted when C. C. was at University of Oxford, UK and when L. H. was working at LSV, CNRS & ENS Cachan, France and then at ETH Zurich, Switzerland. This work was partially supported by a COST grant (COST-STSM-IC1306-33371) granted by the European Cooperation in Science and Technology (Crypto Action) and a mobility grant granted by the Doctoral School of the Paris-Saclay University.

during a registration phase, as in JCJ and Belenios.

(c) Spurious attacks for threat models in which *no dishonest voter* is assumed (we will explain later why this is a more complex case than with dishonest voters that we handle as well).

(d) The current techniques have *scalability issues* (for reasons explained later). For instance, we were not able to obtain results in less than 2 days for the simple protocol LEE [24].

Contributions. In this work, we advance the state-of-the art in automated symbolic verification of ballot secrecy in e-voting protocols. Our key idea is to soundly modularize ballot secrecy verification. We develop three tight conditions on e-voting protocols and prove that, together, they imply ballot secrecy. The three conditions in our theorem are inspired by our analysis of the different types of attacks on ballot secrecy. Since each condition focuses on one specific aspect of ballot secrecy, it is typically simpler to analyse the combination of the three conditions than to verify ballot secrecy directly, as was done in prior works. Our conditions and our analysis algorithm give rise to a new method to verify ballot secrecy, improving the state of the art in several aspects.

First, our approach expands the class of protocols and threat models that can be automatically analysed. We notably address the limitations of the state-of-the-art (a-c) mentioned above. As demonstrated by our case studies, providing support for such features is essential for considering flexible threat models and for establishing more precise security guarantees that also take important practical aspects of protocols into account, such as authentication or registration phases, which are often not considered in the literature.

Second, our approach can significantly improve verification efficiency (d). The increased efficiency can occur for two main reasons. First, because each of our conditions focus on one aspect of the problem and simplifies parts not related to that aspect, it involves smaller processes that are typically easier to verify. Second, previous techniques such as the swapping technique suffer from an exponential blow up related to the number of processes in each phase. In practice, we typically observe a speedup of over two orders of magnitude and even cause the analysis to terminate in cases where it did not do so before.

Third, we use our approach to analyse several new case studies. Thanks to the flexibility and the large class of protocols we can deal with, we are able to analyse a multitude of different threat models allowing comprehensive comparisons. Moreover, thanks to the aforementioned advantages, our approach is able to systematically take the registration phase into account, whereas prior works often consider registrars as fully honest and not model them. We successfully automatically analysed the FOO, Lee, JCJ, and Belenios protocols with respect to various threat models. We show that our theorem also applies to the Okamoto protocol.

We also revisit the state-of-the-art definition of ballot secrecy [24], [25] and propose a more accurate variant (*i.e.* sound, with less spurious attacks) of ballot secrecy whose automated verification does not rely on synchronisation barriers [16], [18], [24]–[26], which was one of the cause of limitations (a) and (c).

While we present our work in the ProVerif framework, our results are applicable beyond this specific tool. Indeed, our conditions and our Main Theorem are stated in a standard applied π-calculus framework. We also believe that our conditions shed light on three crucial aspects that e-voting protocols should enforce; thus improving our understanding of the complex notion ballot secrecy. Finally, the fact that our approach is effective for the analysis of ballot secrecy also suggests that it may be possible to improve the analysis of other e-voting requirements by adopting a similar strategy.

Outline. We first provide intuition for our approach in Section II. In Section III, we present the symbolic model we use to represent protocols and security properties. We then describe our framework in Section IV, notably defining ballot secrecy and the class of e-voting protocols that we deal with. Next, we formally define our conditions and state our Main Theorem in Section V. We show the practicality of our approach in Section VI by explaining how to verify our conditions and presenting case studies. Finally, we discuss related work in Section VII and conclude in Section VIII.

Extended version with Supplementary Material. [27] contains supplementary material, notably full proofs, additional examples, and, further detail on case studies.

II. Intuition behind our approach

a) ***Links & Ballot Secrecy:*** Ballot secrecy boils down to ensuring that an attacker cannot establish a meaningful link between a specific voter and a specific vote (that this voter is willing to cast). For instance, a naive example of such a link occurs when a voter outputs a signed vote in the clear, explicitly linking his vote to his identity. However, in more realistic e-voting protocols, such links can be very complex, possibly relying on long transitive chains of different pieces of data from different outputs. For example, if an attacker is able to link a credential with the identity of the recipient voter during a registration phase, and then the voter anonymously sends his vote along with the credential during a casting phase, then the attacker may be able to link the vote to the voter.

As noted before, diff-equivalence (as an under-approximation of behavioural equivalence) is rarely appropriate to directly verify ballot secrecy [14], [16]. An underlying reason for this is that considering diff-equivalence gives the attacker more additional structural links than when considering the intended behavioural equivalence. This often leads to spurious attacks.

b) ***Informal Presentation of the Conditions:*** We analysed typical attacks and the underlying links. We classified them and identified three classes of links leading to privacy breaches. The purpose of each of our conditions is to guarantee the absence of links from the corresponding class. Our Main Theorem states that together, the three conditions suffice to ensure ballot secrecy.

(Dishonest Condition) By adopting a malicious behaviour, the attacker may be able to link messages that would not be linkable in the intended, honest execution. For instance, if the attacker sends tampered data to a voter, the attacker may be able to later observe the tampered part in different messages, and

conclude that it comes from the same voter, which allows the attacker to establish possibly harmful links. Our first condition essentially requires that a voting system is indistinguishable for the attacker from a voting system in which at the beginning of each phase, all agents forget everything about the past phases and pretend that everything happened as expected, *i.e.*, as in an honest execution. The previous example would violate the condition, because in the second system, the attacker would not be able to observe the tainted data. Interestingly, this condition is mostly a reachability property that does not suffer from the lack of precision of diff-equivalence.

(Honest Relations Condition) Even in the expected honest execution, the attacker may be able to exploit useful links. Thanks to the previous condition, we can focus on a system where each role is split into sub-roles for each phase. This allows us to verify the absence of the former relations using diff-equivalence, without giving the attacker spurious structural links, as mentioned above.

(Tally Condition) We take into account the tally outcome, which enables establishing more links. Typically, the attacker may link an identity to a vote if it can forge valid ballots related to (*i.e.*, containing the same vote) data that can be linked to an identity. This introduces a bias in the tally outcome that can reveal the vote in the forged ballot. This attack class strictly extends *ballot independence attacks* [3]. The Tally Condition requires that when a valid ballot was forged by the attacker then it must have been forged without meaningfully using voter's data already linked to an identity.

III. Model

We model security protocols using the standard process algebra in the style of the dialect of Blanchet *et al.* [28] (used in the ProVerif tool), that is inspired by the applied π-calculus [29]. Participants are modelled as processes, and the exchanged messages are modelled using a term algebra.

Since most of the e-voting protocols are structured in a sequence of *phases* (*e.g.registration phase*, *voting phase*, *tallying phase*), our model includes explicit phases. We briefly present this model in this section; full details are given in [27].

a) ***Term algebra****:* We use a term algebra to model messages built and manipulated using various cryptographic primitives. We assume an infinite set $\mathcal{N}$ of *names*, used to represent keys and nonces; and two infinite and disjoint sets of *variables* $\mathcal{X}$ (to refer to unknown parts of messages expected by participants) and $\mathcal{W}$ (called *handles*, used to store messages learned by the attacker). We consider a *signature* Σ (*i.e.* a set of function symbols with their arity). Σ is the union of two disjoint sets: the *constructor* Σ_c and *destructor* Σ_d symbols. Given a signature $\mathcal{F}$, and a set of atoms A, we denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of terms built using atoms from A and function symbols from $\mathcal{F}$. The terms in $\mathcal{T}(\Sigma_c, \mathcal{N})$ are called *messages*. Sequences of elements are shown bold (*e.g.* $\mathbf{x}, \mathbf{n}$). The application of a substitution σ to a term u is written $u\sigma$, and $\mathrm{dom}(\sigma)$ denotes its *domain*.

As in the process calculus presented in [28], messages are subject to an equational theory used for for modelling algebraic properties of cryptographic primitives. Formally, we consider a congruence $=_{\mathsf{E}}$ on $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$, generated from a set of equations E over $\mathcal{T}(\Sigma_c, \mathcal{X})$. We say that a function symbol is *free* when it does not occur in E. We assume the existence of a *computation relation* $\downarrow$: $\mathcal{T}(\Sigma, \mathcal{N}) \times \mathcal{T}(\Sigma_c, \mathcal{N})$ that gives a meaning to destructor symbols. In [27], we describe how this relation can be obtained from *rewriting systems* and give a full example. For modelling purposes, we also split the signature Σ into two parts, namely Σ_{pub} (public function symbols, known by the attacker) and Σ_{priv} (private function symbols). An attacker builds his own messages by applying public function symbols to terms he already knows and that are available through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe* (noted R), *i.e.*, a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W})$.

$$
\begin{array}{rcll}
P, Q & := & 0 & \text{null} \\
 & | & \mathtt{in}(c, x).P & \text{input} \\
 & | & \mathtt{out}(c, u).P & \text{output} \\
 & | & \mathsf{let}\ x = v\ \mathtt{in}\ P\ \mathsf{else}\ Q & \text{evaluation} \\
 & | & P \mid Q & \text{parallel} \\
 & | & \nu n.P & \text{restriction} \\
 & | & !P & \text{replication} \\
 & | & i : P & \text{phase}
\end{array}
$$

where $c \in \mathcal{C}$, $x \in \mathcal{X}$, $n \in \mathcal{N}$, $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$, $i \in \mathbb{N}$, and $v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$.

Fig. 1. Syntax of processes

Example 1. Consider the signature

$\Sigma_c = \{\mathsf{eq}, \langle\,\rangle, \mathsf{sign}, \mathsf{pkv}, \mathsf{blind}, \mathsf{unblind}, \mathsf{commit}, \mathsf{ok}\}$

$\Sigma_d = \{\mathsf{verSign}, \mathsf{open}, \pi_1, \pi_2, \mathsf{eq}\}$.

The symbols $\mathsf{eq}, \langle\rangle, \mathsf{sign}, \mathsf{verSign}, \mathsf{blind}, \mathsf{unblind}, \mathsf{commit}$ and open have arity 2 and represent equality test, pairing, signature, signature verification, blind signature, unblind, commitment and commitment opening. The symbols π_1, π_2 and pkv have arity 1 and represent projections and the agents' verification keys. Finally, ok is a constant symbol (*i.e.* arity 0). To reflect the algebraic properties of the blind signature, we may consider $=_{\mathsf{E}}$ generated by the following equations:

$$
\begin{array}{rcl}
\mathsf{unblind}(\mathsf{sign}(\mathsf{blind}(x_m, y), z_k), y) & = & \mathsf{sign}(x_m, z_k) \\
\mathsf{unblind}(\mathsf{blind}(x_m, y), y) & = & x_m.
\end{array}
$$

Symbols in Σ_d can be given a semantics through the following rewriting rules: $\mathsf{verSign}(\mathsf{sign}(x_m, z_k), \mathsf{pkv}(z_k)) \rightarrow x_m$, $\mathsf{open}(\mathsf{commit}(x_m, y), y) \rightarrow x_m$, $\pi_i(\langle x_1, x_2\rangle) \rightarrow x_i$, $\mathsf{eq}(x, x) \rightarrow \mathsf{ok}$.

b) ***Process algebra****:* We assume $\mathcal{C}_{\mathsf{pub}}$ and $\mathcal{C}_{\mathsf{priv}}$ are disjoint sets of public and private channel names and note $\mathcal{C} = \mathcal{C}_{\mathsf{pub}} \cup \mathcal{C}_{\mathsf{priv}}$. Protocols are specified using the syntax in Figure 1. Most of the constructions are standard. The construct $\mathsf{let}\ x = v\ \mathtt{in}\ P\ \mathsf{else}\ Q$ tries to evaluate the term v and in case of success, *i.e.* when $v \downarrow u$ for some message u, the process P in which x is substituted by u is executed; otherwise the process Q is executed. Note also that the let instruction together with the eq theory (see Example 1) can encode the usual conditional construction. The replication $!P$ behaves like an infinite parallel

composition $P|P|P|\ldots$. The construct $i : P$ indicates that the process P may only be executed when the current phase is i. The construct $\nu n.P$ allows to create a new, fresh name n; it binds n in P which is subject to α-renaming. For a sequence of names $\mathbf{n}$, we may note $\nu\mathbf{n}.P$ to denote the sequence of creation of names in $\mathbf{n}$ followed by P. For brevity, we sometimes omit "else 0" and null processes at the end of processes. A process P is *ground* if it has no free variable (*i.e.*, a variable not in the scope of an input or a let construct). A process is *guarded* if it is of the form $i : P$.

