Election Verifiability Revisited: Automated Security Proofs and Attacks on Helios and Belenios

Sevdenur Baloglu*, Sergiu Bursuc*, Sjouke Mauw†, Jun Pang†
*SnT, University of Luxembourg
†DCS, University of Luxembourg
Email: {sevdenur.baloglu, sergiu.bursuc, sjouke.mauw, jun.pang}@uni.lu

***Abstract*—Election verifiability aims to ensure that the outcome produced by electronic voting systems correctly reflects the intentions of eligible voters, even in the presence of an adversary that may corrupt various parts of the voting infrastructure. Protecting such systems from manipulation is challenging because of their distributed nature involving voters, election authorities, voting servers and voting platforms. An adversary corrupting any of these can make changes that, individually, would go unnoticed, yet in the end will affect the outcome of the election. It is, therefore, important to rigorously evaluate whether the measures prescribed by election verifiability achieve their goals. We propose a formal framework that allows such an evaluation in a systematic and automated way. We demonstrate its application to the verification of various scenarios in Helios and Belenios, two prominent internet voting systems, for which we capture features and corruption models previously outside the scope of formal verification. Relying on the Tamarin protocol prover for automation, we derive new security proofs and attacks on deployed versions of these protocols, illustrating trade-offs between usability and security.**

***Index Terms*—electronic voting, verifiability, verification.**

I. Introduction

Towards symbolic models. Election verifiability has emerged over the years as a crucial property for ensuring that the outcome of an electronic voting system correctly reflects the intention of eligible voters [1]–[5]. In parallel, we have seen a series of real-world attacks on election verifiability in deployed systems [6]–[10] as well as the development of formal models for its security foundations [5], [11]–[14]. An important factor contributing to the success of formal models for other classes of security protocols has been their ability to provide tool support for finding attacks or security proofs. See [15] for a recent survey and [16], [17] for an illustration in the case of secure messaging protocols. Of particular interest is fully automated verification, which relies on the so-called symbolic model used in tools like ProVerif [18] and Tamarin [19]. A first feature of the symbolic model is the perfect cryptography assumption, whereby messages and processes are represented in abstract calculus [20]. A second feature is that security properties are typically defined by a conjunction of trace formulas, relating events in an execution trace. In defining a symbolic model for election verifiability, the main challenge is that each individual trace formula can only mention a fixed number of events in a trace, while taken together they should imply a global property of end-to-end verifiability, referring to an unbounded number of voters and ballots. Then we say the symbolic formulas are a set of sound verification conditions for verifiability. We call this property *soundness* in the following. Another property we expect from symbolic models is *generality*, i.e. the ability to cover a large class of protocols and realistic corruption scenarios.

Between theory and practice. Existing symbolic models and security proofs for election verifiability [13], [14], [21] make certain assumptions opening a gap with the way e-voting protocols are deployed, used and corrupted in practice. Consider the case of Helios [22]–[24] and Belenios [25], [26], two prominent internet voting systems that are also within the scope of previous work. These systems allow revoting and allow voters to verify their vote at any time during the voting phase, while existing symbolic models disallow revoting. The definition of election verifiability in [27] (using the more expressive computational model of EasyCrypt [28]) allows revoting, but voters can verify their votes only at the end of the voting phase. Another limitation of current formal models (that allow machine-assisted proofs) is their particular handling of corruption. Even the most general models, i.e. those of [27] and [21], do not consider the case when all election authorities are corrupted, which is, informally, conceived as the end goal of end-to-end verifiability. Moreover, considering Belenios, the two paradigmatic corruption scenarios (corrupted registrar or corrupted server) are subject to two different formal definitions in [27] and [21]. An immediate problem is one of generality, making it a challenge to apply their definitions to other systems and corruption models. Another problem is that, while the two corruption scenarios in Belenios are symmetric and expected to provide the same degree of security, we show that they do not. Indeed, concrete attacks are possible in one scenario and not in the other - incidentally when revoting is allowed and voters verify their ballots anytime.

Existing symbolic models do not allow dynamic corruption and do not provide any guarantee for corrupted voters. The computational model of [27] allows dynamic corruption, yet still does not provide any guarantee for corrupted voters. We argue that such a guarantee is desirable (like receipt-freeness with respect to privacy [29]) and should be within

the scope of the definition. The formal notion of corruption (i.e. leaked voter credentials) may, in practice, cover several different scenarios (e.g. credentials leaked by a storage device); appropriate procedures may allow effective verifiability even for such inadvertently corrupted voters, or for voters subject to dynamic corruption, e.g. after voting. Indeed, we show that this stronger property is achievable for example in Helios and Belenios.

Previous work. The first symbolic model for election verifiability was proposed by Kremer et al. [13], but the scenarios to which it applies assume that all voters are honest and verify their votes. Moreover, although the protocol models in [13] are symbolic - being specified in an abstract process algebra - the formulas used for specifying the security properties cannot be expressed in ProVerif/Tamarin. They are global properties referring to an unbounded number of events in a trace. The challenge for symbolic verifiability is, relying on universal quantification over events in a trace, to obtain a sound model for end-to-end verifiability within the standard class of trace formulas accepted by ProVerif/Tamarin.

A type-based symbolic model, also covering privacy properties and applied to Helios, is proposed by Cortier et al. in [14], which does not cover revoting and the associated notion of end-to-end verifiability is weaker than the one later proposed for Belenios in [21]: it only states that the multiset of verified votes for honest voters should be part of the final outcome. In general, we need a complete characterisation of the outcome, also limiting the multiset of adversarial votes, even for systems like Helios, which may be considered to provide weaker verifiability than Belenios. A first reason is that Helios can, in fact, achieve a verifiability property like Belenios, if one makes the assumption that the voting server and registrar are trusted (Belenios splits the trust between the two). A second reason is that, as we show, even when both of these parties are corrupted, Helios (and Belenios) can achieve a meaningful, complete notion of end-to-end verifiability. In order to prevent the clash attacks of [6], the model of [14] considers a property ensuring that ballots constructed by distinct voters are distinct. This focuses on a particular aspect of clash attacks, and we show that, when revoting is allowed, they are possible even under this restriction.

The symbolic model of verifiability that comes closest to our goals is the one by Cortier et al. [21], designed to verifiability for a variant of Belenios (a model for privacy is also analysed in [21]). It allows automated symbolic analysis for an unbounded number of parties, relying on sound verification conditions that can be checked by Tamarin. It has, however, several limitations in terms of generality: distinct corruption scenarios relating to election authorities are covered by distinct symbolic verifiability definitions, making it hard to use for new protocols and corruption scenarios; to obtain soundness, it disallows dynamic corruption of voters and it does not provide guarantees for corrupted voters; it does not take into account revoting, which, as we show, can be exploited by the adversary. The absence of clash attacks is not directly implied by symbolic verifiability in [21], but follows from additional (trust) assumptions, which vary according to the corruption scenario that is considered (honest server or honest registrar). More generally, these assumptions together with symbolic verifiability are shown to imply end-to-end verifiability. Intuitively, any verifiable electronic voting system should ensure the absence of clash attacks, and we show that making it explicitly part of symbolic verifiability requirements can entail end-to-end verifiability, without assumptions specific to various corruption scenarios.

Our contributions. Extending the definition from [21], we propose a general symbolic definition for election verifiability:
– it allows a unified, modular framework to test different versions of the same protocol with respect to various corruption abilities of the adversary; it has in its scope a general class of voting protocols and is, at the same time, suitable for automated protocol verification tools;
– it takes into account revoting;
– it captures classic clash attacks, as well as a more general class of clash attacks, where the clash is only on public credentials, without requiring a clash on ballots;
– it implies end-to-end verifiability in a generic framework, without requiring scenario-specific trust assumptions as in [21]. The implication relies on a minimal set of restrictions that are justifiable by public audits on a bulletin board, and it provides end-to-end verifiability guarantees even in presence of dynamic corruption, assuming voters have successfully verified their vote.

We apply our definition to the automated verification (with the Tamarin prover) of multiple scenarios in Helios and Belenios, considering various sets of corrupted components and individual verification procedures. For both protocols, we discover new attacks in some scenarios and derive new security proofs in others - the latter, notably, also for the scenarios when all election authorities are corrupted. The crucial factor delineating proofs from attacks, in the same corruption scenario, is the individual verification procedure, illustrating some trade-offs between usability and security.

A particular class of attacks that we (re)discover are clash attacks, where the verifiability mechanism of the system can be subverted so that ballots and credentials of two voters clash, facilitating ballot stuffing [6]. We show that against some versions of Helios, including the one currently deployed online, a clash attack can be performed by a significantly weaker adversary than the one of [6]. Indeed, it is sufficient to corrupt the authorities that assign public credentials and authenticate voters, and to rely on the way revote ballots are processed and verified by honest parties in the system, while the attack of [6] also assumes a corrupted voting platform.

In Belenios, when the voting server is honest, it performs some consistency checks between voter identities and public credentials, in order to ensure that a clash attack is not possible. However, our analysis shows that these checks are not as effective as expected. They can be subverted in order to mount clash attacks, ballot reordering attacks and ballot stuffing attacks. The main problem comes, again, from revoting

and also from the fact that the relation between voters and their public credentials is not known by the server beforehand.

Paper structure. Section II contains preliminaries for e-voting protocols and formal verification with Tamarin. Section III contains our formal framework and definition for election verifiability, which are applied to Helios and Belenios in Section IV.

II. Preliminaries

We present the general structure of typical electronic voting protocols, that we aim to formalise and capture within the scope of our definition. Then we introduce the symbolic framework of Tamarin, that constitutes the foundation of our formalisation.

A. Verifiable electronic voting protocols

Bulletin board. The main component of an electronic voting protocol is a so-called bulletin board, which we denote by BB. The BB records information regarding the execution of the protocol and can be used by voters to verify that their ballots are correctly recorded by the voting server. Any party can also audit it to verify that the election is executed as expected. The BB should satisfy two basic assumptions: (i) *public availability*, i.e. anyone can read its contents anytime; (ii) *immutability*, i.e. it can only be updated by adding content to it, and data cannot be removed or changed after it is published. We note that realising a perfect BB is still an active area of research [30]. We distinguish different parts of the BB by annotations suggestive of the role they play for electronic voting:

- BBkey - for the public key of the election;
- BBcand - for the set of eligible candidates;
- BBreg - for the set of registered eligible voters;
- $\mathsf{BBtally}$ - to represent the set of ballots to be tallied.

Setup and registration. A voting protocol assumes a set of eligible candidates $\mathsf{v}_1, \ldots, \mathsf{v}_k$ and a set of eligible voters represented by identities $\mathsf{id}_1, \ldots, \mathsf{id}_n$. When these identities should remain private, public credentials $\mathsf{cr}_1, \ldots, \mathsf{cr}_n$ are used to record ballots for the corresponding voters on BB. We note that the public credential can be equal to the voter identity, i.e. we can have $\mathsf{cr}_i = \mathsf{id}_i$. At registration time, voters obtain their (private) id, their public credential cr, and any additional private credentials which will be used by election authorities to authenticate voters. Moreover, the public key of the election is generated by authorities and posted on BB.

Voting procedure. A voter uses a voting platform (e.g. browser) to construct a ballot b, encoding the desired vote, and post it to a voting server. The server may authenticate voters and validate ballots before publishing them on BB.

Verification procedures. Voters can perform certain procedures in order to ensure that their vote is correctly recorded by the system. Typically, procedures instruct voters to check that tracking information they obtained after voting is present next to their credentials on BB.

Ballot tallying. Assume $\mathsf{BBtally}$ at the end of the voting phase contains $(\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n)$. Cryptographic techniques based on mixnets [31]–[33] or homomorphic encryption [34]–[36] allow to compute the outcome corresponding to $\mathsf{b}_1, \ldots, \mathsf{b}_n$ in a universally verifiable and privacy-preserving way.

Assumptions. In the presentation above, and in order to specify our symbolic definition of verifiability, we make the following assumptions about the class of e-voting protocols that we consider:

- each ballot on $\mathsf{BBtally}$ is associated to a public credential cr;
- the vote v recorded in a ballot b is uniquely determined given the public key of the election pk.

These assumptions are true for a significant class of protocols, including Helios [22]–[24] and Belenios [25], [26] that we present in detail. Other positive examples are Selene [37] and the Norwegian voting system [38]. There are, however, some negative examples. The first assumption is not true for JCJ/Civitas [39], [40] and for a version of Helios with detached aliases [6], since there is no way to publicly link tallied ballots to public credentials. The second assumption is not true when votes are encoded with perfectly hiding commitments, used e.g. in [41], [42] to obtain everlasting privacy. To formalise the second assumption, we use the notation $\mathsf{open}(\mathsf{b}, \mathsf{pk})$ to represent the vote determined by b and pk. Previous symbolic models for verifiability in e-voting also use similar assumptions and notation [14], [21]. In the next example, we consider a simple protocol to illustrate our class of protocols and definition. It can be seen as a basic version of Helios, for which we will describe a more detailed model in Section IV.