The operational semantics of processes is given by a labelled transition system over *configurations* (denoted by K) $(\mathcal{P}; \phi; i)$ made of a multiset $\mathcal{P}$ of guarded ground processes, $i \in \mathbb{N}$ the current phase, and a *frame* $\phi = \{w_j \mapsto u_j\}_{j\in\mathcal{J}}$ (*i.e.* a substitution where $\forall j \in \mathcal{J}, w_j \in \mathcal{W},\ u_j \in \mathcal{T}(\Sigma_c, \mathcal{N})$). The frame ϕ represents the messages known to the attacker. Given a configuration K, $\phi(K)$ denotes its frame. We often write $P \cup \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$ and implicitly remove null processes from configurations.

The operational semantics of a process is given by the relation $\xrightarrow{\alpha}$ defined as the least relation over configurations satisfying the rules in Figure 2. For all constructs, phases are just handed over to continuation processes. The rules are quite standard and correspond to the intuitive meaning of the syntax given above. The rules IN,OUT,NEXT are the only rules that produce observable actions (*i.e.*, non τ-actions). The relation $\xrightarrow{\alpha_1\ldots\alpha_n}$ between configurations (where α_i are actions) is defined as the transitive closure of $\xrightarrow{\alpha}$.

Example 2. We use the FOO protocol [30] (modelled as in [16]) as a running example. FOO involves voters and a registrar role. In the first phase, a voter commits and then blinds its vote and sends this blinded commit signed with his own signing key $\mathsf{key}(\mathrm{id})$ to the registrar. The function symbol $\mathsf{key}(\cdot)$ is a free private function symbol associating a secret key to each identity. The registrar then blindly signs the committed vote with his own signing key $k_R \in \Sigma_c \cap \Sigma_{\mathsf{priv}}$ and sends this to the voter. In the voting phase, voters anonymously output their committed vote signed by the registrar and, on request, anonymously send the opening for their committed vote. The process corresponding to a voter session (depending on some constants id, v) is depicted below, where $c \in \mathcal{C}_{\mathsf{pub}}$, $M = \mathsf{commit}(v, k)$, $e = \mathsf{blind}(M, k')$ and $s = \mathsf{sign}(e, \mathsf{key}(\mathrm{id}))$:

$$\begin{aligned} V(\mathrm{id}, v) = 1 : \nu k.\nu k'.&\mathtt{out}(c, \langle \mathsf{pk}(\mathsf{key}(\mathrm{id})); s\rangle).\mathtt{in}(c, x).\\ &\text{if } \mathsf{verSign}(x, \mathsf{pk}(k_R)) = e\\ &\text{then } 2 : \mathtt{out}(c, \mathsf{unblind}(x, k')).\mathtt{in}(c, y).\\ &\text{if } y = \langle y_1; M\rangle\\ &\text{then } \mathtt{out}(c, \langle y_1, \langle M, k\rangle\rangle) \end{aligned}$$

A configuration corresponding to a voter A ready to vote v_1 with an environment knowing the registrar's key is $K_1 = (\{V(A, v_1)\}; \{w_R \mapsto k_R\}; 1)$. It notably has an execution $K_1 \xrightarrow{\mathsf{tr}_h} (\emptyset; \phi; 2)$, where:

$$\begin{aligned} \mathsf{tr}_h = \ & \tau.\tau.\mathtt{out}(c, w_1).\mathtt{in}(c, R).\tau_{\mathsf{then}}.\tau.\mathtt{phase}(2).\\ & \mathtt{out}(c, w_2).\mathtt{in}(c, \langle C, w_2\rangle).\tau_{\mathsf{then}}.\mathtt{out}(c, w_3) \end{aligned}$$

and where C is any constant in $\Sigma_c \cap \Sigma_{\mathsf{pub}}$, $\phi = \{w_R \mapsto k_R, w_1 \mapsto \langle \mathsf{pk}(k_{\mathrm{id}}), s\rangle, w_2 \mapsto \mathsf{sign}(M, k_R), w_3 \mapsto \langle n; M; k\rangle\}$, s, M are as specified above and $R = \mathsf{sign}(\mathsf{verSign}(\pi_2(w_1), \pi_1(w_1)), w_R)$. This corresponds to a normal, expected execution of one protocol session.

c) ***Discussion on Phases****:* Our notion of phases, also known as *stages* or *weak phase* [26], [28], faithfully model the notion of phases with deadlines in the context of e-voting protocols. Once the deadline of a phase i has passed (*i.e.* the action $\mathtt{phase}(j)$ has been triggered for $j > i$) then, no remaining actions from phase i can be executed. It also can be modelled in ProVerif (see [15], [26], [28], [31]). Note that in the literature, phases are often modelled with *synchronisation barriers* [16], [26] (also called *strong phases*). The latter are a much stronger notion of phases that require all initial processes to reach the next phase before the system can progress to the next phase (i.e., no processes can be dropped). In our view, synchronisation barriers model phases in e-voting protocols less faithfully than our (weak) phases, and come with limitations that we discuss in Section IV-B. We note that *stages* can be combined with replication without restriction while *strong phases* cannot be put under replication [16], [31].

d) ***Trace equivalence****:* Trace equivalence is commonly used [14] to express many privacy-type properties such as ballot secrecy. Intuitively, two configurations are trace equivalent if an attacker cannot tell whether he is interacting with one or the other. Such a definition is based on a notion of indistinguishability between frames, called *static equivalence*. Intuitively, two frames are statically equivalent, if there is no computation (nor equality test) that succeeds in one frame and fails in the other one. Then, *trace equivalence* is the active counterpart taking into account the fact that the attacker may interfere during the execution of the process in order to distinguish between the two situations. We define $\mathsf{obs}(\mathsf{tr})$ to be the subsequence of tr obtained by erasing all the $\tau, \tau_{\mathsf{then}}, \tau_{\mathsf{else}}$ actions. Intuitively, trace equivalence holds when any execution of one configuration can be mimicked by an execution of the other configuration having same observable actions and leading to statically equivalent frames. We give a formal definition in [27].

Example 3. Consider the frame ϕ from Example 2. The fact that the attacker cannot *statically* distinguish the resulting frame from a frame obtained after the same execution but starting with $V(A, v_2)$ instead of $V(A, v_1)$ is modelled by the following static equivalence: $\phi \sim^? \phi'$ where $\phi' = \phi\{v_1 \mapsto v_2\}$ which in fact does not hold (see witness given in [27]). Consider $K_i = (\{V(A, v_i)\}; \{w_R \mapsto k_R\}; 1)$ for $i \in \{1, 2\}$. We may be interested whether $K_1 \approx^? K_2$. This equivalence does not hold because there is only one execution starting with K_1 (resp. K_2) following the trace $\mathsf{obs}(tr_h)$ (see Example 2) and the resulting frame is ϕ (resp. ϕ'). But, as shown above, $\phi \not\sim \phi'$. Therefore, $K_1 \not\approx K_2$. However, ballot secrecy is not defined by such an equivalence (see Section IV-B) and we will see that the FOO protocol actually satisfies it.

e) ***Diff-Equivalence****:* Trace equivalence is hard to verify, in particular because of its forall-exists structure: for any execution

$$
\begin{array}{lll}
\textsc{In} & (i : \mathtt{in}(c,x).P \cup \mathcal{P}; \phi; i) \xrightarrow{\mathtt{in}(c,R)} (i : P\{x \mapsto u\} \cup \mathcal{P}; \phi; i) & \\
& \quad \text{with } c \in \mathcal{C}_{\mathsf{pub}} \text{ where } R \text{ is a recipe such that } R\phi{\downarrow}u \text{ for some message } u & \\
\textsc{Out} & (i : \mathtt{out}(c,u).P \cup \mathcal{P}; \phi; i) \xrightarrow{\mathtt{out}(c,w)} (i : P \cup \mathcal{P}; \phi \cup \{w{\mapsto}u\}; i) & \\
& \quad \text{with } c \in \mathcal{C}_{\mathsf{pub}} \text{ and } w \text{ a fresh variable in } \mathcal{W} & \\
\textsc{Com} & (i : \mathtt{in}(c,x).P \cup i : \mathtt{out}(c,u).Q \cup \mathcal{P}; \phi; i) \xrightarrow{\tau} (i : P\{x \mapsto u\} \cup i : Q \cup \mathcal{P}; \phi; i) & \\
& \quad \text{with } c \in \mathcal{C}_{\mathsf{priv}} & \\
\textsc{Let} & (i : \mathsf{let}\ x = v\ \mathtt{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P}; \phi; i) \xrightarrow{\tau_{\mathsf{then}}} (i : P\{x \mapsto u\} \cup \mathcal{P}; \phi; i) & \\
& \quad \text{when } v{\downarrow}u \text{ for some } u & \\
\textsc{Let-Fail} & (i : \mathsf{let}\ x = v\ \mathtt{in}\ P\ \mathsf{else}\ Q \cup \mathcal{P}; \phi; i) \xrightarrow{\tau_{\mathsf{else}}} (i : Q \cup \mathcal{P}; \phi; i) & \text{when } v{\not\downarrow} \\
\textsc{New} & (i : \nu n.P \cup \mathcal{P}; \phi; i) \xrightarrow{\tau} (i : P \cup \mathcal{P}; \phi; i) & \text{where } n \text{ is a fresh name from } \mathcal{N} \\
\textsc{Next} & (\mathcal{P}; \phi; i) \xrightarrow{\mathtt{phase}(j)} (\mathcal{P}; \phi; j) & \text{for some } j \in \mathbb{N} \text{ such that } j > i \\
\textsc{Par} & (\{i : (P_1 \mid P_2)\} \cup \mathcal{P}; \phi; i) \xrightarrow{\tau} (\{i : P_1, i : P_2\} \cup \mathcal{P}; \phi; i) &
\end{array}
$$

$$
\textsc{Phase}\quad (i : j : P \cup \mathcal{P}; \phi; i) \xrightarrow{\tau} (j : P \cup \mathcal{P}; \phi; i) \qquad \textsc{Repl}\quad (i : !P \cup \mathcal{P}; \phi; i) \xrightarrow{\tau} (i : P \cup i : !P \cup \mathcal{P}; \phi; i)
$$

Fig. 2. Semantics for processes

on one side, one has to find a matching execution on the other side. One approach is to consider under-approximations of trace equivalence by over-approximating the attacker's capabilities. *Diff-equivalence* is such an under-approximation. It was originally introduced to enable ProVerif to analyse some form of behavioural equivalence, and was later also implemented in Tamarin and Maude-NPA.

Such a notion is defined on bi-processes, which are pairs of processes with the same structure that only differ in the terms they use. The syntax is similar to above, but each term u has to be replaced by a bi-term written $\mathsf{choice}[u_1, u_2]$ (using ProVerif syntax). Given a bi-process P, the process $\mathsf{fst}(P)$ is obtained by replacing all occurrences of $\mathsf{choice}[u_1, u_2]$ with u_1; similarly with $\mathsf{snd}(P)$. The semantics of bi-processes is defined as expected via a relation that expresses when and how a bi-configuration may evolve. A bi-process reduces if, and only if, both sides of the bi-process reduce in the same way triggering the same rule: *e.g.*, a conditional has to be evaluated in the same way on both sides. The relation $\xrightarrow{\mathsf{tr}}_{\mathsf{bi}}$ on bi-processes is therefore defined as for processes. Finally, diff-equivalence of a biprocess intuitively holds when for any execution, the resulting frames on both sides are statically equivalent and resulting configurations on both sides are able to perform the same kind of actions. A formal definition is given in [27].

As expected, this notion of diff-equivalence is stronger than trace equivalence. It may be the case that the two sides of the bi-process reduce in different ways (*e.g.*, taking two different branches in a conditional) but still produce the same observable actions. Phrased differently: diff-equivalence gives the attacker the ability to see not only the observable actions, but also the processes' structures. This strong notion of diff-equivalence is sufficient to establish some properties but is too strong to be useful for establishing ballot secrecy off-the-shelf (we discuss this at greater length in Section VII).

IV. Framework

In this section we present our framework that we need to establish our results, including definitions for e-voting protocols and ballot secrecy.

a) ***Preliminaries:*** We first define *symbolic traces* which are traces whose recipes are symbolic; *i.e.*, they are from $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W} \cup \xi)$, where ξ is a new set of second-order variables. Intuitively, a symbolic recipe is a partial computation containing unknown parts symbolised by second-order variables. Symbolic traces represent attacker behaviours with non-fully specified recipes. A symbolic trace can be instantiated to a concrete trace by replacing the second-order variables by recipes (*i.e.*, in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W})$). To an honest trace th, we associate a distinguished instantiation called the *idealised trace of* th that can be obtained from th by replacing each variable $Y \in \xi$ by a fixed free, public constant C_Y that we add to $\Sigma_c \cap \Sigma_{\mathsf{pub}}$.

Example 4 (Resuming Example 2)**.** The recipe of the last input of tr_h $\langle C, w_2 \rangle$ could be replaced by the symbolic recipe $\langle X, w_2 \rangle$ with $X \in \xi$ (*i.e.*, reflecting that the choice of C is unimportant) resulting in a symbolic trace th. The idealised trace is $\mathsf{th}\{X \mapsto C_X\}$, where $C_X \in \Sigma_c \cap \Sigma_{\mathsf{pub}}$.

A. Class of e-voting protocols

We explain in this section how we model e-voting protocols and the considered scenarios. Essentially, we may consider an arbitrary number of honest voters plus all necessary authorities (*e.g.*, ballot box, registrar, tally), which can perform an unbounded number of sessions. Depending on the threat model, we also consider an arbitrary number of dishonest voters. We use *role* to refer to a specific role of the protocol, such as voter, authority, etc. Together, the agents performing the roles are able to produce a public bulletin board of ballots from which the tally computes the final result (*i.e.*, multisets of accepted votes).

First, the protocol should specify a fixed finite set of possible votes as a set of free, public constants $\mathcal{V}$ (*e.g.*, $\mathcal{V} = \{\mathtt{yes}, \mathtt{no}\}$ for a referendum). We also distinguish a specific free, public constant $\perp$ modelling the result of an invalid ballot.

a) ***Roles:*** E-voting protocols specify a process for each honest role (in particular, the voter role). Dishonest roles can be left unspecified because they will be played by the environment. Those processes may use *e.g.* phases, private data, private

channels but no replication nor parallel composition, as a role specifies how a *single* agent behaves during *one* session.

Definition 1. An honest role is specified by a process of the form $i : \nu\mathbf{n}.A$, where A is a process without parallel composition, replication nor creation of names. There should be at least a process for the voter role and one for the ballot box role (noted A_b). Moreover, for the specific case of voter role, the corresponding process noted $V(\mathrm{id}, v)$ should be parameterized by id (modelling an identity) and v (modelling the vote this voter is willing to cast). Finally, initial attacker's knowledge is specified through a frame ϕ_0.

The process A_b shall contain (at least) one output on the distinguished public channel $c_b \in \mathcal{C}_{\mathsf{pub}}$. Intuitively, each session of the ballot box processes input data and may output a ballot on channel c_b (this may depend on private checks). We eventually define the bulletin board itself as the set of messages output on channel c_b. W.l.o.g., we assume that role processes do not feature creation of names, since one can always create the required names at the top level.

In threat models with dishonest voters, honest voters are played by the environment and we let $\mathcal{R}_V = \emptyset$. If the considered threat model does not consider dishonest voters, then the honest voters cannot be played by the environment. For such threat models, we model honest voters explicitly using the following set of processes: $\mathcal{R}_V = \{\nu\mathrm{id}.\ \nu\mathbf{n}\ V(\mathrm{id}, v) \mid v \in \mathcal{V}\}$, where $\mathbf{n}$ are all the free names in $V(\mathrm{id}, v)$. We write $\mathcal{R}_o$ for the set of all processes of honest roles except the voter role and let $\mathcal{R}$ be the set $\mathcal{R}_V \cup \mathcal{R}_o$.

Example 5. The process $V(v, \mathrm{id})$ defined in Example 2 is the voter role one could define for the FOO protocol. We consider the ballot box as untrusted, and we therefore model it by the process $A_b = 2 : \mathtt{in}(u, x).\mathtt{out}(c_b, x)$, where $u \in \mathcal{C}_{\mathsf{pub}}$. In contrast, we leave the registrar unspecified for the moment because we consider it corrupted and thus played by the environment.We thus have $\mathcal{R}_V = \emptyset$ and $\mathcal{R} = \mathcal{R}_o = \{A_b\}$. Finally, the initial frame contains the registrar's key: $\phi_0 = \{w_0 \mapsto k_R\}$.

b) ***Bulletin Board & Tally****:* We assume a public test Ψ_b that everyone can execute on the bulletin board to know if a ballot is well-formed or not. Formally $\Psi_b[]$ is a term with a hole. For instance, Ψ_b can be a combination of a signature and ZK proof verification. The protocol should also specify a term with hole $\mathrm{Extract}[]$ that models the extraction of the vote from a valid ballot. As defined below, we require that this operator only computes votes or $\perp$.

Definition 2. The bulletin board and the tally are specified through a public term $\Psi_b[] \in \mathcal{T}(\Sigma_{\mathsf{pub}}, [])$ and a term $\mathrm{Extract}[] \in \mathcal{T}(\Sigma, [])$ such that: for any message t, it holds that $\mathrm{Extract}[t]{\downarrow}u$ for some $u \in \mathcal{V} \cup \{\perp\}$.

Given a trace tr and a frame ϕ, we define respectively the bulletin board and the tally's outcome:

$$\mathrm{BB}(\mathsf{tr}, \phi) = \{w\phi \mid \exists \mathtt{out}(c_b, w) \in \mathsf{tr},\ \Psi_b[w\phi]{\downarrow\downarrow}\}^{\#}$$
$$\mathrm{Res}(\mathsf{tr}, \phi) = \{v \quad \mid \exists \mathrm{ba} \in \mathrm{BB}(\mathsf{tr}, \phi), \mathrm{Extract}(\mathrm{ba}){\downarrow}v \in \mathcal{V}\}^{\#}.$$

The bulletin board is the multiset of messages that pass the Ψ_b condition and channel c_b. Then, the tally's outcome is the multiset of votes obtained by applying $\mathrm{Extract}(\cdot)$ on the bulletin board. While our notion of tally seems very restrictive, note that many operations can be performed by roles (*e.g.* A_b) such as mixnets as done *e.g.* in [16] where the shuffling is done between two phases.

Example 6 (Continuing Example 5)**.** The public test Ψ_b is defined as the following term with hole:

$$\Psi_b[] = \mathsf{and}(\mathsf{verSign}(\pi_1(\pi_2([])), \mathsf{pk}(\mathsf{sk}_R)), \mathsf{open}(\mathsf{getMess}(\pi_1(\pi_2([]))), \pi_2(\pi_2([]))))$$

where the destructor and is such that $\mathsf{and}(t_1, t_2){\downarrow\downarrow}$ if and only if $t_1{\downarrow}$ and $t_2{\downarrow}$ (formal definition in [27]). Indeed, expected ballots are of the form $\langle X, \langle \mathsf{sign}(\mathsf{commit}(k, v), k_R), k\rangle\rangle$. The evaluation of $\Psi_b[b]$ may fail if either the signature verification fails or the commit opening fails. Finally, the extraction function is $\mathrm{Extract}[] = \mathsf{wrapVote}(\mathsf{open}(\mathsf{getMess}(\pi_2(\pi_1([]))), \pi_2(\pi_2([]))))$ where $\mathsf{wrapVote}(\cdot)$ corresponds to the identity function on $\mathcal{V}$ and maps all values not in $\mathcal{V}$ (modulo $=_{\mathsf{E}}$) to $\perp$.

c) ***Honest Trace****:* As said before, no process is given for dishonest roles. However, we require a notion of honest trace that itself specifies what behaviour should be expected from dishonest roles.

Definition 3. The protocol shall specify a symbolic trace $\mathsf{th} = \mathsf{th}^0.\mathtt{out}(c_b, w_b)$ (*i.e.*, the last action corresponds to the casting of a ballot) and a distinguished execution, called the *honest execution*, of the form: $(\{V(\mathrm{id}, v)\} \uplus \mathcal{R}_o; \phi_0; 1) \xrightarrow{\mathsf{tr}_h} (\mathcal{P}; \phi_h; k_f)$ for some $v \in \mathcal{V}$ and a free constant id, with tr_h the idealised trace associated to th. Additionally, we assume that th contains the action $\mathtt{phase}(k)$ for all $2 \leq k \leq k_f$ (no phase is skipped).

The honest trace describes the honest expected execution of one voter completing the voting process until casting a ballot possibly through an interaction with different roles. Here, the notion captures the fact that some corrupted roles are played by the attacker. Hence the fact that the honest trace is a symbolic trace with sub-messages that are unknown and not specified because chosen by the attacker. Note that the honest trace specifies how conditionals are expected to evaluate thanks to the $\tau_{\mathsf{then}}/\tau_{\mathsf{else}}$ dichotomy.

Example 7 (Resuming Example 5)**.** We consider the following extension of the symbolic trace described in Example 4, where $X \in \xi$ and $R_1 = \mathsf{sign}(\mathsf{verSign}(\pi_2(w_1), \pi_1(w_1)), w_R)$:

$$\mathsf{th} = \tau.\tau.\mathtt{out}(c, w_1).\mathtt{in}(c, R_1).\tau_{\mathsf{then}}.\tau.\mathtt{phase}(2).\mathtt{out}(c, w_2).\mathtt{in}(c, \langle X, w_2\rangle).\tau_{\mathsf{then}}.\mathtt{out}(c, w_3).\tau.\mathtt{in}(u, w_3).\mathtt{out}(c_b, w_3)$$

Definition 4 (E-voting Protocols)**.** An e-voting protocol is given by a tuple $(\mathcal{V}; \phi_0; V(\mathrm{id}, v); \mathcal{R}; (\Psi_b[], \mathrm{Extract}[]); \mathsf{th})$ where $\mathcal{V}$ are the allowed votes (*i.e.* free, public constants), $V(\mathrm{id}, V)$ and $\mathcal{R}$ are the processes modelling honest roles and ϕ_0 is the attacker's initial knowledge as in Definition 1, $\Psi_b[]$ and $\mathrm{Extract}[]$ model the bulletin board and the tally as in Definition 2, and th describes the intended, honest execution as in Definition 3.

*d) **Flexible threat models**:* Our generic definition of e-voting protocols allows to model many different threat models. First, the processes that model roles may use different kinds of channels. For instance, by using private channels for some inputs and outputs, we model communication channels that prevent the attacker to eavesdrop on or tamper with those exchanged messages. By using public channels and adding the identity of voter in exchanged data, we model an insecure, non-anonymous communication channel. In contrast, by using only a single public channel, we model an anonymous communication channel, since all voters will use the same channel. Moreover, some roles can be considered dishonest or honest yielding different threat models. Finally, different frames ϕ_0 allow modelling different initial attacker knowledge (*e.g.*, secret keys of some roles).

*e) **Annotated Processes**:* Finally, we equip the semantics with annotations that will help subsequent developments. We assume a notion of annotations over processes so that we can keep track of *role sessions* and *specific voters* throughout executions. Each action can then be labelled by this information. For a voter process $V(\mathrm{id}, v)$, we note $[\mathrm{id}, v]$ the annotation given to actions produced by this process. Formally we may define such annotations by giving explicit annotations to processes in the initial multiset and modify the semantics so that it keeps annotations on processes as one could expect. Those notations notably allow to define when a specific voter casts a ballot as shown next.