Example 1. *Assume two election authorities: one responsible for generating election keys and tallying ballots, called trustee, and one responsible for registration and ballot casting, called server. We assume there is a portion of the bulletin board, called* BBcast*, that records all ballots cast by voters:*
Setup. *The trustee generates a key pair* $(\mathsf{sk}, \mathsf{pk})$ *and publishes* pk *on* BBkey*. For each voter* id*, the server generates a public credential* cr *and a password* pwd*. It publishes* cr *on* BBreg*. Each eligible voter obtains* $\langle \mathsf{id}, \mathsf{cr}, \mathsf{pwd} \rangle$*.*
Voting. *The voter, relying on the voting platform, generates a ballot* $\mathsf{b} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r})$*, for some desired vote* v *and fresh randomness* r*. The voter authenticates with* pwd *and* b *is submitted to the server.*
Verification. *The ballots are posted on* BBcast *and voters can check it to ensure their ballot is present.*
Tally. *The last ballot corresponding to each credential on* BBcast *is recorded in* $\mathsf{BBtally}$*. If* $\mathsf{BBtally}$ *consists of* $(\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n)$ *and* pk *is the public key of the election, then the result is* $\mathsf{open}(\mathsf{b}_1, \mathsf{pk}), \ldots, \mathsf{open}(\mathsf{b}_n, \mathsf{pk})$*.*

In the previous example, as well as in Helios and Belenios, there are a few deployment options to consider for BBcast when revoting is allowed:

– *Show only the last ballot cast.* If a voter with public credential cr casts ballots $\mathsf{b}^1, \ldots, \mathsf{b}^m$, then BBcast will display $(\mathsf{cr}, \mathsf{b}^m)$, i.e. the last ballot cast by the voter. This is the version currently deployed online in Helios [24] and Belenios [25].
– *Show all ballots cast.* For a cr as above, BBcast will show $(\mathsf{cr}, \mathsf{b}^1), \ldots, (\mathsf{cr}, \mathsf{b}^m)$, i.e. all ballots that were cast by cr. In that case, it should be clear for the voter which ballot is considered to be the last one for the corresponding cr.
– *Verify anytime.* Voters can inspect BBcast anytime after voting to ensure it contains their ballot. This is the version currently deployed online in Helios [24] and Belenios [25].
– *Verify after voting ends.* Voters are instructed to verify the bulletin board only after the voting phase has ended. In this case voters verify their ballots directly on BBtally, after computed from BBcast. Abstaining voters may verify that there is no ballot present on BBtally associated to their credential.

B. Formal verification with Tamarin

We present the (multiset) rewriting framework as instantiated by Tamarin, referring to [43], [44] for more details. To represent cryptographic primitives and messages, we consider a set of function symbols $\mathcal{F}$ endowed with a set of equations $\mathcal{E}$. A message (also called a term) is built by applying functions symbols from $\mathcal{F}$ to variables from an infinite set $\mathcal{X}$, constants from $\mathcal{F}$, names from an infinite set $\mathcal{N}$ or, iteratively, to other messages built in this way. Certain names may be specified to be fresh and private in an execution trace, as explained below. Equalities between terms are implicitly interpreted as being modulo $\mathcal{E}$.

Example 2. *Let*

$$\mathcal{F} = \{\mathsf{pk}, \mathsf{enc}, \mathsf{dec}, \mathsf{h}, \mathsf{sign}, \mathsf{ver}, \mathsf{proof}_1, \mathsf{proof}_2, \mathsf{ver}_1, \mathsf{ver}_2\}$$

and $\mathcal{E}$ be the set of following equations:

$$\begin{aligned}
&(1) && \mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x},\\
&(2) \quad (\forall \mathsf{i}) && \mathsf{ver}_1(\mathsf{proof}_1(\mathsf{enc}(\mathsf{x}_\mathsf{i}, \mathsf{y}, \mathsf{z}), \mathsf{z}, \langle \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{k}\rangle),\\
& && \qquad \mathsf{enc}(\mathsf{x}_\mathsf{i}, \mathsf{y}, \mathsf{z}), \mathsf{y}, \langle \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{k}\rangle) = \mathsf{ok},\\
&(3) && \mathsf{ver}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{x}, \mathsf{pk}(\mathsf{y})) = \mathsf{ok},\\
&(4) && \mathsf{ver}_2(\mathsf{proof}_2(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{z}, \mathsf{w}), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{w}) = \mathsf{ok}.
\end{aligned}$$

A term $\mathsf{enc}(\mathsf{m}, \mathsf{pk}(\mathsf{k}), \mathsf{r})$ represents a ciphertext where m is the plaintext, k is the private key, whose public counterpart is $\mathsf{pk}(\mathsf{k})$, and r is the randomness used by the encryption algorithm. Equations (1) and (3) specify the standard properties of asymmetric encryption and digital signatures. Cryptography is assumed perfect: there is no other way to derive messages other than applying function symbols and equations. The proof_1 symbol models a zero-knowledge proof showing that the corresponding plaintext is within a certain range $\langle \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{k}\rangle$ and that the public encryption key is equal to y. The set of equations (2) models the corresponding verification procedure. The proof_2 symbol models a zero-knowledge proof that allows a party to link a constructed ciphertext to a given term w. In our model for Helios, we use the theory corresponding to equations (1)-(2), and for Belenios the full theory corresponding to (1)-(4).

The set of symbols is extended with fact symbols to represent adversarial knowledge, protocol state, freshness information, etc. A fact is represented by $\mathsf{F}(\mathsf{t}_1, \ldots, \mathsf{t}_\mathsf{k})$, where F is a fact symbol and $\mathsf{t}_1, \ldots, \mathsf{t}_\mathsf{k}$ are terms. There are the following special fact symbols: K - for attacker knowledge; Fr - for fresh data; In and Out - for protocol inputs and outputs. Other symbols may be added as required, e.g. for representing the protocol state. Facts can be persistent (used in the execution any number of times) or linear (used at most once). The notation $!\mathsf{F}$ is used in Tamarin to distinguish persistent facts, but all facts in our models can be assumed persistent, thus, we do not use this notation in the paper to avoid clutter.

A multiset rewriting rule is defined by $[L]\text{–}[\ M\ ]\!\!\rightarrow\![N]$, where L, M, N are multisets of facts called respectively premises, actions and conclusions. To ease protocol specification, we extend the syntax of multiset rules with variable assignments and equality constraints, i.e. we can write rules of the form $[L]\text{–}[\ E, M\ ]\!\!\rightarrow\![N]$ where L may contain expressions $\mathsf{x} = \mathsf{t}$ to define local variables and E is a set of equations of the form $u = v$. For two multisets of facts M_0, M_1 and rule $R = [L]\text{–}[\ E, M\ ]\!\!\rightarrow\![N]$ we say that M_1 can be obtained from M_0 by applying the rule R, instantiated with a substitution θ if: (1) every equality in $E\theta$ is true; (2) every fact in $L\theta$ is included in M_0; (3) M_1 is obtained from M_0 by removing linear facts included in $L\theta$ and adding all facts from $N\theta$. There are three special classes of rules in Tamarin: *network deduction rules* specify that the adversary obtains protocol outputs, provides protocol inputs, knows public data and does not know fresh data; *intruder deduction rules* allow the adversary to apply functions and exploit their equational properties (a function can be declared private if the adversary is not supposed to use it); *protocol rules* allow to specify the behaviour of honest parties.

Example 3. *Consider the set of rules Q_{keys}:*

(1) $[\ \mathsf{Fr}(\mathsf{k})\ ]\text{–}[\ \mathsf{Key}(\mathsf{k})\ ]\!\!\rightarrow\![\ \mathsf{Sk}(\mathsf{k}), \mathsf{Pk}(\mathsf{pk}(\mathsf{k})), \mathsf{Out}(\mathsf{pk}(\mathsf{k}))\ ]$
(2) $\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{r})\ \ \mathsf{in}$
$[\ \mathsf{Sk}(\mathsf{x}), \mathsf{Pk}(\mathsf{y}), \mathsf{Fr}(\mathsf{r})\ ]\text{–}[\ \mathsf{Enc}(\mathsf{c})\ ]\!\!\rightarrow\![\ \mathsf{Out}(\mathsf{c})\ ]$
(3) $[\ \mathsf{In}(\mathsf{y}), \mathsf{Sk}(\mathsf{x})\ ]\text{–}[\ \mathsf{Dec}(\mathsf{y})\ ]\!\!\rightarrow\![\ \mathsf{Out}(\mathsf{dec}(\mathsf{y}, \mathsf{x}))\ ]$

The first rule models the generation of a fresh secret key k, which is stored for later use in $\mathsf{Sk}(\mathsf{k})$, while its public counterpart is also stored and output to the network. The second rule outputs encryptions of stored secret keys. The third rule specifies that any received message may be decrypted with a stored secret key. The action facts $\mathsf{Key}(\mathsf{k})$, $\mathsf{Enc}(\mathsf{c})$ and $\mathsf{Dec}(\mathsf{y})$ record the respective events in the execution trace. We use the Tamarin $\mathsf{let}\ \ldots\ \mathsf{in}$ notation for assignments.

Traces and properties. For a rule R, we let $act(R)$ be the action facts of R. For a set of rules P, an execution trace is defined by a sequence of multisets of facts $M_0, M_1, \ldots, M_n$ and a sequence of rules $R_1, \ldots, R_n \in P$ such that, for every $i \in \{1, \ldots, n\}$, M_i can be obtained from M_{i-1} by applying R_i instantiated with a substitution θ_i. We define:

- $facts(\tau, i) = act(R_i)\theta_i$ if R_i is a protocol or network deduction rule. This represents the fact that certain actions

took place at timepoint i.

- $facts(\tau, i) = \{\mathsf{K}(v\theta_i)\}$ if R_i is an intruder deduction rule with conclusion $\{\mathsf{K}(v)\}$. This represents the fact that the adversary knows $v\theta_i$ at timepoint i.

We consider a set of timepoint variables, denoted by $i, j, l, \ldots$, which will be interpreted over rational numbers. A *trace atom* is either a term equality $\mathsf{t}_1 = \mathsf{t}_2$, or a timepoint ordering $i \prec j$, or a timepoint equality $i = j$, or an action fact $\mathsf{F}@i$ for a fact F and timepoint i. A *trace formula* is a first-order logic formula obtained from trace atoms by applying the usual quantification and logical connectives. The satisfaction relation $\tau \vDash \Phi$, for a trace τ and a trace formula Φ, whose variables are all bounded, is defined recursively as expected, with the following notable case: $\tau \vDash \mathsf{F}@i$ if and only if $\mathsf{F} \in facts(\tau, i)$.

For a set of rules P, we let $\mathsf{tr}(P)$ be the set of traces of R. For trace formulas Ψ, Φ, we let:

$$P \vDash \Phi \quad \text{iff} \quad \forall \tau \in \mathsf{tr}(P).\ \tau \vDash \Phi,$$
$$P; \Psi \vDash \Phi \quad \text{iff} \quad \forall \tau \in \mathsf{tr}(P).\ \tau \vDash \Psi \Rightarrow \Phi.$$

For verification of security properties, $(P; \Psi)$ is typically a protocol specification and Φ the property to be verified. Having the component Ψ in a protocol specification can help to express in a concise way some properties that protocol parties should ensure along an execution trace.

Example 4. *Continuing Example 3, the formula* $\Phi_{\mathsf{sec}} : \forall \mathsf{x}, i.\ \mathsf{Key}(\mathsf{x})@i \Rightarrow \neg(\exists j.\ \mathsf{K}(\mathsf{x})@j)$ *says that, if a term* x *is a secret key generated by the first rule, then there is no timepoint at which the intruder knows it. We have* $Q_{\mathsf{keys}} \not\vDash \Phi_{\mathsf{sec}}$*, since the third rule in* Q_{keys} *may decrypt messages published by the second rule. The restriction* $\Psi_{\mathsf{keys}} : \forall \mathsf{x}, i.\ \mathsf{Dec}(\mathsf{x})@i \Rightarrow \neg(\exists j.\ \mathsf{Enc}(\mathsf{x})@j)$ *models a global check performed by the decrypting party ensuring that the message* x *that is received was not produced by the second rule, and then we have* $(Q_{\mathsf{keys}}; \Psi_{\mathsf{keys}}) \vDash \Phi_{\mathsf{sec}}$.

Notation. To simplify presentation, we adopt ProVerif notation that omits connectives $\exists, \forall, @$. A simplified formula $\mathsf{F}_1(\mathsf{x}_1, \mathsf{x}_2) \Rightarrow \mathsf{F}_2(\mathsf{y}_1, \mathsf{y}_2)$ represents $\forall \mathsf{x}_1, \mathsf{x}_2, i.\ \mathsf{F}_1(\mathsf{x}_1, \mathsf{x}_2)@i \Rightarrow \exists \mathsf{y}_1, \mathsf{y}_2, j.\ \mathsf{F}_2(\mathsf{y}_1, \mathsf{y}_2)@j$. More generally, variables to the left of $\Rightarrow$ are universally quantified and those to the right are existentially quantified, and quantifiers are always applied to action facts. For example, $\mathsf{F}(\mathsf{x}) \Rightarrow \mathsf{F}_1(\mathsf{y}) \vee \neg \mathsf{F}_2(\mathsf{y})$ represents $\forall \mathsf{x}, i.\ \mathsf{F}(\mathsf{x})@i \Rightarrow \exists \mathsf{y}, j.\ \mathsf{F}_1(\mathsf{y})@j \vee \neg(\exists \mathsf{y}, j.\ \mathsf{F}_2(\mathsf{y})@j)$. We may still use $@$ when we need to express a timepoint relation. We note, on this occasion, that our models can also be adapted to ProVerif if needed.

III. Formal models for election verifiability

We need two specifications for formal verification: $\mathcal{S}$ - for the considered e-voting protocol, including the procedures for honest parties and the corruption abilities of the adversary; and Φ - for the the desired security property. $\mathcal{S}$ will be defined based on multiset rewriting rules and restrictions, while Φ will be a conjunction of trace formulas referring to executions of $\mathcal{S}$.

Fig. 1: Individual verification procedures.

$\mathsf{R}^0_{\mathsf{ver}}$: **voter verifies the receipt on** BBcast
$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \dashv\{\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ \} \mapsto [\ ]$

$\mathsf{R}^1_{\mathsf{ver}}$: **voter verifies the receipt on** BBtally
$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ] \dashv\{\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})\ \} \mapsto [\ ]$

$\mathsf{R}^2_{\mathsf{ver}}$: **voter verifies there is no ballot on** BBtally
$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{BBtally}(\mathsf{cr}, \bot)\ ] \dashv\{\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \bot)\ \} \mapsto [\ ]$

$\mathsf{R}^0_{\mathsf{ver}}$ **can be combined with restrictions below:**

Ψ_{last} : **the verified ballot is currently the last on BB**
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i \wedge \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \wedge$
$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @l \wedge i \prec l \wedge j \prec l \Rightarrow j \prec i \vee \mathsf{b} = \mathsf{b}'$

Ψ_{mine} : **all ballots currently on BB are cast by** id
$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i \wedge \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \wedge j \prec i$
$\Rightarrow \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @l$

Specification	Voter instructions
$\mathcal{V}_1 : (\mathsf{R}^0_{\mathsf{ver}}, \Psi_{\mathsf{last}})$	verify the last ballot in BBcast
$\mathcal{V}_2 : (\mathsf{R}^0_{\mathsf{ver}}, \Psi_{\mathsf{last}} \wedge \Psi_{\mathsf{mine}})$	as $\mathcal{V}_1$, and ensure all ballots are own
$\mathcal{V}_3 : (\mathsf{R}^1_{\mathsf{ver}})$	verify the ballot directly on BBtally
$\mathcal{V}_4 : (\mathsf{R}^2_{\mathsf{ver}})$	verify abstention directly on BBtally

A. Verifiable e-voting specifications

A *protocol component* is a pair $(\mathcal{R}, \Psi)$, where $\mathcal{R}$ is a set of protocol rules and Ψ is a conjunction of restrictions, i.e. trace formulas that may restrict the execution of rules from $\mathcal{R}$ under certain conditions.

Definition 1. *An* e-voting specification $\mathcal{S}$ *is a triple of protocol components* $(\mathcal{P}, \mathcal{V}, \mathcal{A})$*, where*

- $\mathcal{P}$ *specifies the voting protocol procedures,*
- $\mathcal{V}$ *specifies the individual verification procedures,*
- $\mathcal{A}$ *specifies the corruption abilities of the adversary.*

If $\mathcal{S} = \big((\mathsf{R}_{\mathcal{P}}, \Psi_{\mathcal{P}}), (\mathsf{R}_{\mathcal{V}}, \Psi_{\mathcal{V}}), (\mathsf{R}_{\mathcal{A}}, \Psi_{\mathcal{A}})\big)$*, then* $\mathcal{S} \vDash \Phi$ *if and only if* $\mathsf{R}_{\mathcal{P}} \cup \mathsf{R}_{\mathcal{V}} \cup \mathsf{R}_{\mathcal{A}};\ \Psi_{\mathcal{P}} \wedge \Psi_{\mathcal{V}} \wedge \Psi_{\mathcal{A}} \vDash \Phi$.

While $\mathcal{V}$ may also be considered as part of the protocol specification $\mathcal{P}$, we treat it separately since we analyse the security properties that are ensured by various verification procedures $\mathcal{V}$ in the context of the same basic protocol $\mathcal{P}$ with various adversaries $\mathcal{A}$. We note that $\mathcal{P}$ may also include public verification checks that can be performed by any external party on the bulletin board. When considering particular adversaries, care should be taken that restrictions from $\mathcal{P}$ are still justified for the considered corruption scenario; if not, they should be removed, resulting in a specification $(\mathcal{P}', \mathcal{V}, \mathcal{A})$. Next, we discuss and illustrate in more detail each component of a specification $(\mathcal{P}, \mathcal{V}, \mathcal{A})$.

Example 5. *Continuing Example 1, we present certain rules and restrictions that model protocol procedures* $\mathcal{P}$*:*

Setup. *First rule models the trustee actions, and the second rule the server's:*

$$[\ \mathsf{Fr}(\mathsf{sk})\ ] -\!\![\ \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}))\ ]\!\!\rightarrow [\ \mathsf{Sk}(\mathsf{sk}), \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}))\ ]$$

$$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{cr}), \mathsf{Fr}(\mathsf{pwd})\ ] -\!\![\ \mathsf{BBreg}(\mathsf{cr})\ ]\!\!\rightarrow [\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]$$

The facts $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ *and* $\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})$ *represent a voter with identity* id *that is communicated the public credential* cr *and the password* pwd *at registration. The fact* $\mathsf{Sk}(\mathsf{sk})$ *represents the storage of the secret key by the trustee.*

Voting. *First rule models the voting platform actions, and the second rule the server's:*

$$\begin{aligned}&\mathsf{let}\quad \mathsf{b} = \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r});\quad \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b}\rangle)\quad \mathsf{in}\\ &[\ \mathsf{BBcand}(\mathsf{v}), \mathsf{BBkey}(\mathsf{pkey}), \mathsf{Fr}(\mathsf{r}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]\\ &-\!\![\ \mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow\\ &[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a}\rangle)\ ]\end{aligned}$$

$$\begin{aligned}&\mathsf{let}\quad \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b}\rangle)\quad \mathsf{in}\\ &[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a}\rangle), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]\\ &-\!\![\ \mathsf{a}' = \mathsf{a}, \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\end{aligned}$$

We record events Vote *and* VoteB *for later use.*

Tally. *To model that the last ballot cast for each* cr *is recorded in* $\mathsf{BBtally}$*, we use a rule and a restriction:*

$$[\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] -\!\![\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow [\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$$

$$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j\ \wedge\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ @l \Rightarrow j < i\ \vee\ \mathsf{b} = \mathsf{b}'$$

This operation is publicly verifiable on BB*; adding the above restriction to the model does not mean we trust the server to perform the operation correctly.*

In Figure 1, we present rules and restrictions for individual verification procedures $\mathcal{V}$ that we consider for our case studies. The premises of $\mathsf{R}^{\mathsf{i}}_{\mathsf{ver}}$ contain two facts: one referring to a ballot cast by the voter, and one referring to one present on BB. A basic verification step performed in the rule ensures that these two ballots match. Further verification steps are enabled by the action fact VerB, which records some relevant parameters. The restrictions $\Psi_{\mathcal{V}} \in \{\Psi_{\mathsf{last}}, \Psi_{\mathsf{mine}}\}$ can then express further requirements. The procedure $\mathcal{V}_1$ represents the scenario where voters are instructed (or allowed by BB) to verify only the last ballot cast for their credential on BB. $\mathcal{V}_1$ can be augmented to $\mathcal{V}_2$ using an additional test, modelled by Ψ_{mine}, ensuring that all ballots cast on BB belong to the voter performing the verification. $\mathcal{V}_3$ and $\mathcal{V}_4$ are performed directly on $\mathsf{BBtally}$ after it is computed from BBcast. In practice, voters actually check a hash of the ballot, rather than the full ballot, but we omit this for simplicity.

When modelling an adversary against election verifiability, it is standard to assume that it may corrupt voters, trustees and the communication network. In addition, the adversary may control other parties: registrars, voting servers, voting platforms, etc. We model all such abilities by a set of corruption rules $\mathcal{A}$, which is a parameter of our definition, and is an addition to the standard network and intruder deduction rules.

Example 6. *Continuing Example 5, we present adversarial corruption rules for the trustee, the server and the voter.* ***A corrupted trustee*** *allows the adversary to generate the secret key* sk *of the election, whereas* ***a corrupted server*** *allows the adversary to control the generation and distribution of credentials, as well as direct ballot casting on the bulletin board:*

$$[\ \mathsf{In}(\mathsf{sk})\ ] -\!\![\ \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}))\ ]\!\!\rightarrow [\ \mathsf{Sk}(\mathsf{sk}), \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}))\ ]$$

$$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{cr}'\rangle), \mathsf{Fr}(\mathsf{pwd})\ ] -\!\![\ \mathsf{BBreg}(\mathsf{cr}')\ ]\!\!\rightarrow [\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ]$$

$$[\ \mathsf{In}(\langle \mathsf{cr}, \mathsf{b}\rangle)\ ] -\!\![\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$$

The second rule models a partial compromise where the password is generated honestly, while the public credential adversarially: cr *is received by the voter, and a possibly different* cr' *is published on the bulletin board.*

Corrupted voters *reveal their credentials to the adversary:*

$$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ] -\!\![\ \mathsf{Corr}(\mathsf{id}, \mathsf{cr})\ ]\!\!\rightarrow [\ \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{pwd}\rangle)\ ]$$

Any subset of these rules can define the adversary component $\mathcal{A}$ *complementing protocol procedures* $\mathcal{P}$ *and verification procedures* $\mathcal{V}$ *in an e-voting specification.*

The following definition captures formally the class of protocols that falls within the scope of our verifiability definition. It relies on events that are informally introduced in Section II-A, relating to the set of registered public credentials, the set of eligible candidates, the multiset of cast votes, the multiset of verified votes, the multiset of votes in the result, and the set of corrupted voters.

Definition 2. *An e-voting specification* $(\mathcal{P}, \mathcal{V}, \mathcal{A})$ *is* verifiable *if it relies on fact symbols*

$$\mathsf{BBkey}, \mathsf{BBcand}, \mathsf{BBreg}, \mathsf{BBtally}, \mathsf{Vote}, \mathsf{Verified}, \mathsf{Corr}$$

to record the following events:

- $\mathsf{BBkey}(\mathsf{y})$*: the term* y *is recorded on* BB *as being the public key of the election;*
- $\mathsf{BBcand}(\mathsf{v})$*: the candidate* v *is recorded on* BB *as eligible;*
- $\mathsf{BBreg}(\mathsf{cr})$*: the public credential* cr *is recorded on* BB *as corresponding to an eligible voter;*
- $\mathsf{BBtally}(\mathsf{cr}, \mathsf{b})$*: the ballot* b *is recorded on* BB *as to be tallied for the respective public credential;*
- $\mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$*: a voter with private identity* id *and public credential* cr *casts a vote* v*;*
- $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$*: a voter with private identity* id *and public credential* cr *has successfully performed the verification procedure related to a vote* v*;*
- $\mathsf{Corr}(\mathsf{id}, \mathsf{cr})$*: all private credentials associated to* $(\mathsf{id}, \mathsf{cr})$ *are leaked to the adversary.*

In addition, we require that the equational theory contains a private function symbol open*, and equations associated to it, such that* $\mathsf{open}(\mathsf{b}, \mathsf{y}) = \mathsf{v}$ *if and only if* b *is a valid encoding of the vote* v *with respect to the public key* y*.*

Example 7. *We refer to Examples 5 and 6 and to Figure 1 for an illustration of how events from Definition 2 can be added to an e-voting specification. We complete the requirements of Definition 2 by defining the equation* $\mathsf{open}(\mathsf{enc}(\mathsf{x},\mathsf{y},\mathsf{z}),\mathsf{y}) = \mathsf{x}$ *for the private symbol* open.

B. Symbolic election verifiability

Revoting policy. If a voter with credentials $\langle \mathsf{id}, \mathsf{cr}, \mathsf{pwd} \rangle$ casts several votes $\mathsf{v}^1 \ldots, \mathsf{v}^m$, only one of these votes, say v^i, will be counted in the final outcome, according to the revoting policy in place. If the individual verification procedure is appropriate for the revoting policy, then it should be the case that $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}^j)$ is true in an execution trace if and only if $\mathsf{v}^j = \mathsf{v}^i$. In consequence, all successfully verified votes would be part of the final outcome. However, some systems may allow weaker verification procedures, which can only ensure that a given ballot is present on BBcast. In that case, the formal definition of verifiability needs to consider additional constraints that should be satisfied in order for the verified vote to be counted. We use a (parameterised) formula $\Omega(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ to add the respective constraints for a given triple $(\mathsf{id}, \mathsf{cr}, \mathsf{v})$. In this paper, we are interested in two natural revoting policies: Ω^{last} and Ω^{ok} from Figure 2. Ω^{last} specifies that if the verified vote v is the last one cast, then it is counted. Ω^{ok} is set to be always true, and this requires any verified vote to be counted.

Definition 3. *Consider the trace formulas from Figure 2. For* $\diamond \in \{\circ, \bullet\}$*, we define* $\Phi^{\diamond}_{\mathsf{E2E}} = \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\diamond}_{\mathsf{res}}$*. We say that a verifiable e-voting specification* $\mathcal{S}$ *satisfies symbolic election verifiability if and only if* $\mathcal{S} \vDash \Phi^{\diamond}_{\mathsf{E2E}}$*.*

Fig. 2: Symbolic election verifiability.