Definition 5. Consider an e-voting protocol $(\mathcal{V}; \phi_0; V(\mathrm{id}, v); \mathcal{R}; (\Psi_b[], \mathrm{Extract}[]); \mathsf{th})$. We say that a voter $V(\mathrm{id}, v)$ *casts a ballot* w in an execution $(\mathcal{P} \uplus \{V(\mathrm{id}, v), !A_b\}; \phi_0; 1) \xrightarrow{\mathsf{tr}} K$ when there exists an output $\mathtt{out}(c, w_b) \in \mathsf{tr}$ annotated $[\mathrm{id}, v]$ and a ballot box (*i.e.* A_b) session s_b such that actions from tr annotated s_b are $\mathtt{in}(c, w'_b), \mathtt{out}(c_b, w)$ such that $w_b\phi(K) =_{\mathsf{E}} w'_b\phi(K)$. We say that $V(\mathrm{id}, v)$ *casts a valid ballot* w when, in addition, $\Psi_b[w\phi(K)]\downarrow$.

B. Ballot Secrecy

Next, we define the notion of ballot secrecy that we aim to analyse. Intuitively, ballot secrecy holds when the attacker is not able to observe any difference between two situations where voters are wiling to cast different votes. However, we cannot achieve such a property by modifying just one vote, since the attacker will always be able to observe the difference on the final tally outcome. Example 3 illustrates this problem: one has that $K_1 \not\approx K_2$ while the FOO protocol actually ensures ballot secrecy. Instead, we shall consider a *swap* of votes that preserves the tally's outcome as usually done [24], [25]. More formally, we are interested in comparing $\mathcal{S} = (!\,\mathcal{R}) \uplus \{V(A, v_0), V(B, v_1)\}$ and $\mathcal{S}_r = (!\,\mathcal{R}) \uplus \{V(A, v_1), V(B, v_0)\}$, where v_0, v_1 are two distinct votes in $\mathcal{V}$ and A, B are two distinct free, public constants, and, $!\mathcal{Q}$ refers to $\{!P \mid P \in \mathcal{Q}\}^{\#}$ for a multiset of processes $\mathcal{Q}$. Because the attacker should neither be able to distinguish $\mathcal{S}$ and $\mathcal{S}_r$ when having access to the tally's outcome, we are actually interested in the *trace equivalence* between $\mathcal{S} \cup \{\mathrm{Tally}\}$ and $\mathcal{S}_r \cup \{\mathrm{Tally}\}$ where the Tally is a process computing the e-voting protocol's outcome; *e.g.* $\mathrm{Tally} = !\mathtt{in}(c_b, x).\mathsf{let}\ z = \Psi_b[x]\ \mathtt{in}\ \mathtt{out}(c, \mathrm{Extract}(x))$. This is the most well-established definition of ballot secrecy in symbolic model introduced in [25].

However, many e-voting protocols in our class would not satisfy such a property because the attacker may force[1] a particular voter (*e.g.* A) to not cast any ballot in order to infer, from the tally's outcome, the vote that the other voter (*e.g.* B) has cast. This is well-known and usually addressed by modelling phases as *synchronisation barriers* as already acknowledged in [25]: "when we omit the synchronization [...] privacy is violated." With such synchronisation barriers, all participants shall reach the same barrier in order to move to the next phase preventing the previous scenario from happening. However, the use of barriers (as done *e.g.* in [16], [18], [24]–[26]) also limits the range of e-voting protocols one can model and analyse. For instance, no synchronisation barrier can be put under a replication, which forbids modelling authorities that act during several phases or threat models with no dishonest voter.

In contrast, we choose to model e-voting phases as *weak phases* to avoid those limitations and thus need an extra assumption as a counterpart to synchronisation barriers. We shall restrict our analysis to *fair executions*[2] where, at each beginning of phase, the voter A and B are still present and A casts a ballot, if and only if, B does so. Note that all executions of protocols modelled with synchronisation barriers are necessarily fair. We are thus conservative over prior definitions. Our fairness assumption can also be seen as an extension of the tally's assumption in [17] that process the bulletin boards only if they contain both Alice and Bob's ballots.

Definition 6. Consider an e-voting protocol $(\mathcal{V}; \phi_0; V(\mathrm{id}, v); \mathcal{R}; (\Psi_b[], \mathrm{Extract}[]); \mathsf{th})$. An execution $(\mathcal{P}; \phi_0; 1) \xrightarrow{\mathsf{tr}} K$ for $\mathcal{P} \in \{\mathcal{S}, \mathcal{S}_r\}$ is said to be *fair for voter* $[\mathrm{id}, v]$ when at each beginning of phase i, there is a a process annotated $[\mathrm{id}, v]$ at phase i. Such an execution is said to be *fair* when, for some $v, v' \in \mathcal{V}$, (i) it is fair for $[A, v]$ and $[B, v']$ and (ii) $[A, v]$ casts a ballot if, and only if, so does $[B, v']$.

Finally, we give below the definition of *ballot secrecy*. We could have defined it as the trace equivalence (by symmetry) between $\mathcal{S} \cup \{\mathrm{Tally}\}$ and $\mathcal{S}_r \cup \{\mathrm{Tally}\}$ with a restriction over the explored traces (*i.e.* the ones that are fair) but we prefer our equivalent formulation in the interest of clarity. Note that the fairness assumptions get rid of strictly less behaviours than the use of synchronisation barriers, and are therefore more precise from that point of view.

Definition 7 (Ballot Secrecy)**.** An e-voting protocol $(\mathcal{V}; \phi_0; V(\mathrm{id}, v); \mathcal{R}; (\Psi_b[], \mathrm{Extract}[]); \mathsf{th})$ ensures *ballot se-*

[1]This attack is captured by the model but is unrealistic in practice. Indeed, in practical scenarios, to break the ballot secrecy of a particular voter, it would require the attacker to prevent all other voters from casting a vote or, in case of dishonest tally, from performing their individual verifiability checks (as observed in [4]).

[2]This should not be confused with the *fairness property* [3], [17] that is one of the security property often required from e-voting protocols.

crecy when for any fair execution $(\mathcal{S};\phi_0;1)\xrightarrow{\mathsf{tr}}K$, there exists a fair execution $(\mathcal{S}_r;\phi_0;1)\xrightarrow{\mathsf{tr}'}K_r$ such that:

- the attacker observes the same actions: $\mathsf{obs}(\mathsf{tr}) = \mathsf{obs}(\mathsf{tr}')$;
- the attacker observes the same data: $\phi(K) \sim \phi(K_r)$;
- the attacker observes the same tally outcome: $\mathrm{Res}(\mathsf{tr},\phi(K)) = \mathrm{Res}(\mathsf{tr}',\phi(K_r))$.

V. Conditions

We introduce three conditions and prove that together, they imply ballot secrecy. In Section V-A we provide intuition for our approach and formally define the support notions. We then define the conditions (*i.e.*, *Dishonest, Honest Relations, Tally Condition*) in sections V-B to V-D. We state in Section V-E that our conditions are sufficient.

A. Protocol phases and their links

a) ***Identity-leaking vs. Vote-leaking Phases****:* In a nutshell, ballot secrecy boils down to the absence of link between an identity and the vote this identity is willing to cast. However, as illustrated by the next example, the attacker is able to link different actions performed by the same voter as long as they take part in the same phase. Thus, each phase of the e-voting protocol must hide and protect either the identity of voters or the votes voters are willing to cast. It is thus natural to associate to each phase, a *leaking label*: either the phase (possibly) leaks identity (we call such phases *id-leaking*) or it (possibly) leaks vote (we call such phases *vote-leaking*). In order to ensure ballot secrecy, the *Honest Relations Condition*, which we define later, will enforce that the attacker cannot establish meaningful links (*i.e.* links that would hold for $\mathcal{S}$ but not for $\mathcal{S}_r$) between id-leaking phase outputs and vote-leaking phase outputs.

Example 8. Consider a voter's role process $V(\mathrm{id},v) = 1 : \mathtt{out}(a,\mathrm{id}).\mathtt{out}(a,v)$ (other components are unimportant here). This trivial protocol is an abstraction of a registration phase (voter sends its identity) followed by a voting phase (voter sends its vote). We show this does not ensure ballot secrecy (see also the full witness in [27]). Consider the (fair) execution starting with $(\mathcal{S};\emptyset;1)$ and producing the trace $\mathsf{tr} = \mathtt{out}(a,w_{\mathrm{id}}).\mathtt{out}(a,w_v).\mathtt{out}(a,w'_{\mathrm{id}}).\mathtt{out}(a,w'_v)$ whose the two first (resp. two last) actions are performed by the voter A (resp. B). This execution has no indistinguishable counterpart in $\mathcal{S}_r$. Indeed, because the first message *reveals the identity* of the voter, the attacker can test that the first output is performed by A. After the first output A, the $\mathcal{S}_r$ side can only output either B or v_1 but not v_0. However, because the second message *reveals the vote*, the attacker can make sure the output vote is v_0 and not v_1. Thus, this protocol does not ensure ballot secrecy because in a single phase (*i.e.*, phase 1), there is one output revealing the identity of the voter and one output revealing the voter's vote. However, the process $V(\mathrm{id},v) = 1 : \mathtt{out}(a,\mathrm{id}).\ 2 : \mathtt{out}(a,v)$ ensures ballot secrecy and does not suffer from the above problem. The attacker cannot force A to execute its first message leaking identity and then immediately its second message leaking its vote, because doing so would *kill* the process $V(B,v_1)$ (which is still in phase 1) preventing the whole execution from being fair. Thus, the attacker has to trigger all possible first-phase actions of A and B before moving to the second phase. After the first phase, we end up with the processes $\{\mathtt{out}(a,v_0),\mathtt{out}(a,v_1)\}$ on the $\mathcal{S}$ side and $\{\mathtt{out}(a,v_1),\mathtt{out}(a,v_0)\}$ on the $\mathcal{S}_r$ side, which are indistinguishable.

Thus, in this first iteration, we split outputs revealing identity and outputs revealing votes in distinct phases. This enables breaking links between identity and vote.

As we will later see, our approach requires that we associate a *leaking label* to each phase which is a binary label indicating whether we consider the phase to be *vote-leaking* or *id-leaking*. Our only goal is not find such a labelling for which our conditions hold, implying ballot secrecy. In practice and on a case-by-case basis, we can immediately associate the appropriate leaking label to a phase. However, we explain in Section VI-A how those labels can be automatically guessed.

Example 9 (Continuing Example 7)**.** We consider phase 1 (resp. phase 2) as id-leaking (resp. vote-leaking).

b) ***Id-leaking vs. Vote-leaking Names****:* As illustrated by the next example, a name presents in different outputs can also be exploited to link those outputs. This is problematic when phases of those outputs have different leaking labels since it would enable linking those phases and thus maybe an identity with a vote. That is why, similarly to phases, we associate a *leaking label* to each name created by role processes. Note that the phase in which the name is created is irrelevant. What really matters is where the name is used and to what kind of data it can be linked. Again this classification is easily done on a case-by-case basis in practice but we present simple heuristics to automatically infer it in Section VI-A.

Example 10. We continue Example 8 and consider $V(\mathrm{id},v) = 1 : \nu r.\mathtt{out}(a,\mathrm{id}\oplus r).2 : \mathtt{out}(a,v\oplus r)$ where $\oplus$ denotes the exclusive or operator. This new protocol seems similar to the previous iteration. However, it does not satisfy ballot secrecy. Now, the attacker can use the name r to link the action of the *id-leaking* phase with the action of the *vote-leaking* phase (see witness in Example 14), defeating the role of the phase which previously broke this link. Note that only names can lead to the this issue: all other kinds of data (*e.g.*, public constants) are uniform and do not depend on a specific voter session (*e.g.*, replace r by a constant $\mathsf{ok}\in\Sigma_c$ and the resulting protocol ensures ballot secrecy).

Example 11 (Continuing Example 9)**.** We consider the names k,k' (created by the voter as shown in Example 2) to be vote-leaking.

c) ***"Divide & Conquer"****:* One reason that ballot secrecy is hard to verify using existing techniques, is the fact that diff-equivalence is too rigid w.r.t. phases: it does not allow any flexibility at the beginning of phases. We should be able to stop there, and start again with a new pairing left-right, a

new biprocess[3]. A core ingredient of our technique is to split each role into independent, standalone sub-roles (each sub-role playing one phase of the initial role), which allows us to consider many more pairings including ones (left-right) that are not consistent over phases. One of our conditions will require that the attacker cannot distinguish the voter and other roles processes from standalone sub-role processes that do not need to know the execution of past phases. This is also important to ensure ballot secrecy, because otherwise the attacker might link two actions coming from two different phases and then learn that they came from the same voter.