Formulas defining symbolic E2E

$\Phi_{\mathsf{iv}} : \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \Omega(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \wedge \mathsf{BBkey}(\mathsf{y}) \Rightarrow \mathsf{v} = \mathsf{open}(\mathsf{b},\mathsf{y})$

$\Phi_{\mathsf{eli}} : \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}) \vee \mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \Rightarrow \mathsf{BBreg}(\mathsf{cr})$

$\Phi_{\mathsf{cl}} : \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{Verified}(\mathsf{id}',\mathsf{cr},\mathsf{v}') \Rightarrow \mathsf{id} = \mathsf{id}'$

$\Phi^{\diamond}_{\mathsf{res}} : \mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \wedge \mathsf{b} \neq \bot \wedge \mathsf{BBkey}(\mathsf{y}) \Rightarrow (\ \mathsf{Vote}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b},\mathsf{y})\ ) \vee \Phi^{\diamond}_{\mathsf{adv}}(\mathsf{cr})$

Two possible cases for $\Phi^{\diamond}_{\mathsf{adv}}$

$\Phi^{\bullet}_{\mathsf{adv}}(\mathsf{cr}) : \mathsf{Corr}(\mathsf{id},\mathsf{cr})$

$\Phi^{\circ}_{\mathsf{adv}}(\mathsf{cr}) : \mathsf{Corr}(\mathsf{id},\mathsf{cr}) \vee \neg\mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}')$

Examples of revote policies Ω

$\Omega^{\mathsf{ok}}(\mathsf{id},\mathsf{cr},\mathsf{v}) : \mathsf{true}$

$\Omega^{\mathsf{last}}(\mathsf{id},\mathsf{cr},\mathsf{v}) : \mathsf{Vote}(\mathsf{id},\mathsf{cr},\mathsf{v})\ @i \wedge \mathsf{Vote}(\mathsf{id},\mathsf{cr},\mathsf{v}')\ @j \Rightarrow j \prec i \vee \mathsf{v} = \mathsf{v}'$

Additional property for homomorphic tally

$\Phi_{\mathsf{cand}} : \mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \wedge \mathsf{b} \neq \bot \wedge \mathsf{BBkey}(\mathsf{y}) \Rightarrow \mathsf{BBcand}(\mathsf{open}(\mathsf{b},\mathsf{y}))$

Possible weakening of individual verifiability

$\Phi^{\mathsf{h}}_{\mathsf{iv}} : \neg\mathsf{Corr}(\mathsf{id},\mathsf{cr}) \Rightarrow \Phi_{\mathsf{iv}}(\mathsf{id},\mathsf{cr})$

Assumptions to achieve multiset-based E2E

$\Psi_{\mathsf{tally1}} : \mathsf{BBreg}(\mathsf{cr}) \Rightarrow \mathsf{BBtally}(\mathsf{cr},\mathsf{b})$

$\Psi_{\mathsf{tally2}} : \mathsf{BBtally}(\mathsf{cr},\mathsf{b})\ @i \wedge \mathsf{BBtally}(\mathsf{cr},\mathsf{b}')\ @j \Rightarrow i = j$

- Φ_{iv} corresponds to individual verifiability, stating the fact that a successful verification should imply that the corresponding vote is counted as intended in the outcome. Formally, Φ_{iv} requires that any ballot b claimed to represent a voter's choice in the final tally contains the vote expected by the voter. Moreover, by public audits, it can be ensured that there is always a ballot recorded on BBtally for every registered public credential cr; this can be the empty ballot $\bot$ if there is no ballot cast for cr.

- Φ_{eli} ensures that, for every voter, successful verification implies that the corresponding public credential is registered as eligible on the bulletin board, and therefore will be accounted for in the final tally. Conversely, every tallied ballot should correspond to a registered credential.

- Φ_{cl} specifies that no clash should occur on public credentials: two distinct voters that successfully verify their ballots should have distinct public credentials.

- $\Phi^{\diamond}_{\mathsf{res}}$ circumscribes adversarial influence on the final result, relying on $\Phi^{\diamond}_{\mathsf{adv}}$ as parameter. We expect the adversary $\mathcal{A}$ to be able to cast votes for corrupted voters. The strongest version of $\Phi^{\diamond}_{\mathsf{res}}$, obtained when $\diamond = \bullet$, specifies that all other votes should be cast by honest voters. Proving $\Phi^{\bullet}_{\mathsf{res}}$ may require trust assumptions about election authorities, as we show for Helios and Belenios. To analyse verifiability in stronger scenarios, e.g. when the authorities are corrupted, we consider a weaker version of $\Phi^{\diamond}_{\mathsf{res}}$, $\Phi^{\circ}_{\mathsf{res}}$, additionally allowing $\mathcal{A}$ to cast ballots for voters who do not verify their votes. For illustration, consider a few scenarios where votes from a particular voter may not be counted: (i) the voter does not vote; (ii) the submitted ballot is dropped by $\mathcal{A}$; (iii) revoting is allowed, and the submitted ballot is replaced by $\mathcal{A}$. In all these cases, the voter can perform the verification procedure associated to the voter choice (abstention or vote). $\Phi^{\circ}_{\mathsf{adv}}$ says that the adversary can cast a vote for the respective public credential only if the voter did not perform the verification procedure, or if its outcome was negative, in which case accountability and dispute resolution mechanisms should be applied [11], [45].

- Φ_{cand} is for the particular case when the tally is based on homomorphic encryption. Then, we need to ensure that ballots encode valid votes, otherwise the adversary may submit multiple votes within a single ballot, annulling the benefits of $\Phi^{\diamond}_{\mathsf{res}}$. For systems where each ballot is decrypted individually, like those based on mixnets, invalid votes can be removed directly from the outcome.

- $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ is Φ_{iv} that is applied only to voters whose credentials are not leaked. For some systems and procedures, only this

weaker version may be ensured. For Helios, we see examples of procedures that achieve Φ_{iv} as well as examples that only achieve weak individual verifiability.

Comparison with [21]. The properties defining $\Phi^{\diamond}_{\mathsf{E2E}}$ are similar in nature to properties defining symbolic verifiability in [21]. There are, however, important differences that allow us to overcome the limitations discussed in the introduction:

– we introduce cr as an argument in Vote and Verified, and we remove id as an argument in BBtally. In [21], these are Vote(id, v), Verified(id, v) and BBtally(id, cr, b). Our version of Vote and Verified makes a stronger connection between these events and public information on BB. In the same spirit, removing id from BBtally makes the verifiability property more transparent (the connection to id is only known to the voting server in [21]).

– the hidden link between id and cr in BBtally requires [21] to come up with additional restrictions (which vary according to the corruption scenario, and are justified by trust assumptions) in order to make a formal and consistent connection between the public information in BBtally and voter information in Vote and Verified. In our case, this connection is directly provided by the public credential cr used both by voter and the public bulletin board. We demonstrate the generality of our approach by applying it to various scenarios in Helios and Belenios, where we only need to switch between $\Phi^{\bullet}_{\mathsf{res}}$ and $\Phi^{\circ}_{\mathsf{res}}$ to choose the appropriate level of security.

– there is one simple consistency property that we need to ensure between cr and id, which is formalised by Φ_{cl}. It is a property that should intuitively hold for any system, and it can be proved instead of taken as a trust assumption.

– we capture revoting by introducing a revote policy Ω and, more generally, a systematic way of linking Verified events to other events in the execution trace.

– we argue that the stronger version of individual verifiability that we propose, Φ_{iv}, should be preferred to $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ whenever possible. It allows to derive verifiability guarantees even for voters who are dynamically or unwillingly compromised.

On the strength of definitions. When the event BBtally(cr, b) occurs in a trace, the formula $\Phi^{\bullet}_{\mathsf{res}}$, used in the strong version of our definition, says that b can be cast by the adversary only if a voter that receives cr at registration is corrupted. On the other hand, in the case of a dishonest registrar, the definition in [21] allows b to be cast by the adversary as soon as any corrupted voter convinces the honest server to accept a ballot for cr. This means that, instantiated for this corruption scenario, our definition is stronger than [21], since the circumstances under which it tolerates a corrupted cast for cr are more strict. As a consequence, we find an attack on $\Phi^{\bullet}_{\mathsf{res}}$ for Belenios in this case, while it is proved secure with respect to the definition in [21]. One may consider that $\Phi^{\bullet}_{\mathsf{res}}$ is too strong and we note that, under appropriate verification procedures, Belenios satisfies $\Phi^{\circ}_{\mathsf{res}}$ in this scenario, i.e. the weaker version of our definition. However, the attack we find on $\Phi^{\bullet}_{\mathsf{res}}$ does also have a notable impact on security: since the server enforces the consistency between voters and their public credentials, an honest voter would be permanently prevented from casting a ballot if another voter has already used the public credential. We also find other practical attacks on Belenios, unrelated to the strength of $\Phi^{\bullet}_{\mathsf{res}}$.

C. Multiset-based election verifiability

As in [14], [21], [27], we introduce a definition of end-to-end verifiability that is based on multisets. We use this definition to argue the soundness of symbolic verifiability under realistic assumptions. Specifically, we aim for assumptions that can be enforced by public audits on the bulletin board, and do not represent additional trust assumptions. We consider the two assumptions from Figure 2, where Ψ_{tally1} requires that for each registered credential there has to be a corresponding ballot (possibly $\bot$) recorded for the tally phase, and Ψ_{tally2} requires that there is at most one ballot recorded for each credential. Both restrictions can be ensured relying on the basic assumption of a public bulletin board, where the list of registered credentials is published at setup time and the list of ballots to be tallied is published at the end of the voting phase. We let $\Psi_{\mathsf{E2E}} = \Psi_{\mathsf{tally1}} \wedge \Psi_{\mathsf{tally2}}$.

We denote multisets by $\{\!|\mathsf{a}_1, \ldots, \mathsf{a}_n|\!\}$, where each element a_i may have multiple occurrences. $A \uplus B$ represents the disjoint union of multisets A and B, where multiplicities of common elements of A and B add up. For an execution trace τ, revote policy Ω and $\diamond \in \{\circ, \bullet\}$, we define the following sets and multisets:

$$\mathsf{Ver}(\tau) = \{\!|(\mathsf{cr}, \mathsf{v}) \mid \exists \mathsf{id}.\ \tau \vDash \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \wedge \Omega(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \wedge \mathsf{v} \neq \bot|\!\}$$

$$\mathsf{Ver}_{\mathsf{vote}}(\tau) = \{\!|\mathsf{v} \mid \exists \mathsf{cr}.\ (\mathsf{cr}, \mathsf{v}) \in \mathsf{Ver}(\tau)|\!\}$$

$$\mathsf{Ver}_{\mathsf{cr}}(\tau) = \{\mathsf{cr} \mid \exists \mathsf{v}.\ (\mathsf{cr}, \mathsf{v}) \in \mathsf{Ver}(\tau)\}$$

$$\mathsf{Adv}^{\diamond}(\tau) = \{\mathsf{cr} \mid \tau \vDash \Phi^{\diamond}_{\mathsf{adv}}(\mathsf{cr})\} \quad \textit{(adversarial credentials)}$$

$$\mathsf{Result}(\tau) = \{\!|\mathsf{open}(\mathsf{b}) \mid \exists \mathsf{cr}.\ \tau \vDash \mathsf{BBtally}(\mathsf{cr}, \mathsf{b}) \wedge \mathsf{b} \neq \bot)|\!\}$$

Definition 4 is a slight generalisation of definitions in [21], [27]: it accounts for revoting; it extends the notion of corrupted voters (covering $\Phi^{\bullet}_{\mathsf{adv}}$) to a generic notion defined by $\Phi^{\diamond}_{\mathsf{adv}}$ (covering both $\Phi^{\bullet}_{\mathsf{adv}}$ and $\Phi^{\circ}_{\mathsf{adv}}$); it offers end-to-end verifiability benefits also to corrupted voters that have verified their vote. In summary, it says that the final result can be partitioned into: votes that have been verified, votes that have not been verified, and additional votes that may be cast by the adversary. Moreover, for each of these sets, there are associated constraints.

Definition 4. *Let Ω be a revoting policy. We say that a trace τ satisfies $\mathsf{MS}^{\diamond}_{\mathsf{E2E}}$, for $\diamond \in \{\circ, \bullet\}$, if there exist multisets V_1, V_2, V_3 such that $\mathsf{Result}(\tau) = V_1 \uplus V_2 \uplus V_3$ and*

1) *$V_1 = \mathsf{Ver}_{\mathsf{vote}}(\tau)$,*
2) *$V_2 = \{\!|\mathsf{v}_1, \ldots, \mathsf{v}_l|\!\}$ and there are mutually distinct $\mathsf{cr}_1, \ldots, \mathsf{cr}_l$ such that $\forall i \in \{1, \ldots, l\}$,*
   - *$\exists \mathsf{id}_i.\ \tau \vDash \mathsf{Vote}(\mathsf{id}_i, \mathsf{cr}_i, \mathsf{v}_i)$, and*
   - *$\mathsf{cr}_i \notin \mathsf{Ver}_{\mathsf{cr}}(\tau) \cup \mathsf{Adv}^{\diamond}(\tau)$,*
3) *$|V_3| < |\ \mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}_{\mathsf{cr}}(\tau)\ |$.*

We denote this by $\tau \Vdash \mathsf{MS}^{\diamond}_{\mathsf{E2E}}$.