We now formally define the sub-roles. Let $\mathbf{n}_\mathbf{i}^\mathbf{v}$ be the vector made of the constant v_i and all vote-leaking names (with indices i). Let $\mathbf{n}_\mathbf{i}^\mathbf{id}$ be the vector made of the constant id_i and all id-leaking names (with indices i). The pair $(\mathbf{n}_\mathbf{i}^\mathbf{v}, \mathbf{n}_\mathbf{j}^\mathbf{id})$ (deterministically) describes the initial data needed to start one full honest interaction of one voter with all necessary role sessions.

Definition 8. Recall that the voter process is of the form $V(\mathrm{id}, v) = k : \nu\mathbf{n}.V'(\mathrm{id}, v)$ where V' is without creation of names. We define $V(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ as the process $k : V'(\mathrm{id}_i, v_j)\sigma$ where σ maps names in $\mathbf{n}$ to corresponding names in $\mathbf{n}_\mathbf{i}^\mathbf{v} \cup \mathbf{n}_\mathbf{j}^\mathbf{id}$. We similarly define $A(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ for $A \in \mathcal{R}_o$. Finally, we define $\mathcal{R}_o(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) = \{A(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) \mid A \in \mathcal{R}_o\}^\#$.

Example 12 (Resuming Example 10)**.** Assuming r is said to be vote-leaking, one has $\mathbf{n}_\mathbf{i}^\mathbf{id} = \mathrm{id}_i$ and $\mathbf{n}_\mathbf{j}^\mathbf{v} = v_j, r_j$, and, $V(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) = 1 : \mathtt{out}(a, \mathrm{id}_i \oplus r_j).2 : \mathtt{out}(a, v_j \oplus r_j)$

Intuitively, the process $V(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ corresponds to the voter role process of identity id_i and vote v_i that will use all given names instead of creating fresh ones. Similarly for authorities. Note that in the vectors $\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}$, there may be names that are never used in some roles; we still give the full vectors as arguments though. We remark that given names $\mathbf{n}_\mathbf{i}^\mathbf{v}, \mathbf{n}_\mathbf{j}^\mathbf{id}$, there is a unique (modulo $=_\mathsf{E}$) execution of $(\{V(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})\} \uplus \mathcal{R}_o(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}); \phi_0; 1)$ following the idealised trace that is (up to some τ-actions) the bijective renaming of the honest execution (see Definition 3) from names used in the honest execution to names in $\mathbf{n}_\mathbf{i}^\mathbf{v}, \mathbf{n}_\mathbf{j}^\mathbf{id}$. We call that execution the *idealised execution for* $\mathbf{n}_\mathbf{i}^\mathbf{v}, \mathbf{n}_\mathbf{j}^\mathbf{id}$.

Definition 9 (Phase Roles)**.** Given $\mathbf{n}_\mathbf{i}^\mathbf{v}, \mathbf{n}_\mathbf{j}^\mathbf{id}$, consider the unique idealised execution for $\mathbf{n}_\mathbf{i}^\mathbf{v}, \mathbf{n}_\mathbf{j}^\mathbf{id}$: $(\{V(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})\} \uplus \mathcal{R}_o(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}); \phi_0; 1) \xrightarrow{\mathsf{tr}^h} (\emptyset; \phi; n)$. For some $A(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) \in \mathcal{R}(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ and some phase number $k \in [1; n]$, we note $A_f^k(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ the first process resulting from $A(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ of the form $k : P$ for some P if it exists; and 0 otherwise. Finally, the *phase role* of A for k, noted $A^k(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$, is the process one obtains from $A_f^k(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ by replacing by 0 each sub-process of the form $l : P'$ for some P' and $l > k$. Further, the process $A^\forall(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ is the sequential composition of all $A^i(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$. Finally, $V^k(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ and $V^\forall(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v})$ are defined similarly.

[3]We would like to achieve this even for roles that can perform an unbounded number of sessions.

In a nutshell, phase roles describe how roles behave in each phase, assuming that previous phases followed the idealised executions for the given names. A crucial property we eventually deduce from our conditions is that it is sufficient (*i.e.*, we do not lose behaviours and hence neither attacks) to analyse the phase roles in parallel instead of the whole e-voting system. Note that, by doing so, we do not only put processes in parallel that were in sequence, we also make them forget the execution of past phases cutting out some potential links that rely on that aspect. Indeed, the voter process in a phase i may use data received in a phase $j < i$ creating links between those two (*e.g.* via malicious tainted data). When put in parallel, all parts are standalone processes that are no longer linked by past execution. Note also that we put standalone processes in parallel that behave as if previous phases followed one specific instantiation of the honest trace (*i.e.*, the idealised trace) thus reducing a lot possible behaviours. This will be crucial for defining *Honest Relations Condition* via biprocesses that could not be defined otherwise.

Definition 10. The *id-leaking sub-roles* (respectively *vote-leaking sub-roles*) are as follow:

$\mathcal{R}^{\mathrm{id}}(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) = \{A^k(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) \mid A \in \mathcal{R}_o \cup \{V\}, k \text{ id-leak.}\}$

$\mathcal{R}^{v}(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) = \{A^k(\mathbf{n}_\mathbf{i}^\mathbf{id}, \mathbf{n}_\mathbf{j}^\mathbf{v}) \mid A \in \mathcal{R}_o \cup \{V\}, k \text{ vote-leak.}\}$

Example 13 (Continuing Example 11)**.** The *phase roles* are:

$$\begin{aligned}
V^1(\mathbf{n}_{\mathrm{id}}^\mathbf{id}, \mathbf{n}_\mathbf{i}^\mathbf{v}) &= 1 : M = \mathsf{commit}(v_i, k_i),\\
&\quad e = \mathsf{blind}(M, k_i'),\ s = \mathsf{sign}(e, \mathsf{key}(\mathrm{id})),\\
&\quad \mathtt{out}(c, \langle \mathsf{pk}(\mathsf{key}(\mathrm{id})); s\rangle).\ \mathtt{in}(c, x);\\
&\quad \mathsf{if}\ \mathsf{verSign}(x, \mathsf{pk}(k_R)) = e\ \mathsf{then}\ 0\\
V^2(\mathbf{n}_{\mathrm{id}}^\mathbf{id}, \mathbf{n}_\mathbf{i}^\mathbf{v}) &= 2 : M = \mathsf{commit}(v_i, k_i),\\
&\quad \mathtt{out}(c, \mathsf{sign}(M, k_R)).\ \mathtt{in}(c, y).\\
&\quad \mathsf{if}\ y = \langle y_1; M\rangle\ \mathsf{then}\ \mathtt{out}(c, \langle y_1, \langle M, k_i\rangle\rangle
\end{aligned}$$

The sub-roles are $\mathcal{R}^{\mathrm{id}} = \{V^1\}$, $\mathcal{R}^v = \{V^2, A_b\}$.

d) ***Honest Interactions****:* We will show that under our conditions, $\mathcal{S}$ is indistinguishable from an e-voting system based on the reunion of id-leaking and vote-leaking sub-roles. To achieve this property we eventually require that when a voter reaches a phase k then it must be the case that it had an honest interaction so far. This notion of honest interaction is captured by the honest trace th as formally defined next.

For two traces $\mathsf{tr}_1, \mathsf{tr}_2$ and a frame ϕ, we note $\mathsf{tr}_1 \equiv_\phi \mathsf{tr}_2$ if tr_1 and tr_2 are equal up to recipes and for all recipes M_1 of tr_1 we have that $M_1\phi{\downarrow} =_\mathsf{E} M_2\phi{\downarrow}$, M_2 being the corresponding recipe in tr_2. For some $1 < j \leq k_f$, we say that a trace tr_1 and a frame ϕ *follow a trace* tr_2 *up to phase* j (resp. *follow* tr_2) if $\mathsf{tr}_1 \equiv_\phi (\mathsf{tr}_2'\rho)$ where tr_2' is such that $\mathsf{tr}_2 = \mathsf{tr}_2'.\mathtt{phase}(j).\mathsf{tr}_2''$ for some tr_2'' (resp. $\mathsf{tr}_2' = \mathsf{tr}_2$) and ρ is some bijection of handles (so that the notion is insensitive to choices of handles). The above ensures that tr_1 and tr_2 compute messages having the same relations w.r.t. outputs (handles). For instance, if $\mathsf{tr} = \mathtt{out}(c, w).\mathtt{in}(c, w)$ is some trace, we would like to capture the fact that for a frame ϕ, any trace $\mathtt{out}(c, w).\mathtt{in}(c, M)$ follows tr as long as M computes the same message as w (*i.e.* $w\phi =_\mathsf{E} M\phi{\downarrow}$). Finally, given an execution, we say that a voter *had an honest interaction up to phase* j (resp. *had an honest*

interaction) when there exist role session annotations $s_1, \ldots, s_n$ such that the sub-trace made of all actions labelled by this voter or s_i with the resulting frame follow an instance of the honest trace up to phase j (resp. follow an instance of the honest trace).

Recall that the trace tr^h from the honest execution is an instance of the honest trace th and are equal when th has no unknown part (*i.e.* second-order variable). The purpose of the idealised trace in subsequent developments is to consider an arbitrary, fixed, instantiation that is uniform for all voters and sessions.

B. Honest Relations Condition

This condition aims at ensuring the absence of id-vote relations in honest executions. Let $\mathbf{n_A^{id}}$ (resp. $\mathbf{n_B^{id}}$) be the public identity id_A (resp. id_B) and as many as necessary (depending on the protocol) fresh names. We may use A (resp. B) to refer to id_A (resp. id_B). Let $\mathbf{n_0^v}$ (resp. $\mathbf{n_1^v}$) be the public vote v_0 (resp. v_1) and fresh names. We require these names to be pairwise distinct. We define the biprocess $\mathcal{B}$ at the core of the *Honest Relations Condition*:

$$\begin{aligned}\mathcal{B} = \quad & (\{\mathcal{R}^{\mathrm{id}}(\mathbf{n_A^{id}}, \mathsf{choice}[\mathbf{n_0^v}, \mathbf{n_1^v}]), \mathcal{R}^{v}(\mathsf{choice}[\mathbf{n_A^{id}}, \mathbf{n_B^{id}}], \mathbf{n_0^v}),\\ & \mathcal{R}^{\mathrm{id}}(\mathbf{n_B^{id}}, \mathsf{choice}[\mathbf{n_1^v}, \mathbf{n_0^v}]), \mathcal{R}^{v}(\mathsf{choice}[\mathbf{n_B^{id}}, \mathbf{n_A^{id}}], \mathbf{n_1^v})\}\\ & \uplus\ !\mathcal{R}; \phi_0; 1)\end{aligned}$$

The biprocess $\mathcal{B}$ represents a system where votes (and vote-leaking names) are swapped in id-leaking phase and identities (and id-leaking names) are swapped in vote-leaking phase. The attacker should not be able to observe any difference in the absence of relation between identity plus id-leaking names and vote plus vote-leaking names.

Note that the swaps are inconsistent across phases (*i.e.* we do not swap same things in all phases). We could not have defined such non-uniform swaps by relying on the roles' processes. Instead, this has been made possible thanks to our divide & conquer approach.

Example 14 (Resuming Example 12)**.** One has $\mathcal{B} = (\{1 : \mathtt{out}(a, \mathrm{id}_A \oplus \mathsf{choice}[r_0, r_1]), 2 : \mathtt{out}(a, v_0 \oplus r_0), A_b, 1 : \mathtt{out}(a, \mathrm{id}_B \oplus \mathsf{choice}[r_1, r_0]), 2 : \mathtt{out}(a, v_1 \oplus r_1), A_b, !A_b\}; \emptyset; 1)$. We argue that this biprocess is not diff-equivalent. Indeed, the attacker can xor id_A, v_0, an output of phase 1, and an output of phase 2. For one choice of the outputs, the attacker may obtain 0 on the left. This cannot happen on the right. The same interaction is also an attack trace for ballot secrecy.

One requirement of the *Honest Relations Condition* is the diff-equivalence of $\mathcal{B}$. However, this alone does not prevent the honest trace to make explicit links between outputs of id-leaking phases and inputs of vote-leaking phase (or the converse). This happens when the honest trace is not *phase-oblivious* as defined next.

Definition 11. The honest trace is *phase-oblivious* when:

- in all input $\mathtt{in}(c, M)$ of th in a phase i, handles in M must not come from phases with a different leaking labels (*i.e.*, vote or id-leaking) than that of phase i, and
- a variable $X \in \xi$ of th must not occur in two phases having different leaking labels.

Condition 1. (Honest Relations) The Honest Relations Condition is satisfied if $\mathcal{B}$ is diff-equivalent and the honest trace is phase-oblivious.

C. Tally Condition

The *Tally Condition* prevents ballot secrecy attacks that exploit the tally's outcome. Intuitively, the Condition requires that for any valid ballot produced by $\mathcal{S}$, either (i) the ballot stems from an honest execution of A or B, or (ii) it is a dishonest ballot and in that case, it must be that the vote the Tally would extract from that ballot is the same before or after the swap $A \leftrightarrow B$. Formally, we deal with the case (ii) by considering executions of $\mathcal{B}$ so that we can always compare ballots before or after the swap $A \leftrightarrow B$ (*i.e.* intuitively in $\mathcal{S}$ or in $\mathcal{S}_r$).

Condition 2. (Tally) We assume that $\mathcal{B}$ is diff-equivalent. The *Tally Condition* holds if for any execution $\mathcal{B} \xrightarrow{\mathsf{tr}} \mathcal{B}'$ leading to two frames ϕ_l, ϕ_r such that the corresponding execution on the left is fair, it holds that for any ballot $\mathrm{ba} \in \mathrm{BB}(\mathsf{tr}, \phi_l)$ (with w as the handle) then either:

1) there exists a voter $V(\mathrm{id}, v)$ which had an honest interaction and cast a valid ballot w (it stems from an honest voter);
2) or there exists some $v \in \mathcal{V} \cup \{\bot\}$ such that $\mathrm{Extract}(w\phi_l)\downarrow v$ and $\mathrm{Extract}(w\phi_r)\downarrow v$ (it may correspond to a dishonest ballot that should not depend on A's or B's vote).

The Tally Condition does not forbid making copies of a ballot completely "blindly" (*i.e.*, without being able to link this ballot to a specific voter/identity). Indeed, votes in vote-leaking phases are identical on both sides of $\mathcal{B}$ and the second case (2) will thus trivially hold. This actually improves the precision of the condition since such copies are not harmful w.r.t. ballot secrecy. In fact, the attacker may observe a bias that he might exploit to learn the vote contained in a specific ballot, but the attacker would be unable to link this ballot (and its vote) to a specific voter. Therefore, our condition captures a refined notion of *ballot independence attacks* [3].

D. Dishonest Condition

This condition prevents attacks based on actively dishonest interactions where the attacker deliberately deviates from the honest trace in order to exploit possibly more links (*e.g.* see tainted data example from Introduction). The idea of the condition is to be able to reduce the behaviours of the voter system to the parallel composition of all *phase roles* that are based on *the idealised execution* for some names chosen non-deterministically. The condition requires that if a voter process moves to the next phase in an execution of the e-voting system then it must be the case that it had an honest interaction up to that phase and all agents involved in that honest interaction are not involved in others. When $\mathsf{th} = \mathsf{tr}^h$ (no unknown part in the honest execution), this is sufficient to show that roles are indistinguishable from the parallel composition of phase roles. However, when $\mathsf{th} \neq \mathsf{tr}^h$, some attacker choices for second second-order variables of th may break the latter. For that case, the

condition thus requires an additional diff-equivalence between the system based on roles and the system based on the sequential composition of the phase roles (*i.e.* processes $A^{\forall}$). To make sure that the tally's outcome could not break this equivalence, we test the former in presence of an oracle opening *all* ballots.

Formally, we let $V^D(\mathbf{n}^{\mathbf{id}}_{\mathrm{id}}, \mathbf{n}^{\mathbf{v}}_{\mathbf{i}})$ be the biprocess obtained by the (straightforward) merge of the two following processes: (1) $V(\mathbf{n}^{\mathbf{id}}_{\mathrm{id}}, \mathbf{n}^{\mathbf{v}}_{\mathbf{i}})$ and (2) $V^{\forall}(\mathbf{n}^{\mathbf{id}}_{\mathrm{id}}, \mathbf{n}^{\mathbf{v}}_{\mathbf{i}})$ (*i.e.* see Definition 9). Recall that the process $V^{\forall}$ forgets the past execution at the beginning of each phase and behaves as if the past execution followed the idealised trace. In particular, it forgets previous (possibly malicious) input messages. We similarly define biprocesses A^D for $A \in \mathcal{R}_o$. Given an identity id and a vote v, we define a process:

$$S_f(\mathrm{id}, v) = \nu \mathbf{n}^{\mathbf{id}}_{\mathbf{0}}.\nu \mathbf{n}^{\mathbf{v}}_{\mathbf{0}}.(\Pi_{A \in \mathcal{R}_o \cup \{V\}} A^D((\mathrm{id}, \mathbf{n}^{\mathbf{id}}_{\mathbf{0}}), (v, \mathbf{n}^{\mathbf{v}}_{\mathbf{0}})))$$

where Π denotes a parallel composition and $\mathbf{n}^{\mathbf{id}}_{\mathbf{0}}$ (resp. $\mathbf{n}^{\mathbf{v}}_{\mathbf{0}}$) is made of all id-leaking names except the identity (resp. vote). Intuitively, S_f starts by creating all necessary names and is then ready to complete one voter session (according to processes V and $A \in \mathcal{R}$ on the left and $V^{\forall}, A^{\forall}$ on the right) using those names. Next, the oracle opening all valid ballots is as follows: $\mathrm{OpenBal} = k_f : \mathtt{in}(c_u, x).\mathsf{let}\, z = \Psi[x]\, \mathsf{in}\, \mathsf{let}\, v = \mathrm{Extract}[x]\, \mathtt{in}\, \mathtt{out}(c_u, v))$ where c_u is some public channel and k_f is the last phase that occurs in the honest trace th. Finally, we define: $\mathcal{B}^D = (\{S_f(A, v_0), S_f(B, v_1), !\mathrm{OpenBal}\} \cup !\,\mathcal{R}; \phi_0)$.

Example 15 (Continuing Example 13)**.** The process V^D associated to the FOO protocol is shown below:

$1 : M = \mathsf{commit}(v_i, k_i)$,
$e = \mathsf{blind}(M, k'_i)$, $s = \mathsf{sign}(e_i, \mathsf{key}(\mathrm{id}))$,
$\mathtt{out}(c, \langle \mathsf{pk}(\mathsf{key}(\mathrm{id})); s\rangle)$. $\mathtt{in}(c, x)$.
$\mathsf{if}\ \mathsf{verSign}(x, \mathsf{pk}(k_R)) = e$
$\mathsf{then}\ 2 : \mathtt{out}(a, \mathsf{choice}[\mathsf{unblind}(x, k'_i), \mathsf{sign}(M, k_R)])$.
$\mathtt{in}(c, y)$. $\mathsf{if}\ y = \langle y_1; M\rangle\ \mathsf{then}\ \mathtt{out}(c, \langle y_1, \langle M, k_i\rangle\rangle)$

Condition 3. (Dishonest) The *Dishonest Condition* holds when:

1) For any fair execution $(\mathcal{S}; \phi_0; 1) \xrightarrow{\mathtt{tr.phase}(j)} (\mathcal{P}; \phi; j)$, if a process at phase j annotated $[\mathrm{id}, v]$ for $\mathrm{id} \in \{A, B\}$ and $v \in \mathcal{V}$ is present in $\mathcal{P}$ then it had an honest interaction in tr, ϕ up to phase j. Moreover, authority sessions a_i involved in this honest interaction are not involved in other honest interactions.
2) If th has some unknown part (*i.e.* $\mathsf{th} \neq \mathsf{tr}^h$), then the biprocess $\mathcal{B}^D$ is diff-equivalent.

E. Main Theorem

Our main theorem states that our three conditions together imply ballot secrecy. It is based on the following Lemma that states the essential property we deduce from the Dishonest Condition. Note that the definition of having honest interactions is straightforwardly extended to executions performed by phase sub-roles. For instance, $V^1(\mathbf{n}^{\mathrm{id}}_{\mathrm{id}_{\mathbf{A}}}, \mathbf{n}^v_i)$ would be annotated $[\mathrm{id}_A, v_i]$. We give full proofs of the lemma and our Main Theorem in [27].

Let v_i, v_j be some distinct votes in $\mathcal{V}$ and tr a trace of the form $\mathsf{tr}_0.\mathtt{phase}(k).\mathsf{tr}_1$ for some $1 \leq k \leq k_f$ where no $\mathtt{phase}(\cdot)$ action occurs in tr_1. If the dishonest condition holds then there exists a fair execution

$$(\{V(\mathrm{id}_A, v_i), V(\mathrm{id}_B, v_j)\} \cup !\mathcal{R}; \phi_0; 1) \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; k),$$

if, and only if, there exist pairwise distinct names $\mathbf{n}^{\mathbf{id}}_{\mathbf{A}}, \mathbf{n}^{\mathbf{id}}_{\mathbf{B}}, \mathbf{n}^{\mathbf{v}}_{\mathbf{i}}, \mathbf{n}^{\mathbf{v}}_{\mathbf{j}}$ (not including vote or identity), a trace $\mathsf{tr}' = \mathsf{tr}'_0.\mathtt{phase}(k).\mathsf{tr}'_1$ and a fair execution

$$\begin{aligned}&(\{\mathcal{R}^{\mathrm{id}}((\mathrm{id}_A, \mathbf{n}^{\mathbf{id}}_{\mathbf{A}}), (v_i, \mathbf{n}^{\mathbf{v}}_{\mathbf{i}})), \mathcal{R}^v((\mathrm{id}_A, \mathbf{n}^{\mathbf{id}}_{\mathbf{A}}), (v_i, \mathbf{n}^{\mathbf{v}}_{\mathbf{i}})),\\ &\quad \mathcal{R}^{\mathrm{id}}((\mathrm{id}_B, \mathbf{n}^{\mathbf{id}}_{\mathbf{B}}), (v_j, \mathbf{n}^{\mathbf{v}}_{\mathbf{j}})), \mathcal{R}^v((\mathrm{id}_B, \mathbf{n}^{\mathbf{id}}_{\mathbf{B}}), (v_j, \mathbf{n}^{\mathbf{v}}_{\mathbf{j}}))\} \uplus !\mathcal{R}; \phi_0; 1)\\ &\xrightarrow{\mathsf{tr}'} (\mathcal{Q}; \psi; k).\end{aligned}$$

where $[\mathrm{id}_A, v_i]$ and $[\mathrm{id}_B, v_j]$ had an honest interaction in $\mathsf{tr}'_0.\mathtt{phase}(k)$ up to phase k. ψ. In both directions, we additionally have that $\mathsf{obs}(\mathsf{tr}') = \mathsf{obs}(\mathsf{tr})$, $\phi \sim \psi$ and $\mathrm{Res}(\mathsf{tr}, \phi) = \mathrm{Res}(\mathsf{tr}', \psi)$.

If an e-voting protocol ensures the Dishonest Condition, the Tally Condition, and, the Relation Condition then it ensures ballot secrecy.

VI. Mechanisation and Case Studies

We now apply our technique to several case studies, illustrating its scope and effectiveness. We show in Section VI-A how ProVerif can be used to automatically verify the three conditions. In Section VI-B, we present and benchmark several e-voting protocols within our class, and explore several threat models.