Intuitively, we have: 1) all of the verified votes should be part of the final outcome; 2) if a credential is not adversarial, and there is a vote counted for that credential, there can be at most one such vote and it comes from an honest execution of the voting procedure; 3) the number of any additional votes that is part of the final result is bounded by the number of adversarial credentials. In (2) and (3), we exclude credentials for which some voter verified their vote, since these have been counted in (1).

Theorem 1. *For every trace τ and $\diamond \in \{\circ, \bullet\}$, we have $\tau \vDash \Phi^{\diamond}_{\mathsf{E2E}} \wedge \Psi_{\mathsf{E2E}} \Longrightarrow \tau \Vdash \mathsf{MS}^{\diamond}_{\mathsf{E2E}}$, where*

$$\boxed{\begin{array}{lcl} \Phi^{\diamond}_{\mathsf{E2E}} & = & \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\diamond}_{\mathsf{res}}, \\ \Psi_{\mathsf{E2E}} & = & \Psi_{\mathsf{tally1}} \wedge \Psi_{\mathsf{tally2}}. \end{array}}$$

The proof is given in Appendix A.

Corollary 1. *For any e-voting specification $\mathcal{S}$ and $\diamond \in \{\circ, \bullet\}$, we have*

$$\boxed{\mathcal{S} \vDash \Phi^{\diamond}_{\mathsf{E2E}} \wedge \mathcal{S} \vDash \Psi_{\mathsf{E2E}} \Longrightarrow \mathcal{S} \Vdash \mathsf{MS}^{\diamond}_{\mathsf{E2E}}.}$$

We note that, in order to capture a weaker notion of end-to-end verifiability for honest voters, one can define $\Phi^{\diamond}_{\mathsf{E2Eh}} = \Phi^{\mathsf{h}}_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\diamond}_{\mathsf{res}}$. Then, a corresponding notion $\mathsf{MS}^{\diamond}_{\mathsf{E2Eh}}$ has to be defined in order for a soundness result as in Corollary 1 to be shown.

IV. Verification of Helios and Belenios

Helios [6], [22]–[24] and Belenios [25], [26] are extensions of the simple protocol from Example 1 in order to obtain better usability and security properties. The main differences between these protocols are the ballot structure and the role of a special party, called registrar, responsible in Helios and Belenios for generating public credentials and distributing them to voters. They also share a common structure, that is reflected in protocol components for their specification.

A. Common protocol structure and components

Apart from voters V, the parties in these protocols are administrator A, trustees T, registrar VR, voting server VS, voting platform VP and election auditors EA. VR and VS are usually subsumed by the same party in Helios.

Setup phase. A determines the list of eligible candidates and voters $\mathsf{id}_1, \ldots, \mathsf{id}_n$, VR generates public credentials, VS generates passwords, T generate the election key. We have:

$$\mathsf{BBkey} : \mathsf{pk}; \quad \mathsf{BBcand} : \mathsf{v}_1, \ldots, \mathsf{v}_k; \quad \mathsf{BBreg} : \mathsf{cr}_1, \ldots, \mathsf{cr}_n$$

on the bulletin board. In this phase, each voter id obtains a public credential cr and a password pwd.

Helios with identities. The administrator in Helios can decide to have the public credential equal to the voter identity, i.e. $\mathsf{cr} = \mathsf{id}$. In practice, the voter identity can be the real name of the voter, and the cast ballot is displayed on the bulletin board next to it. Our symbolic models also apply in this case.

Voting phase. The voter interacts with VP to construct a ballot b and authenticate with respect to VS using the password obtained in the setup phase. Our symbolic model abstracts the password-based authentication between VP and VS with the help of a hash function, thereby assuming the underlying authentication protocol to be perfect (i.e. only holder of pwd can submit ballots for id). Specifically, VP computes $\mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle)$ and posts $\langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle$ to VS, which matches it with $\langle \mathsf{id}, \mathsf{cr}, \mathsf{pwd} \rangle$ and publishes b on BBcast next to cr.

Individual verification procedures for both Helios and Belenios, are as discussed in Section II-A and presented in Figure 1. We note that the current deployment of these protocols shows only the last ballot cast for each credential on the bulletin board and voters can verify their ballot anytime [24], [25]. With respect to Figure 1, this means that: $\mathcal{V}_1$ is the procedure regularly used; $\mathcal{V}_3$ and $\mathcal{V}_4$ can be used (voters can check their ballots after the results have been released); and $\mathcal{V}_2$ cannot be used (since previously cast ballots are not visible).

Tally phase. The last ballot present on BBcast for each credential is chosen for $\mathsf{BBtally}$. We have:

$$\mathsf{BBtally} : (\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n),$$

where $\mathsf{b} = \perp$ if no ballot was cast for cr. The ciphertexts corresponding to non-empty ballots on $\mathsf{BBtally}$ are tallied by T, who announce the final result along with a zero-knowledge proof that it corresponds to input ciphertexts.

Adversarial models. In addition to the generic facts required by Definition 2 and described in Section III, the protocol specifications for Helios and Belenios use the following facts to model the protocol state:

- $\mathsf{Sk}(\mathsf{x})$: to represent the secret key of the election;
- $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$: to represent that a voter with private identity id is communicated the public credential cr;
- $\mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{skey})$: for Belenios, we override Reg to include a private signing key skey, where cr is the corresponding public verification key;
- $\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})$: to represent that the voter receives the password pwd for connecting to the server;
- $\mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})$: to represent that the voter has cast a ballot b encoding the desired vote;
- $\mathsf{VScast}(\mathsf{id}, \mathsf{b})$: to represent that the ballot b is received by VS for the respective id.

These facts are also used in rules and restrictions that model the corruption abilities of the adversary in Figure 3. Corrupted trustees, voters and server are modelled as in Example 6. The rule $\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}$ corrupts registration, allowing the adversary to construct, allocate and communicate credentials to voters. Adversarial inputs are not constrained (in particular we may have $\mathsf{cr}' \neq \mathsf{cr}$), letting the adversary choose the desired attack strategy, e.g. for mounting a clash attack. The rule $\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$ models a corrupted server that forgoes the prescribed way of validating ballots before casting them on BB. The rule $\mathcal{C}^{\mathsf{VP}}_{\mathsf{vote}}$ represents a corrupted voting platform which allows the adversary to select the randomness used in the ballot encoding the vote.

Fig. 3: Adversarial models for Helios and Belenios.

$\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}$: **corrupt trustee to control the secret key**
[In(sk)]−[BBkey(pk(sk))]→[Sk(sk), BBkey(pk(sk))]

$\mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}$: **corrupt voter to reveal credentials (no** skey **in Helios)**
[Reg(id, cr, skey), Pwd(id, pwd)]
−[Corr(id, cr)]→[Out(⟨id, cr, skey, pwd⟩)]

$\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$: **corrupt server to stuff ballots**
[In(⟨cr, b⟩)]−[BBcast(cr, b)]→[BBcast(cr, b)]

$\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}$: **corrupt registration of credentials (no** skey **in Helios)**
[In(⟨id, cr, skey, cr′⟩)]−[BBreg(cr′)]→
[Reg(id, cr, skey), BBreg(cr′)]

$\mathcal{C}^{\mathsf{VP}}_{\mathsf{vote}}$: **corrupt platform to choose randomness**
rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ where Fr(r) is replaced by In(r)

Ψ_{order} : **ensure ballots are delivered in the right order**
$\mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i\ \wedge\ \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @j\ \wedge$
$\mathsf{VScast}(\mathsf{id}, \mathsf{b})\ @k\ \wedge\ \mathsf{VScast}(\mathsf{id}, \mathsf{b}')\ @l\ \wedge\ i \prec j \Rightarrow k \prec l$

Specification	**Corrupted parties**
$\mathcal{A}_1 : \mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}, \Psi_{\mathsf{order}}$	trustees and voters
$\mathcal{A}_2 : \mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}, \mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$	trustees, voters and server
$\mathcal{A}^*_3 : \mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}, \mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}, \Psi_{\mathsf{order}}$	trustees, voters and registrar
$\mathcal{A}_4 : \mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}, \mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}, \mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$	trustees, voters, registrar and server
$\mathcal{A}_5 : \mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}, \mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}, \mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}, \mathcal{C}^{\mathsf{VP}}_{\mathsf{vote}}$	trustees, voters, registrar, server and voting platform

* : $\mathcal{A}_3$ does not apply to Helios.

Restrictions can be used to reduce the adversarial power, when it is justified by the considered scenario. For example, if the server is honest and a voter casts multiple ballots, we can assume that they are processed by the server in the right order. We can model this by the restriction Ψ_{order} used in conjunction with the corresponding adversarial rules. We assemble these rules in five adversarial models $\mathcal{A}_i$ that we consider for verification. The registrar in Helios is also in charge of authenticating voters (that is, it plays the role of VR and VS), thus the adversary $\mathcal{A}_3$ (only corrupting VR) does not naturally apply to it. The adversary $\mathcal{A}_2$, however, does apply to the variant of Helios in [23], where a special party is in charge of posting ballots and managing the BB.

B. Helios specification and verification

We perform automated verification of several scenarios, finding both security proofs and attacks with Tamarin (which typically takes less than 10 minutes to terminate). Each scenario is formally defined as an e-voting specification $(\mathcal{P}, \mathcal{V}_i, \mathcal{A}_j)$ assembling a protocol component $\mathcal{P} \in \{\mathcal{P}_{\mathcal{H}}, \mathcal{P}_{\mathcal{B}}\}$ (corresponding to Helios or Belenios), an individual verification procedure $\mathcal{V}_i$ from Figure 1 and adversarial corruption rules $\mathcal{A}_j$ from Figure 3. The Tamarin code is available online [46] and details for $\mathcal{P}_{\mathcal{H}}$ and $\mathcal{P}_{\mathcal{B}}$ are in Appendix B. The specification $\mathcal{P}_{\mathcal{H}}$ is similar to Example 5, the most notable additions being rules that model the actions of the administrator and of election auditors. The administrator rules allow to determine a set of facts $\mathsf{Id}(\mathsf{id}_1), \ldots, \mathsf{Id}(\mathsf{id}_n)$, respectively $\mathsf{BBcand}(\mathsf{v}_1), \ldots, \mathsf{BBcand}(\mathsf{v}_k)$, that represent the set of eligible voters, respectively eligible candidates. These rules take their input from the environment, allowing to cover any desired scenario within the Tamarin exploration space.

In addition to the encryption of the vote, the ballot is accompanied by a zero-knowledge proof if a homomorphic tally is used, as in the current deployment of Helios. Therefore, we extend the ballot structure as $\mathsf{b} = \langle \mathsf{c}, \mathsf{pr}_1 \rangle$, where $\mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r})$ and $\mathsf{pr}_1 = \mathsf{proof}_1(\mathsf{c}, \mathsf{r}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle)$. The zero-knowledge proof pr_1 shows that the encrypted vote v is within the valid range $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle$. To model the cryptographic primitives, we consider the symbols $\mathsf{pk}, \mathsf{enc}, \mathsf{dec}, \mathsf{h}, \mathsf{proof}_1, \mathsf{ver}_1$ and the equations (1)-(2) described in Example 2. We note that, in order to use the zero-knowledge proof equations (2) in Tamarin, we have to fix in advance the number of candidates. This is only required for proving Φ_{cand}, which is specific to the systems using homomorphic encryption; all other properties do not require the zero-knowledge proof and can be analysed without fixing the number of candidates apriori. The specification $\mathcal{P}_{\mathcal{H}}$ includes a restriction that corresponds to a test that can be performed by election auditors on BB to ensure that all cast ballots encode valid votes and correspond to registered credentials:

$$\Psi_{\mathsf{cast}} : \mathsf{BBcast}(\mathsf{cr}, \langle \mathsf{c}, \mathsf{pr}_1 \rangle) \Rightarrow \mathsf{BBreg}(\mathsf{cr}) \wedge \mathsf{BBkey}(\mathsf{pkey}) \wedge \mathsf{BBcand}(\mathsf{vlist}) \wedge \mathsf{ver}_1(\mathsf{pr}_1, \mathsf{c}, \mathsf{pkey}, \mathsf{vlist}) = \mathsf{ok}$$

For applying the security definition, we adapt the equation for open according to the ballot structure in Helios: $\mathsf{open}(\langle \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r}), \mathsf{pr}_1 \rangle, \mathsf{pk}) = \mathsf{v}$. Verification results obtained with Tamarin are presented in Table I, where $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_j)$ is represented by $[\mathcal{V}_i, \mathcal{A}_j]$. We discuss some notable results for each corruption scenario. Recall that $\Phi^{\diamond}_{\mathsf{E2E}} = \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\diamond}_{\mathsf{res}}$, where $\diamond = \circ$ corresponds to weak, while $\diamond = \bullet$ corresponds to strong verifiability. Unless otherwise specified, the revoting policy is Ω^{ok} from Figure 2.