A. Verifying the conditions

We explain in this section how to leverage ProVerif to verify the three conditions via systematic encodings producing ProVerif models. At the end of this section we present an algorithm that shows that writing those encodings can be automated, but leave its implementation as future work. We show that the time spent by the algorithm computing those encodings is negligible compared to the time ProVerif spends to verify the produced models.

a) ***Guessing leaking labels****:* While it would be reasonable to require from users leaking labels for given e-voting protocols, very simple heuristics to guess them allow to conclude on all our case studies. First, the registration phase is often the first phase. Hence, guessing that the first phase is the only id-leaking phase always allows to conclude on our examples. Similarly, the following heuristic for guessing leaking labels of names proved to be precise enough: if the name is output then it takes the leaking label of the corresponding phase, if the name is used as signature key then it is id-leaking and otherwise it takes the leaking label of the phase of its first use.

b) ***Sound Verification of The Tally Condition****:* It is possible to verify the Tally Condition by analysing the diff-equivalence of the biprocess $\mathcal{B}$ in presence of an oracle opening all ballots (*i.e.* OpenBal defined in Section V-D). The diff-equivalence of $\mathcal{B}^T = \mathcal{B} \uplus \{!\mathrm{OpenBal}\}$ implies the diff-equivalence of $\mathcal{B}$ and for all executions and valid ballots, item 2 of the Tally Condition. We formally state and prove the former in [27].

*c) **The Dishonest Condition**:* We explain how to verify item (1) of the Dishonest Condition using *correspondence properties of events* that ProVerif can verify. We can equip the e-voting system $\mathcal{S}$ with events that are fired with each input and output, and that contain exchanged messages and session annotations. Then, the fact that a specific voter passes a phase or casts a valid ballot can be expressed by such events. Further, the fact that a specific voter had an honest interaction (up to a certain phase or not) can be expressed as implications between events. For instance, if $\mathtt{th} = \mathtt{out}(c, w).\mathtt{in}(c, \langle X; w\rangle)$ then one would write $\mathtt{EventIn}(a, \langle x; y_w\rangle) \Rightarrow \mathtt{EventOut}((\mathrm{id}, v), y_w)$ where id, v are voter annotations and a is a role session annotation, x and y_w are variables and open messages in events must pattern-match with the exchanged messages. Note that this technique has already been used in a different context in the tool UKano [32]. Next, the fact that such an honest interaction should be disjoint can be established by verifying that outputs from honest executions should be different modulo $=_{\mathsf{E}}$.

*d) **Algorithm for verifying all conditions**:* The input format of our algorithm is a valid ProVerif file containing at least: public constants modelling ϕ_0 and $\mathcal{V}$, function and reduction rules modelling Ψ_b and $\mathrm{Extract}$ and a biprocess for each role describing $V, \mathcal{R}_o$ and the idealised execution. Formally, the left part of a biprocess associated to a role A should model $A(\mathbf{n}^{\mathrm{id}}, \mathbf{n}^v)$ while the right part should model $A^{\forall}(\mathbf{n}^{\mathrm{id}}, \mathbf{n}^v)$ where input messages are replaced by messages received in the honest execution. Moreover, constants in such messages corresponding to second-order variables in the honest execution shall be given distinguished names. Therefore, the right part of those biprocesses both specify the idealised execution and the honest trace. Hence, the user is just required to specify an e-voting protocol according to Definition 4.

As explained, from such a file, the honest trace th can be retrieved (by syntactical equality between inputs and parts of outputs) and the fact that th is phase-oblivious can be checked via a linear-time syntactical check. Exploiting the right part of the given biprocesses, the algorithm can compute $A^{\forall}(\mathbf{n}^{\mathrm{id}}, \mathbf{n}^v)$ and thus $A^k(\mathbf{n}^{\mathrm{id}}, \mathbf{n}^v)$ for all $1 \leq k \leq k_f$. Using the aforementioned heuristic, the algorithm guesses leaking labels for names and phases and deduce $\mathcal{R}^v(\mathbf{n}^{\mathrm{id}}, \mathbf{n}^v)$ and $\mathcal{R}^{\mathrm{id}}(\mathbf{n}^{\mathrm{id}}, \mathbf{n}^v)$. The algorithm then deduces $\mathcal{B}$ and, using the two functions modelling Ψ_b and $\mathrm{Extract}$, it also deduces $\mathcal{B}^T$. When th does contain second-order variables, the algorithm computes $\mathcal{B}^D$ from the left part of the given biprocesses and roles $A^{\forall}(\mathbf{n}^{\mathrm{id}}, \mathbf{n}^v)$.

Finally, the algorithm produces two or three files: (a) a file containing $\mathcal{B}^T$, (b) a file containing correspondence properties using encoding described above for modelling the Dishonest Condition, item (1), and, (c) if $\mathsf{th} \neq \mathsf{tr}^h$, a file containing $\mathcal{B}^D$. Then, ProVerif is used to verify the diff-equivalence of $\mathcal{B}^T$, all the correspondence properties and, when necessary, diff-equivalence of $\mathcal{B}^D$. If all checks hold then the algorithm deduces that the given e-voting protocol ensures ballot secrecy.

All the described tasks the algorithm should perform are linear-time syntactical manipulations of the given input data. Therefore, the cost of computing the three ProVerif files is negligible compared to the time spent by ProVerif for verifying the files. In our benchmarks, we thus only measured the latter.

B. Case Studies

We now describe the different e-voting systems we verified with our approach and compare (when possible) with the current state of the art (see Figure 3). We first give in-depth descriptions of the JCJ and Lee case studies, and then list other case studies for which we only give high-level descriptions.

We benchmark the verification times of our approach vs. the only comparable prior approach, *i.e.*, the *swapping technique* [16]. The swapping technique uses a direct encoding of ballot secrecy in ProVerif with synchronisation barriers where processes can be swapped. Other approaches are not automated (require non systematic manual efforts), or do not deal with the protocols and threat models we consider (see discussions in Section VII). Notably, while Tamarin is expressive enough to describe our case studies [18], it does not yet allow to automatically prove them. We summarise our results in Figure 3 and provide all our ProVerif models at [33].

a) **JCJ Protocol [22]:** We analysed the JCJ protocol [22] used in the Civitas system [34]. It has been designed to achieve a strong notion of privacy (*i.e.* coercion-resistance) but we limit our analysis to ballot secrecy.

The JCJ protocol is depicted in Figure 4. In a first phase, the voter requests a credential by disclosing its identity to a registrar who replies on a secure channel with a fresh credential $cred$. In a second phase, the registrar sends to the tally the created credential randomised and encrypted with the tally's public key and signed with the registrar's signing key. This will be used by the tally to authenticate ballots from registered voters. Then, in a third phase, the voter casts his ballot who takes the form of a complex Zero Knowledge (ZK) proof whose the public part (first argument of ZK) includes (i) a randomised encryption of her vote and (ii) a randomised encryption of her credential and whose the private part (second argument) contains the knowledge of the underlying credential and vote. The tally can then verify the ZK proof and perform a Plaintext Equality Test (PET) between part (ii) of the ZK proof and some encrypted credential that has been signed by and received from the registrar (at this point, the ballot is verified and can be published on the bulletin board). Finally, the encrypted vote can be opened, possibly after mixing, to reveal the vote.

We adapted the modelling from [15] (including the modelling of the ZK proofs) to consider a strictly stronger threat model (for ballot secrecy). We assumed that the registrar and the tally are honest, but that their secret keys sk_R and sk_T are compromised. We let the tally output verified ballots on the public channel c_b (thereby also taking the role of the ballot box). A voter requests a credential by revealing its identity in the clear but receives its credential on a secure channel. Voters send ballots on an anonymous channel. Naturally, the first phase and $cred$ are id-leaking, while the two last phases and r_V^1, r_V^2, r_R can be considered vote-leaking (following the heuristic from Section VI-A). We were able to establish our three conditions automatically with ProVerif and therefore establish ballot secrecy.

Protocol	Ballot Secrecy	Analysis time in seconds	
		Swapping	Our approach
FOO	verified	0.26	0.04
Lee 1	verified	46.00	0.04
Lee 2	verified	†	0.05
Lee 3	verified	†	0.01
Lee 4	attack	169.94	6.64
JCJ	verified	$\star$	18.79
Belenios	verified	$\star$	0.02

Fig. 3. Analysed protocols and results. Tests were performed using ProVerif 1.94 on a single 2.67GHz Xeon core with 48GB of RAM. † indicates non-termination within 45 hours or consumption of more than 30GB of RAM. We use $\star$ to indicate the approach yielded spurious attacks, which implies that the analysis is inconclusive. All our ProVerif models are available from [33].

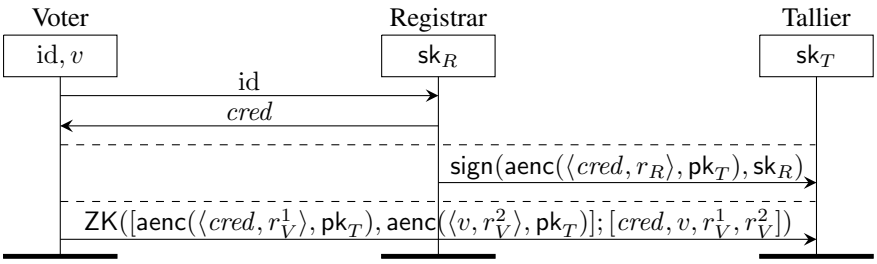

Fig. 4. Informal presentation of JCJ (phases are separated by dashed lines)

In comparison, the direct encoding of ballot secrecy with the swapping technique [16] fails to establish ballot secrecy. We have identified two main, independent reasons. First, when considering an unbounded number of honest voters, one also needs to consider an unbounded number of sessions of the registrar (this holds for the tallier as well). This is incompatible with equipping the registrar with synchronisation barriers yielding spurious attacks in practice (registrar sessions for A and B should be swapped after the first phase). While it may be possible to manually apply the barriers elimination theorem [16], an independent problem inherent to the swapping technique still prevents us to conclude. Indeed, even when considering the simplest scenario with only two honest voters and no dishonest voter, the swapping technique on our model yields a spurious attack. This is caused by a systematic limitation of the latter technique when ballot secrecy relies on the freshness of data produced in previous phases (here $cred$). We explain the underlying reason in Section VII-0a. Those two limitations are still problematic when the two last phases are collapsed and removing all phases is not successful either.

We note that unlike the automatic analysis of JCJ from [17], we took the registration phase into account. Importantly, [17] would not be able to do the same since their framework allows only one phase before the tally (i.e., the voting phase). Therefore, this is a real limitation and not a simple divergence of modelling choices. Note also that [15] analyses JCJ for coercion-resistance. However, they considered a simpler threat model in which the registration phase is completely hidden from the attacker. Moreover, their approach required manually and cleverly designed protocol-specific encodings since one has to "explicitly encode in the biprocess the proof strategy" according to [15].

b) **Lee *et al.* Protocol [24]:** We now support the claim that our class of e-voting protocols is expressive enough to capture a large class of threat models by analysis several threat models for Lee *et al.* (variant proposed in [24]). This protocol contains two phases. In the registration phase, each voter encrypts her vote with the tally's public key, signs the ciphertext and (output i) sends both messages to the registrar. The registrar verifies the signature, re-encrypts the ciphertext using a fresh nonce and (output ii) sends to the voter this signed ciphertext along with a Designated Verifier Proof (DVP) of re-encryption. The voter can then verify the signature and the DVP. Finally, in the voting phase, the voter (output iii) sends its ballot, which is the previously received signed re-encryption. We reused and adapted ProVerif models from [16].

Lee 1. The first threat model we consider is the only one analysed in [16]. It considers the registrar's signature key and the tally's private key corrupted, and considers infinitely dishonest voters. The channel of outputs (i) and (ii) is assumed to be untappable (*i.e.* everything is completely invisible to the attacker) while the channel of output (iii) is anonymous. Since the registrar's signing key is corrupted, the dishonest voters do not need to have access to registrar sessions (they can be played by the environment). Similarly, there is no need to explicitly model the tally. This considerably simplifies the models one needs to consider, partly explaining the effectiveness of the swapping technique [16] (46s).

Lee 2. In this scenario, we no longer consider the tally's key corrupted. When verifying ballot secrecy without our conditions, it is thus mandatory to explicitly model the tally. This change to the model causes ProVerif to not terminate on the direct encod-

ing of ballot secrecy with the swapping technique. We thus tried to approximate the model. We collapsed the two phases into one, which enables ProVerif to terminate in 45.33 seconds on the direct-encoding. Unfortunately, this approximation does not always solve the problem: if the security relies on the phases, one would obtain spurious attacks. For instance, removing all phases causes ProVerif to return a spurious attack. More importantly, this approximation is not sound in general (we may miss some attacks). In contrast, the verification of our conditions only takes a fraction of a second without the above approximation.
Lee 3. We additionally consider a secure registrar signing key. We now need to explicitly model a registrar for dishonest voters. As for the previous model, ProVerif is unable to directly conclude. After collapsing phases, it terminates in 269.06 seconds. In contrast, our approach concludes in under 0.1 second.
Lee 4. We modify the previous threat model and weaken the output channel's security (i) to be insecure instead of untappable. In this case, ballot secrecy no longer holds. ProVerif returns an attack on the tally condition (verified using the ballot-opening oracle; see Section VI-A). Relying on the latter, we can immediately infer the attack on ballot secrecy.

c) **Other Case Studies:** We verified the three conditions for FOO [30] as described in our running example (with a dishonest registrar and considering dishonest voters). We use the same modelling and threat model as in [16].