We recall that Helios is subject to *ballot stuffing attack* by a corrupted server: it can add ballots for abstaining voters and, when revoting is allowed, it can also do it for voters that voted and verified their vote immediately after casting.

Adversary $\mathcal{A}_1$**.** If the procedure $\mathcal{V}_1$ is used, corrupted voters are subject to a violation of individual verifiability, thus we have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_i) \nvDash \Phi_{\mathsf{iv}}$ for any i, since $\mathcal{A}_i$ can use the voter password to cast a ballot after the voter has verified. For honest voters, we can prove $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_1) \vDash \Phi^{\mathsf{h}}_{\mathsf{iv}}$ when used with revoting policy Ω^{last}. All other properties are provable for $\mathcal{A}_1$, hence we have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_1) \vDash \Phi^{\mathsf{h}}_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\bullet}_{\mathsf{res}}$, amounting to an end-to-end verifiability property for honest voters. We also have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_1) \vDash \Phi^{\bullet}_{\mathsf{E2E}}$, for $i \in \{3, 4\}$, guaranteeing end-to-end verifiability even for corrupted voters.

Adversary $\mathcal{A}_2$**.** We have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_2) \nvDash \Phi^{\bullet}_{\mathsf{res}}$ for any i, which represents the well-known ballot stuffing attack in Helios

TABLE I: Verifiability Analysis of Helios

$[\mathcal{V}_i, \mathcal{A}_j]/\Phi_{\text{type}}$	$\Phi^{\dagger}_{\text{iv}}$	Φ_{eli}	Φ_{cl}	$\Phi^{\bullet}_{\text{res}}$	$\Phi^{\circ}_{\text{res}}$
$[\mathcal{V}_i, \mathcal{A}_1]$, $i \in \{1,2\}$	✓*	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_1]$, $i \in \{3,4\}$	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2]$, $i \in \{1,2\}$	✗$^{\times}$	✓	✓	✗	✗
$[\mathcal{V}_i, \mathcal{A}_2]$, $i \in \{3,4\}$	✓	✓	✓	✗	✓
$[\mathcal{V}_1, \mathcal{A}_4]$	✗$^{\times}$	✓	✗	✗	✗
$[\mathcal{V}_2, \mathcal{A}_4]$	✗$^{\times}$	✓	✓	✗	✗
$[\mathcal{V}_3, \mathcal{A}_4]$	✓	✓	✓	✗	✓
$[\mathcal{V}_4, \mathcal{A}_4]$	✓	✓	✗	✗	✓
$[\mathcal{V}_1, \mathcal{A}_5]$	✗$^{\times}$	✓	✗	✗	✗
$[\mathcal{V}_2, \mathcal{A}_5]$	✗$^{\times}$	✓	?*	✗	✗
$[\mathcal{V}_i, \mathcal{A}_5]$, $i \in \{3,4\}$	✓	✓	✗	✗	✓

† : Revote policies: Ω^{last} for $\mathcal{V}_1, \mathcal{V}_2$ and Ω^{ok} for $\mathcal{V}_3, \mathcal{V}_4$.
* : Verification is for $\Phi^{\text{h}}_{\text{iv}}$ with revote policy Ω^{last}.
$^{\times}$: $\Phi^{\text{h}}_{\text{iv}}$ is also violated.
* : Tamarin execution does not terminate.

when the server is corrupted. On the positive side, we have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_2) \vDash \Phi^{\circ}_{\text{E2E}}$, for $i \in \{3,4\}$.

Adversary $\mathcal{A}_4$**.** We have $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_3, \mathcal{A}_4) \vDash \Phi^{\circ}_{\text{E2E}}$, which represents the first automated proof of end-to-end verifiability for Helios when the server is fully corrupted. We note that $\mathcal{V}_3$ is the procedure performed after the voting phase, whereas $\mathcal{V}_1$ represents the procedure currently deployed and used in [24]. For $\mathcal{V}_1$, in addition to the ballot stuffing attack (as for $\mathcal{A}_2$), we find $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_1, \mathcal{A}_4) \nvDash \Phi_{\text{cl}}$. This corresponds to a new version of the clash attack of [6], originally applicable by $\mathcal{A}_5$. The attack scenario is presented in $\mathsf{AS}^1_{\mathcal{H}}$. It can be seen that, if the registrar assigns the same credential to two voters, they can both be happy with their verification results, by verifying BBcast at different points in time. Election auditors cannot detect any irregularity on BB, since the public sequence of ballots is consistent with revoting performed by cr_1. VS can then stuff a ballot for cr_2, in order to create the impression that ballots from all voters have been taken into account.

The attack $\mathsf{AS}^1_{\mathcal{H}}$ is particularly effective when BBcast shows only the last ballot cast for each credential, as in the current deployment. We note however that other deployment options for BBcast also suffer from similar attacks. For example, assume all ballots cast by each voter are published on the bulletin board. Then, adapting the above scenario, after id_2 casts b_2 we have $(\mathsf{cr}_1, \mathsf{b}_1), (\mathsf{cr}_1, \mathsf{b}_2) \in \mathsf{BBcast}$. The voter id_2 has a chance to spot that there is a problem related to cr_1. However, depending on how the bulletin board is presented to id_2 and on the actual voter instruction, the attack may still be undetected. An interesting version of this attack is when id_1 and id_2 choose to abstain. In that case, they can still verify the bulletin board to ensure that there is no ballot cast in their name. A corrupted registrar giving the same cr_1 to $\mathsf{id}_1, \mathsf{id}_2$ can, again, cast an adversarial ballot. This entails $(\mathcal{P}_{\mathcal{H}}, \mathcal{V}_4, \mathcal{A}_4) \nvDash \Phi_{\text{cl}}$. Remarkably, this attack is more difficult to prevent than before.

Attack Scenario $\mathsf{AS}^1_{\mathcal{H}}$ ***(clash attack by*** $\mathcal{A}_4$***)***

1. VR creates $\mathsf{cr}_1, \mathsf{cr}_2$, and gives cr_1 to both $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$, resulting in $\mathsf{Reg}(\mathsf{id}_1, \mathsf{cr}_1)$ and $\mathsf{Reg}(\mathsf{id}_2, \mathsf{cr}_1)$.
2. $\mathsf{V}(\mathsf{id}_1)$ posts a ballot b_1 and verifies $(\mathsf{cr}_1, \mathsf{b}_1) \in \mathsf{BBcast}$.
3. $\mathsf{V}(\mathsf{id}_2)$ posts a ballot b_2 and verifies $(\mathsf{cr}_1, \mathsf{b}_2) \in \mathsf{BBcast}$.
4. EA sees $(\mathsf{cr}_1, \mathsf{b}_1)$ followed by $(\mathsf{cr}_1, \mathsf{b}_2)$ on BBcast.

Outcome: only one ballot will be tallied for id_1 and id_2.

Attack Scenario $\mathsf{AS}^2_{\mathcal{H}}$ ***(clash attack by*** $\mathcal{A}_5$***)***

1. Same operation as $\mathsf{AS}^1_{\mathcal{H}}$.
2. VP prepares the same ballot b for $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$.
3. $\mathsf{V}(\mathsf{id}_1)$ posts the ballot b and verifies $(\mathsf{cr}_1, \mathsf{b}) \in \mathsf{BBtally}$.
4. $\mathsf{V}(\mathsf{id}_2)$ posts the ballot b and verifies $(\mathsf{cr}_1, \mathsf{b}) \in \mathsf{BBtally}$.

Outcome: only one ballot will be tallied for id_1 and id_2.

Our analysis shows that some procedures that protect participating voters do not protect the abstainers. The attack $\mathsf{AS}^1_{\mathcal{H}}$ can be prevented by $\mathcal{V}_3$, i.e. verifying ballots directly on BBtally, or by $\mathcal{V}_2$, i.e. using the restriction Ψ_{mine} to ensure all ballots on BBcast are recognised as theirs by voters. However, it is not possible to prevent clash attack on abstaining voters; (i) there is no ballot on BBcast, thus Ψ_{mine} does not apply in that case, (ii) verification on BBtally also does not help, since two abstaining voters would both be happy seeing no ballot.

Adversary $\mathcal{A}_5$**.** We capture the clash attack of [6], presented in $\mathsf{AS}^2_{\mathcal{H}}$. There are some notable differences from the scenario of $\mathsf{AS}^1_{\mathcal{H}}$: voters id_1 and id_2 are assumed to vote for the same candidate; VP of both voters is assumed corrupted; the attack is possible even when revoting is disallowed and voters perform the stronger verification procedure $\mathcal{V}_3$. Here, the adversary also needs to create a clash on ballots, and not only on public credentials, since there is at most one ballot on BBtally for each credential. That is where it relies on the voting platform.

Helios with identities. We note that the clash attacks by $\mathcal{A}_4$ and $\mathcal{A}_5$ do not apply when we have $\mathsf{cr} = \mathsf{id}$, since we assume a trusted administrator that assigns unique identities to voters.

C. Belenios specification and verification

The Belenios voting protocol is an extension of Helios in order to ensure stronger security properties when some election authorities may be corrupted [26]. The most important change in Belenios with respect to Helios is the addition of signatures in order to attest that ballots come from eligible voters associated to a given credential. The zero-knowledge proof in Belenios has an additional feature, ensuring the ciphertext in the ballot is linked to a given public credential. Symbolically, we represent this additional feature relying on new symbols $\mathsf{proof}_2, \mathsf{ver}_2$ and we use equations (1)-(4) from Example 2 to model all cryptographic properties required for Belenios.

TABLE II: Verifiability Analysis of Belenios

$[\mathcal{V}_\mathbf{i}, \mathcal{A}_\mathbf{j}]/\mathbf{\Phi}_{\mathsf{type}}$	$\Phi^{\dagger}_{\mathsf{iv}}$	Φ_{eli}	Φ_{cl}	$\Phi^{\bullet}_{\mathsf{res}}$	$\Phi^{\circ}_{\mathsf{res}}$
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{1,2\}$	✗×	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{3,4\}$	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{1,2\}$	✗×	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{3,4\}$	✓	✓	✓	✓	✓
$[\mathcal{V}_1, \mathcal{A}_j],\ j \in \{3,4\}$	✗×	✓	✗	✗	✗
$[\mathcal{V}_2, \mathcal{A}_j],\ j \in \{3,4\}$	✗×	✓	✓	✗	✗
$[\mathcal{V}_3, \mathcal{A}_j],\ j \in \{3,4\}$	✓	✓	✓	✗	✓
$[\mathcal{V}_4, \mathcal{A}_j],\ j \in \{3,4\}$	✓	✓	✗	✗	✓
$[\mathcal{V}_1, \mathcal{A}_5]$	?*	✓	✗	✗	✗
$[\mathcal{V}_2, \mathcal{A}_5]$	?*	✓	?*	✗	✗
$[\mathcal{V}_i, \mathcal{A}_5],\ i \in \{3,4\}$	✓	✓	✗	✗	✓

† : Revote policies: Ω^{last} for $\mathcal{V}_1, \mathcal{V}_2$ and Ω^{ok} for $\mathcal{V}_3, \mathcal{V}_4$.
× : $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ is also violated.
* : Tamarin execution does not terminate.

The registrar VR has the special role of generating a key pair - a signing key skey and corresponding verification key cr - for each eligible voter. The ballot constructed by VP contains, in addition to the ciphertext c, a signature s on c with respect to skey as well as the zero-knowledge proof(s). That is, we have $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr}_1, \mathsf{pr}_2 \rangle$ in our symbolic representation. VP sends $\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle$ to VS, which authenticates the voter and checks the validity of the signature and of proofs. In addition, it records $(\mathsf{id}, \mathsf{cr})$ in a log, for which the following consistency property is ensured throughout the execution of the election: $(\mathsf{id}, \mathsf{cr}) \in \mathsf{log} \ \wedge \ (\mathsf{id}, \mathsf{cr}') \in \mathsf{log} \ \Rightarrow \ \mathsf{cr} = \mathsf{cr}'$ and $(\mathsf{id}, \mathsf{cr}) \in \mathsf{log} \ \wedge \ (\mathsf{id}', \mathsf{cr}) \in \mathsf{log} \ \Rightarrow \ \mathsf{id} = \mathsf{id}'$. We should note that it is the voter who communicates the pair $(\mathsf{id}, \mathsf{cr})$ to VS, and not the registrar.

The protocol specification $\mathcal{P}_\mathcal{B}$ is similar to $\mathcal{P}_\mathcal{H}$, following the differences sketched above. For example, we have an extended version of Ψ_{cast}, in order to model the fact that election auditors also check signatures to ensure the validity of ballots on BBcast. We also have a restriction Ψ_{log} to model the consistency property ensured for the log on server. For applying the security definition, we adapt the equation for open according to the ballot structure in Belenios: $\mathsf{open}(\langle \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r}), \mathsf{s}, \mathsf{pr}_1, \mathsf{pr}_2 \rangle, \mathsf{pk}) = \mathsf{v}$. Verification results obtained with Tamarin are presented in Table II. We discuss some notable results in each scenario. For illustrating attacks, we consider two voters id_1 and id_2.