We analysed the Belenios protocol [23] (in its mixnet version), which builds on the Helios protocol [35], and considered the same threat model as for JCJ. Again, contrary to [17], we took the registration phase into account. Note that the swapping technique failed to conclude because of spurious attacks for similar reasons as for JCJ.

We finally discuss the protocol due to Okamoto [36] as modelled in [24]. This protocol features trap-door commitments that ProVerif is currently unable to deal with. However, this protocol lies in our class and our theorem thus applies. This could both ease manual verification and future automated verification (*e.g.* recent analysis [18]).

VII. Related Work

As mentioned before, diff-equivalence is known to be too imprecise to analyse vote-privacy via a direct encoding (acknowledged *e.g.* in [15], [16], [24], [26]).

a) ***Swapping technique:*** The swapping technique originates from [26], and 8 years later, was formally proven and implemented in ProVerif [16]. It aims to improve the precision of diff-equivalence for protocols with *synchronisation barriers*. The main idea is to guess some process permutations at the beginning of each barrier and then verify a modified protocol based on these permutations. We develop an example showing this mechanism in [27]. Theoretically, the permutations do not break trace equivalence since they transform configurations into structurally equivalent configurations. This approach is only compatible with replication in a very constrained way: all barriers above and below a replication must be removed, which reduces precision. Given a model with barriers, the front-end first generates several biprocesses without barriers, each corresponding to a possible swap strategy (*i.e.* the permutation done at each barrier); note that the number of such strategies grows exponentially with the size of the system (number of phases or number of roles). The equivalence holds if one of the produced biprocesses is diff-equivalent (proven in [16]). Similar techniques [18] have been used in the tool Tamarin relying on multisets. Essentially, all agents are put in a multiset at synchronisation barriers and a rule allows to shuffle this multiset before moving to the next phase. Therefore, the same limitations w.r.t. replications hold. Moreover, Tamarin will also have to explore all possible swaps. The fact that no replication can be put under a barrier notably forbids to model authorities crossing phases (because one needs to consider unbounded number of sessions of them) as well as threat models where no dishonest voter is considered for the same reason. The swapping approach also suffers from systematic precision issues when the security relies on the freshness of data created in previous phases. Indeed, the compiler introduces many new internal communications in the produced biprocesses. However, the very abstract treatment of internal communication used by ProVerif causes the tool to also explore the possibility of swapping data with an old session whose data has already been swapped before. We consider this to be a significant limitation, which manifests itself as spurious attacks *e.g.* the ones for JCJ and Belenios (see Figure 3); the credential being the fresh data coming from the registration phase and used during the voting phase. We provide more details in [27].

b) ***Small-attack property:*** A different line of work is to devise small attack properties, as for example in [17] for ballot secrecy. They show that proving ballot secrecy for some specific finite-scenarios implies ballot secrecy for the general, unbounded case. The focus in [17] is on complex ballot weeding mechanisms, as used for example in Helios [35]. In contrast to our work, they require that the pre-tally part contains only one voting phase that must be action-determinate (same actions yield statically equivalent frames). This approach is therefore unable to deal with e-voting protocol models that involve more than one phase, like the ones we consider in Section VI. Moreover, considering only one phase greatly simplifies the verification since it hides the diff-equivalence problems mentioned previously. Moreover, the finite-scenarios still lead to state space explosion problems. Because of this, they were unable to automatically verify the JCJ protocol, even without modelling the registration phase.

c) ***Privacy via type-checking:*** A sound type system for proving trace equivalence has been proposed [19], which seems a promising approach. This work reuses diff-equivalence as an approximation of trace equivalence and thus suffers from its limitations. Moreover, it is limited to standard primitives ((a)symmetric encryption, signatures, pairing, and hashing), which means that it currently cannot deal with our case studies, since the protocols in our case studies use primitives that are not supported by this method yet.

The analysis method we develop in this paper borrows the methodological approach of [32]: devise sufficient conditions implying a complex privacy property hard to verify, via a

careful analysis and categorisation of known attacks. However, we target a different class of protocols and property and devise different conditions.

VIII. Conclusion

We presented three conditions that together imply ballot secrecy. They proved to be tight enough to be conclusive on several case studies. Verifying ballot secrecy in a modular way via our conditions constitutes a new approach which outperforms prior works: we cover a greater class of e-voting protocols and threat models, and the analysis is more efficient.

Our new approach has also opened several avenues for future work. First, our notion of tally is currently limited. Hence, our method is currently unable to deal with revotes. While adding revotes might be directly achievable, we conjecture that considering revoting policies (*e.g.* ballot weeding in Helios as analysed in [17]) is a more intricate challenge. Furthermore, our notion of tally cannot deal with homomorphic tallying and only produces a set of votes, while e-voting protocols satisfying *verifiability* should also produce verification data (*e.g.* ZK proofs of correct decryption). We would like to extend our class of e-voting protocols accordingly. Second, our notion of fairness and our Dishonest Condition currently lack precision in the presence of certain mixnet roles. For instance, a degenerated mixnet such as $M = 1 : \mathtt{in}(c, x).2 : \mathtt{out}(c, \mathtt{dec}(x, k))$ (where c is public) currently introduces spurious attacks that are not prevented by our fairness condition and that are detected by our Dishonest Condition (note that the problem disappears when c is private). Third, we believe that our conditions can be adapted to enforce more complex privacy-type properties in e-voting protocols such as *receipt-freeness* and *coercion-resistance* [15], [24]. Fourth, we want to implement our algorithm for verifying all conditions as a ProVerif front-end.

We think that the modular *privacy via sufficient conditions* methodology we presented advances the state-of-the-art for the automated analysis of privacy-related properties, and paves the way for further developments.

References

[1] Bernhard, D., Cortier, V., Galindo, D., Pereira, O., Warinschi, B.: SoK: A comprehensive analysis of game-based ballot privacy definitions. In: Proc. 35th Symposium on Security and Privacy, (S&P'15), IEEE (2015)

[2] Cortier, V., Galindo, D., Küsters, R., Müller, J., Truderung, T.: Sok: Verifiability notions for e-voting protocols. In: Proc. 36th Symposium on Security and Privacy, (S&P'16), IEEE (2016)

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

[4] Cortier, V., Lallemand, J.: Voting: You can't have privacy without individual verifiability. In: Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM (2018) 53–66

[5] Kremer, S., Rønne, P.: To du or not to du: A security analysis of du-vote. In: Proc. 1st European Symposium on Security and Privacy (EuroS&P'16), IEEE (March 2016) 303–323

[6] Cortier, V., Drăgan, C.C., Dupressoir, F., Schmidt, B., Strub, P.Y., Warinschi, B.: Machine-checked proofs of privacy for electronic voting protocols. In: Proc. Symposium on Security and Privacy, IEEE (2017) 993–1008

[7] Meier, S., Schmidt, B., Cremers, C., Basin, D.: The Tamarin Prover for the Symbolic Analysis of Security Protocols. In: Proc. 25th International Conference on Computer Aided Verification (CAV'13), Springer (2013) 696–701

[8] Armando, A., et al.: The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In: Proc. 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12), Springer 267–282

[9] Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: Proc. 14th Computer Security Foundations Workshop (CSFW'01), IEEE Comp. Soc. Press (2001) 82–96

[10] Cremers, C.: The Scyther Tool: Verification, falsification, and analysis of security protocols. In: Proceedings of CAV'08. LNCS, Springer (2008)

[11] Basin, D., Cremers, C.: Know your enemy: Compromising adversaries in protocol analysis. ACM Transactions on Information and System Security (TISSEC) **17**(2) (2014) 7

[12] Basin, D., Dreier, J., Hirschi, L., Radomirovic, S., Sasse, R., Stettler, V.: A formal analysis of 5g authentication. In: Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM (2018) 1383–1396

[13] Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the tls 1.3 standard candidate. In: Security and Privacy (SP), 2017 IEEE Symposium on, IEEE (2017) 483–502

[14] Delaune, S., Hirschi, L.: A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. Journal of Logical and Algebraic Methods in Programming (2016)

[15] Backes, M., Hritcu, C., Maffei, M.: Automated verification of remote electronic voting protocols in the applied pi-calculus. In: Proc. 21st IEEE Computer Security Foundations Symposium, (CSF'08), IEEE Computer Society Press (2008) 195–209

[16] Blanchet, B., Smyth, B.: Automated reasoning for equivalences in the applied pi calculus with barriers. In: Proc. 29th Computer Security Foundations Symposium, (CSF'16), IEEE (2016)

[17] Myrto Arapinis, V.C., Kremer, S.: Three voters are enough for privacy properties. In: Proc. European Symposium on Research in Computer Security, (ESORICS'16), Springer (2016)

[18] Dreier, J., Duménil, C., Kremer, S., Sasse, R.: Beyond subterm-convergent equational theories in automated verification of stateful protocols. In: Proc. 6th International Conference on Principles of Security and Trust, (POST'17). (2017)

[19] Cortier, V., Grimm, N., Lallemand, J., Maffei, M.: A type system for privacy properties. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM (2017) 409–423

[20] Cortier, V.: Electronic voting: how logic can help. In: Proc. International Joint Conference on Automated Reasoning, (IJCAR'14), Springer (2014) 16–25

[21] Basin, D., Radomirovic, S., Schmid, L.: Alethea: A provably secure random sample voting protocol. In: 2018 IEEE 31st Computer Security Foundations Symposium (CSF), IEEE (2018) 283–297

[22] Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Proc. ACM workshop on Privacy in the electronic society, ACM (2005) 61–70

[23] Cortier, V., Galindo, D., Glondu, S., Izabachene, M.: Election verifiability for helios under weaker trust assumptions. In: Proc. European Symposium on Research in Computer Security, (ESORICS'14), Springer (2014) 327–344

[24] Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security (2009) 435–487

[25] Kremer, S., Ryan, M.D.: Analysis of an electronic voting protocol in the applied pi-calculus. In: Proc. 14th European Symposium on Programming (ESOP'05), Springer (2005) 186–200

[26] Delaune, S., Ryan, M.D., Smyth, B.: Automatic verification of privacy properties in the applied pi-calculus. In: Proceedings 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security, (IFIPTM'08), Springer (2008)

[27] Cremers, C., Hirschi, L.: Improving automated symbolic analysis for e-voting protocols: A method based on sufficient conditions for ballot secrecy. (2018) Extended version can be found at https://arxiv.org/abs/1709.00194.

[28] Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming (2008)

[29] Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proc. 28th Symposium on Principles of Programming Languages, (POPL'01), ACM Press (2001)

[30] Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Proc. International Workshop on the Theory and Application of Cryptographic Techniques, Springer (1992) 244–251
[31] Blanchet, B., Smyth, B., Cheval, V., Sylvestre, M.: ProVerif 1.96: automatic cryptographic protocol verifier, users manual and tutorial. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf (Accessed: 2018-11-12) (2016)
[32] Hirschi, L., Baelde, D., Delaune, S.: A method for verifying privacy-type properties: the unbounded case. In: Proc. 37th IEEE Symposium on Security and Privacy, (S&P'16), IEEE (2016)
[33] Cremers, C., Hirschi, L.: Proverif models for our case studies. https://drive.google.com/open?id=13NYnNTlVff5zQnssWSNzubR8o1dAcHHi (2018) Accessed: 2018-11-12.
[34] Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: Proc. Symposium on Security and Privacy, (S&P'08), IEEE (2008) 354–368
[35] Adida, B.: Helios: Web-based open-audit voting. In: USENIX Security Symposium. Volume 17. (2008) 335–348
[36] Okamoto, T.: An electronic voting scheme. In: Advanced IT Tools. Springer (1996) 21–30