Adversary $\mathcal{A}_1$**.** We expect weak individual verifiability to hold with revote policy Ω^{last}, as in the case of Helios. However, we find $(\mathcal{P}_\mathcal{B}, \mathcal{V}_1, \mathcal{A}_1) \not\models \Phi^{\mathsf{h}}_{\mathsf{iv}}$. This can be explained by the attack scenario presented in $\mathsf{AS}^1_\mathcal{B}$, where we assume id_1 and id_2 have public credentials cr_1 and cr_2, respectively, and that the adversary controls the communication network. The step 3 of the attack is possible because b_2 contains a valid signature with respect to cr_1 and VS is not aware that cr_1 corresponds to id_1. This attack is not possible in Helios since the registrar and the server agree on the correspondence between identities and credentials, and therefore corrupted voters cannot cast ballots for any credential other than theirs. We note that [26] also mentions a potential alternative design for Belenios where the registrar communicates the log to the server before the election starts. That version would not suffer from this attack, and the ballot stuffing attack by $\mathcal{A}_3$ shown below would be less serious. The version we analyse was preferred for deployment since it promises everlasting privacy, yet it does pose new problems as we show.

Attack Scenario $\mathsf{AS}^1_\mathcal{B}$ ***(ballot reordering by*** $\mathcal{A}_1$***)***

1. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b_1 followed by another ballot b_2.
2. $\mathcal{A}$ blocks b_1 and b_2, corrupts $\mathsf{V}(\mathsf{id}_2)$ and casts b_2 using credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
3. VS accepts b_2 coming from $(\mathsf{id}_2, \mathsf{cr}_1)$ and publishes $(\mathsf{cr}_1, \mathsf{b}_2)$ on BBcast.
4. $\mathsf{V}(\mathsf{id}_1)$ successfully verifies $(\mathsf{cr}_1, \mathsf{b}_2) \in$ BBcast.
5. $\mathcal{A}$ casts b_1 using credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
6. VS accepts b_1 and publishes $(\mathsf{cr}_1, \mathsf{b}_1)$ on BBcast.

Outcome: b_1 is tallied for cr_1, even if b_2 is the last ballot was cast and verified by id_1.

Adversary $\mathcal{A}_2$**.** We obtain $(\mathcal{P}_\mathcal{B}, \mathcal{V}_i, \mathcal{A}_2) \models \Phi^{\bullet}_{\mathsf{E2E}}$ for $i \in \{3,4\}$, showing that Belenios indeed satisfies a stronger verifiability property than Helios in this case. We note, however, that the case with $\mathcal{V}_1$ corresponds to the currently recommended procedure in Belenios [25], [26]. We should be able to prove $(\mathcal{P}_\mathcal{B}, \mathcal{V}_1, \mathcal{A}_2) \models \Phi^{\mathsf{h}}_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\bullet}_{\mathsf{res}}$, but a variant of the ballot reordering attack presented for $\mathcal{A}_1$ prevents this, i.e. we have $(\mathcal{P}_\mathcal{B}, \mathcal{V}_1, \mathcal{A}_2) \not\models \Phi^{\mathsf{h}}_{\mathsf{iv}}$ and $(\mathcal{P}_\mathcal{B}, \mathcal{V}_1, \mathcal{A}_2) \models \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\bullet}_{\mathsf{res}}$.

Adversary $\mathcal{A}_3$**.** We expect $(\mathcal{P}_\mathcal{B}, \mathcal{V}_i, \mathcal{A}_3) \models \Phi^{\bullet}_{\mathsf{res}}$, however we find $(\mathcal{P}_\mathcal{B}, \mathcal{V}_i, \mathcal{A}_3) \not\models \Phi^{\bullet}_{\mathsf{res}}$, for any i. The weaker property $\Phi^{\circ}_{\mathsf{res}}$ is also not satisfied for $i \in \{1,2\}$. Before presenting the attacks, we should mention (and rule out) some trivial attacks by a corrupted VR, which consist in distributing to the voter a cr that is not on BBreg, or a skey that does not correspond to cr. These attacks can be detected by audits before the election. The attack that we find, presented in $\mathsf{AS}^2_\mathcal{B}$, can only be observed by the voter while trying to cast a ballot, and it is more difficult to make VR accountable for it. In addition, the adversary does not have to control the communication network for this attack. Formally, the formula $\Phi^{\bullet}_{\mathsf{res}}$ is violated as we obtain $\mathsf{BBtally}(\mathsf{cr}_1, \mathsf{b}_\mathcal{A})$ and cr_1 does not correspond to a corrupted voter and $\mathsf{b}_\mathcal{A}$ does not correspond to an honest execution of the voting procedure. In practice, the consequence is that the voter id_1 is not able to cast a vote, no matter what infrastructure is used.

We have $(\mathcal{P}_\mathcal{B}, \mathcal{V}_i, \mathcal{A}_3) \not\models \Phi^{\mathsf{h}}_{\mathsf{iv}} \ \vee \ \Phi^{\circ}_{\mathsf{res}}$ for $i \in \{1,2\}$. Indeed, we have the scenario $\mathsf{AS}^3_\mathcal{B}$ as a variation of the previous attack, but where we have to assume that the adversary controls the communication network. Reminiscent from Helios, we also find the clash attack on empty ballots in this case, i.e.

Attack Scenario $\mathsf{AS}^2_{\mathcal{B}}$ ***(weak ballot stuffing by*** $\mathcal{A}_3$***)***

1. VR registers $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$ with $\mathsf{Reg}(\mathsf{id}_1, \mathsf{cr}_1, \mathsf{skey}_1)$ and $\mathsf{Reg}(\mathsf{id}_2, \mathsf{cr}_2, \mathsf{skey}_2)$ respectively.
2. $\mathcal{A}$ corrupts $\mathsf{V}(\mathsf{id}_2)$ and VR and obtains $\langle \mathsf{pwd}_2, \mathsf{skey}_1 \rangle$.
3. $\mathcal{A}$ casts a ballot $\mathsf{b}_{\mathcal{A}}$ using the credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
4. VS accepts $\mathsf{b}_{\mathcal{A}}$ and publishes $(\mathsf{cr}_1, \mathsf{b}_{\mathcal{A}})$ on BBcast.
5. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b with credentials $(\mathsf{id}_1, \mathsf{cr}_1)$.
6. VS sees inconsistency in logs: $(\mathsf{id}_2, \mathsf{cr}_1)$ vs. $(\mathsf{id}_1, \mathsf{cr}_1)$, and does not accept b, or any other ballot from $\mathsf{V}(\mathsf{id}_1)$.

Outcome: id_1 cannot vote, instead, $\mathsf{b}_{\mathcal{A}}$ is tallied for cr_1.

Attack Scenario $\mathsf{AS}^3_{\mathcal{B}}$ ***(strong ballot stuffing by*** $\mathcal{A}_3$***)***

1-2. Same steps as $\mathsf{AS}^2_{\mathcal{B}}$.
3. $\mathsf{V}(\mathsf{id}_1)$ casts a ballot b using the credentials $(\mathsf{id}_1, \mathsf{cr}_1)$.
4. $\mathcal{A}$ blocks b, casts it using the credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
5. VS accepts b and publishes $(\mathsf{cr}_1, \mathsf{b})$ on BBcast.
6. $\mathsf{V}(\mathsf{id}_1)$ successfully verifies $(\mathsf{cr}_1, \mathsf{b}) \in$ BBcast.
7. $\mathcal{A}$ casts another ballot $\mathsf{b}_{\mathcal{A}}$ for $(\mathsf{id}_2, \mathsf{cr}_1)$.
8. VS accepts $\mathsf{b}_{\mathcal{A}}$ and publishes $(\mathsf{cr}_1, \mathsf{b}_{\mathcal{A}})$ on BBcast.

Outcome: $\mathsf{b}_{\mathcal{A}}$ is tallied for cr_1, even if b is the only ballot that was cast and verified by id_1.

$(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_4, \mathcal{A}_3) \not\vdash \Phi_{\mathsf{cl}}$. Moreover, we find $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_1, \mathcal{A}_3) \not\vdash \Phi_{\mathsf{cl}}$, which is not expected. It can be explained by the fact that, like in the case of individual verifiability, we require resistance to clash attacks even if some of the targeted voters are corrupted. We then find a clash attack against two voters, presented in $\mathsf{AS}^4_{\mathcal{B}}$, where one is honest and the other is corrupted.

Comment on results for $\mathcal{A}_2$ ***and*** $\mathcal{A}_3$**.** We note that $\mathsf{AS}^2_{\mathcal{B}}$ and $\mathsf{AS}^4_{\mathcal{B}}$ are not attacks against the definitions of [21], [27], since there is a corrupted voter involved in casting the tallied ballot. On the other hand, $\mathsf{AS}^1_{\mathcal{B}}$ and $\mathsf{AS}^3_{\mathcal{B}}$ are attacks against their definitions, since the honest voter successfully verified a ballot and this ballot was not tallied as it should be. The attacks are missed by the symbolic analysis of [21] because it does not consider revoting. They are missed by the computational analysis of [27] because it assumes that voters verify their ballots only after the voting phase, corresponding to $\mathcal{V}_3$ in our case, while [27] leaves open the question of security for $\mathcal{V}_1$, which is exploited by these attacks. Although our analysis shows negative results for $\mathcal{V}_1$ in the presence of both $\mathcal{A}_2$ and $\mathcal{A}_3$, there are important differences between these cases. In the case of $\mathcal{A}_3$, we have substantial attacks that we do not know how to fix immediately. In the case of $\mathcal{A}_2$, although the results of our analysis are negative for $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ (the cause being a ballot reordering attack like $\mathsf{AS}^1_{\mathcal{B}}$), we think the answer could be positive if, as suggested in [27], an appropriate audit of BB is performed and voters are asked to verify each of their ballots after casting. Note also that $\Phi^{\bullet}_{\mathsf{res}}$ is satisfied for $\mathcal{A}_2$, showing ballot stuffing is impossible, independently of the verification procedure. This is not the case for $\mathcal{A}_3$, where ballot stuffing is possible even for voters that verified their ballots (with $\mathcal{V}_1$).

Attack Scenario $\mathsf{AS}^4_{\mathcal{B}}$ ***(clash attack by*** $\mathcal{A}_3$***)***

1. VR registers $\mathsf{V}(\mathsf{id}_1)$ and $\mathsf{V}(\mathsf{id}_2)$ with the same $\langle \mathsf{cr}, \mathsf{skey} \rangle$.
2. $\mathsf{V}(\mathsf{id}_2)$ submits ballot b_2 and verifies $(\mathsf{cr}, \mathsf{b}_2) \in$ BBcast.
3. $\mathsf{V}(\mathsf{id}_1)$ submits ballot b_1.
4. $\mathcal{A}$ blocks b_1, corrupts $\mathsf{V}(\mathsf{id}_2)$ and submits b_1 in the name of $\mathsf{V}(\mathsf{id}_2)$.
5. VS accepts b_1, seeing no inconsistency in logs and publishes $(\mathsf{cr}, \mathsf{b}_1)$ on BBcast.
6. $\mathsf{V}(\mathsf{id}_1)$ successfully verifies $(\mathsf{cr}, \mathsf{b}_1) \in$ BBcast.

Outcome: only one ballot will be tallied for id_1 and id_2, even if both perform a successful verification.

Adversary $\mathcal{A}_4$**.** When both the server and the registrar are corrupted, we find, as expected, the same results for Belenios as for Helios. Most notably, we have the positive result $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_3, \mathcal{A}_4) \vdash \Phi^{\circ}_{\mathsf{E2E}}$.

Adversary $\mathcal{A}_5$**.** We have $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_4) \not\vdash \Phi^{\circ}_{\mathsf{E2E}}$ for any i, since we recover the classic clash attack as in Helios.

V. Conclusion and future work

We have introduced a symbolic framework that allows to use automated tools in order to evaluate end-to-end election verifiability for a large class of protocols and corruption models. It is more expressive than all existing symbolic frameworks. We have applied it to new scenarios in Helios and Belenios, and with the Tamarin prover, we have discovered new attacks and new security proofs. In general, our results show that the deployed versions of these systems do not satisfy the desired level of security, yet we prove positive results, sometimes under appropriate verification procedures, showing that a higher level of security is possible as well. Finding the right balance between usability and security can be seen as the main open question for future work.

There are also challenges concerning the symbolic model. A limitation of our definition is that we cannot handle protocols where there are no public credentials associated to ballots on the bulletin board. There are some examples that are excluded as a consequence, most notably JCJ/Civitas [39], [40]. We need public credentials in order to make an injective correspondence between tallied ballots and earlier events in the trace. A way forward could be injective correspondence assertions, which are supported by ProVerif. Tamarin does not directly provide this feature and would require an encoding. A more general notion of ballot opening is also needed to handle systems with everlasting privacy, where one cannot directly extract votes from ballots [41], [42].

Acknowledgement

This work was supported by the Luxembourg National Research Fund (FNR) and the Research Council of Norway for the joint INTER project SURCVS (No. 11747298).

References

[1] J. Benaloh, "Simple verifiable elections," in *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*, D. S. Wallach and R. L. Rivest, Eds. USENIX Association, 2006. [Online]. Available: https://www.usenix.org/conference/evt-06/simple-verifiable-elections

[2] J. D. C. Benaloh, "Verifiable secret-ballot elections," Ph.D. dissertation, Yale University, New Haven, CT, USA, 1987, aAI8809191.

[3] B. Adida and C. A. Neff, "Ballot casting assurance," in *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06*, 2006. [Online]. Available: https://www.usenix.org/conference/evt-06/ballot-casting-assurance

[4] D. Chaum, P. Y. A. Ryan, and S. A. Schneider, "A practical voter-verifiable election scheme," in *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, 2005, pp. 118–139. [Online]. Available: https://doi.org/10.1007/11555827_8

[5] V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung, "SoK: Verifiability notions for e-voting protocols," in *Proceedings of the 37th IEEE Symposium on Security and Privacy, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 779–798. [Online]. Available: https://doi.org/10.1109/SP.2016.52

[6] R. Küsters, T. Truderung, and A. Vogt, "Clash attacks on the verifiability of e-voting systems," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012, pp. 395–409. [Online]. Available: https://doi.org/10.1109/SP.2012.32

[7] O. Pereira and D. S. Wallach, "Clash attacks and the STAR-Vote system," in *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, ser. Lecture Notes in Computer Science, R. Krimmer, M. Volkamer, N. B. Binder, N. Kersting, O. Pereira, and C. Schürmann, Eds., vol. 10615. Springer, 2017, pp. 228–247. [Online]. Available: https://doi.org/10.1007/978-3-319-68687-5_14

[8] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security analysis of the Estonian Internet voting system," in *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, G. Ahn, M. Yung, and N. Li, Eds. ACM, 2014, pp. 703–715. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660315

[9] J. A. Halderman and V. Teague, "The New South Wales iVote system: Security failures and verification flaws in a live online election," in *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*, ser. Lecture Notes in Computer Science, R. Haenni, R. E. Koenig, and D. Wikström, Eds., vol. 9269. Springer, 2015, pp. 35–53. [Online]. Available: https://doi.org/10.1007/978-3-319-22270-7_3

[10] C. Culnane, A. Essex, S. J. Lewis, O. Pereira, and V. Teague, "Knights and knaves run elections: Internet voting and undetectable electoral fraud," *IEEE Secur. Priv.*, vol. 17, no. 4, pp. 62–70, 2019. [Online]. Available: https://doi.org/10.1109/MSEC.2019.2915398

[11] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. ACM, 2010, pp. 526–535. [Online]. Available: https://doi.org/10.1145/1866307.1866366

[12] A. Kiayias, T. Zacharias, and B. Zhang, "End-to-end verifiable elections in the standard model," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, 2015, pp. 468–498. [Online]. Available: https://doi.org/10.1007/978-3-662-46803-6_16

[13] S. Kremer, M. Ryan, and B. Smyth, "Election verifiability in electronic voting protocols," in *15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010*, ser. Lecture Notes in Computer Science, vol. 6345. Springer, 2010, pp. 389–404. [Online]. Available: https://doi.org/10.1007/978-3-642-15497-3_24

[14] V. Cortier, F. Eigner, S. Kremer, M. Maffei, and C. Wiedling, "Type-based verification of electronic voting protocols," in *4th International Conference on Principles of Security and Trust, London, UK, April 11-18, 2015*, ser. Lecture Notes in Computer Science, vol. 9036. Springer, 2015, pp. 303–323. [Online]. Available: https://doi.org/10.1007/978-3-662-46666-7_16

[15] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "SoK: Computer-Aided Cryptography," in *42nd IEEE Symposium on Security and Privacy*, 2021, to appear.

[16] K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 451–466. [Online]. Available: https://doi.org/10.1109/EuroSP.2017.27

[17] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 435–450. [Online]. Available: https://doi.org/10.1109/EuroSP.2017.38

[18] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and ProVerif," *Foundations and Trends in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[19] D. A. Basin, C. Cremers, J. Dreier, and R. Sasse, "Symbolically analyzing security protocols using Tamarin," *ACM SIGLOG News*, vol. 4, no. 4, pp. 19–30, 2017. [Online]. Available: https://dl.acm.org/citation.cfm?id=3157835

[20] M. Abadi and P. Rogaway, "Reconciling two views of cryptography (the computational soundness of formal encryption)," *J. Cryptology*, vol. 20, no. 3, p. 395, 2007. [Online]. Available: https://doi.org/10.1007/s00145-007-0203-0

[21] V. Cortier, A. Filipiak, and J. Lallemand, "BeleniosVS: Secrecy and verifiability against a corrupted voting device," in *32nd IEEE Computer Security Foundations Symposium, Hoboken, NJ, USA, June 25-28, 2019*, 2019, pp. 367–381. [Online]. Available: https://doi.org/10.1109/CSF.2019.00032

[22] B. Adida, "Helios: Web-based open-audit voting," in *17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, 2008, pp. 335–348. [Online]. Available: http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf

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

[24] "Helios - Verifiable online elections," https://heliosvoting.org/. [Online]. Available: https://heliosvoting.org/

[25] "Belenios - Verifiable online voting system," https://belenios.org/. [Online]. Available: https://belenios.org/

[26] V. Cortier, P. Gaudry, and S. Glondu, "Belenios: A simple private and verifiable electronic voting system," in *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*, ser. Lecture Notes in Computer Science, vol. 11565. Springer, 2019, pp. 214–238. [Online]. Available: https://doi.org/10.1007/978-3-030-19052-1_14

[27] V. Cortier, C. C. Drăgan, F. Dupressoir, and B. Warinschi, "Machine-checked proofs for electronic voting: Privacy and verifiability for Belenios," in *31st IEEE Computer Security Foundations Symposium, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 298–312. [Online]. Available: https://doi.org/10.1109/CSF.2018.00029

[28] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Springer, 2011, pp. 71–90. [Online]. Available: https://doi.org/10.1007/978-3-642-22792-9_5

[29] S. Delaune, S. Kremer, and M. Ryan, "Coercion-resistance and receipt-freeness in electronic voting," in *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*. IEEE Computer Society, 2006, pp. 28–42. [Online]. Available: https://doi.org/10.1109/CSFW.2006.8

[30] L. Hirschi, L. Schmid, and D. A. Basin, "Fixing the achilles heel of e-voting: The bulletin board," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 109, 2020. [Online]. Available: https://eprint.iacr.org/2020/109

[31] K. Sako and J. Kilian, "Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth," in *Advances*

in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 21-25, 1995, Proceeding, ser. Lecture Notes in Computer Science, vol. 921. Springer, 1995, pp. 393–403. [Online]. Available: https://doi.org/10.1007/3-540-49264-X_32

[32] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *CCS 2001, ACM Conference on Computer and Communications Security*, 2001, pp. 116–125. [Online]. Available: http://doi.acm.org/10.1145/501983.502000

[33] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, 2002, pp. 339–353. [Online]. Available: http://www.usenix.org/publications/library/proceedings/sec02/jakobsson.html

[34] M. Hirt and K. Sako, "Efficient receipt-free voting based on homomorphic encryption," in *Advances in Cryptology - EUROCRYPT 2000*. Springer, 2000, pp. 539–556.

[35] T. P. Pedersen, "A threshold cryptosystem without a trusted party (extended abstract)," in *Advances in Cryptology - EUROCRYPT '91*, 1991, pp. 522–526. [Online]. Available: https://doi.org/10.1007/3-540-46416-6_47

[36] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène, "Distributed ElGamal à la Pedersen: Application to Helios," in *ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*. ACM, 2013, pp. 131–142. [Online]. Available: http://doi.acm.org/10.1145/2517840.2517852

[37] P. Y. A. Ryan, P. B. Rønne, and V. Iovino, "Selene: Voting with transparent verifiability and coercion-mitigation," in *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 9604. Springer, 2016, pp. 176–192. [Online]. Available: https://doi.org/10.1007/978-3-662-53357-4_12

[38] K. Gjøsteen, "The Norwegian Internet voting protocol," in *E-Voting and Identity - VoteID 2011*, ser. Lecture Notes in Computer Science, vol. 7187. Springer, 2011, pp. 1–18. [Online]. Available: https://doi.org/10.1007/978-3-642-32747-6_1

[39] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, 2005, pp. 61–70. [Online]. Available: http://doi.acm.org/10.1145/1102199.1102213

[40] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," in *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, 2008, pp. 354–368. [Online]. Available: https://doi.org/10.1109/SP.2008.32

[41] T. Moran and M. Naor, "Receipt-free universally-verifiable voting with everlasting privacy," in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, ser. Lecture Notes in Computer Science, C. Dwork, Ed., vol. 4117. Springer, 2006, pp. 373–392. [Online]. Available: https://doi.org/10.1007/11818175_22

[42] E. Cuvelier, O. Pereira, and T. Peters, "Election verifiability or ballot privacy: Do we need to choose?" in *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, ser. Lecture Notes in Computer Science, vol. 8134. Springer, 2013, pp. 481–498. [Online]. Available: https://doi.org/10.1007/978-3-642-40203-6_27

[43] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The Tamarin prover for the symbolic analysis of security protocols," in *25th International Conference on Computer Aided Verification, Saint Petersburg, Russia, July 13-19, 2013*, ser. Lecture Notes in Computer Science, vol. 8044. Springer, 2013, pp. 696–701. [Online]. Available: https://doi.org/10.1007/978-3-642-39799-8_48

[44] B. Schmidt, S. Meier, C. J. F. Cremers, and D. A. Basin, "Automated analysis of Diffie-Hellman protocols and advanced security properties," in *25th IEEE Computer Security Foundations Symposium, (CSF'12)*. IEEE Computer Society, 2012, pp. 78–94.

[45] D. A. Basin, S. Radomirovic, and L. Schmid, "Dispute resolution in voting," in *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*. IEEE, 2020, pp. 1–16. [Online]. Available: https://doi.org/10.1109/CSF49147.2020.00009

[46] *Additional material: Tamarin codes for verification*, https://github.com/sbaloglu/tamarin-codes.

APPENDIX

A. Proof of Theorem 1

Lemma 1. *For any trace τ and terms cr, v such that $\tau \vDash \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Psi_{\mathsf{tally1}}$ and $\tau \vDash \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \Omega(\mathsf{id},\mathsf{cr},\mathsf{v})$, we have $\tau \vDash \mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b})$, for some term b.*

Proof. From $\tau \vDash \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v})$ and $\tau \vDash \Phi_{\mathsf{eli}}$, we deduce $\tau \vDash \mathsf{BBtally}(\mathsf{cr},\mathsf{b})$, for some term b. Putting this together with $\tau \vDash \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \Omega(\mathsf{id},\mathsf{cr},\mathsf{v})$ and applying $\tau \vDash \Phi_{\mathsf{iv}}$, we deduce $\tau \vDash \mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b})$ as required. □

Theorem 1. *For every trace τ and $\diamond \in \{\circ, \bullet\}$, we have $\tau \vDash \Phi^{\diamond}_{\mathsf{E2E}} \wedge \Psi_{\mathsf{E2E}} \Longrightarrow \tau \Vdash \mathsf{MS}^{\diamond}_{\mathsf{E2E}}$, where*

$$\begin{array}{|lcl|}\hline \Phi^{\diamond}_{\mathsf{E2E}} & = & \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\diamond}_{\mathsf{res}}, \\ \Psi_{\mathsf{E2E}} & = & \Psi_{\mathsf{tally1}} \wedge \Psi_{\mathsf{tally2}}. \\ \hline \end{array}$$

Proof. Assume $\tau \vDash \Psi_{\mathsf{E2E}} \wedge \Phi^{\diamond}_{\mathsf{E2E}}$. Let $R = \{\!|(\mathsf{cr},\mathsf{v}) \mid \tau \vDash \mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \wedge \mathsf{b} \neq \bot \wedge \mathsf{v} = \mathsf{open}(\mathsf{b})|\!\}$. Note that, by definition, $\mathsf{Result}(\tau) = \{\!|\mathsf{v} \mid (\mathsf{cr},\mathsf{v}) \in R|\!\}$. Let us define:

$$\begin{aligned} R_1 &= R \cap \mathsf{Ver}(\tau), \\ R_2 &= R'_2 \setminus \mathsf{Ver}(\tau), \\ R_3 &= R'_3 \setminus \mathsf{Ver}(\tau), \\ R'_2 &= R \cap \{\!|(\mathsf{cr},\mathsf{v}) \mid \exists \mathsf{id}.\ \tau \vDash \mathsf{Vote}(\mathsf{id},\mathsf{cr},\mathsf{v}),\ \mathsf{cr} \notin \mathsf{Adv}^{\diamond}(\tau)|\!\}, \\ R'_3 &= R \cap \{\!|(\mathsf{cr},\mathsf{v}) \mid \mathsf{cr} \in \mathsf{Adv}^{\diamond}(\tau)|\!\}. \end{aligned}$$

For $i \in \{1,2,3\}$, let $V_i = \{\!|\mathsf{v} \mid \exists \mathsf{cr}.\ (\mathsf{cr},\mathsf{v}) \in R_i|\!\}$. We show that V_1, V_2, V_3 satisfy the requirements of Definition 4. First, let us show that $\mathsf{Result}(\tau) = V_1 \uplus V_2 \uplus V_3$. By definition, this is equivalent to showing $R = R_1 \uplus R_2 \uplus R_3$. From $\tau \vDash \Phi^{\diamond}_{\mathsf{res}}$, we deduce that, for each $(\mathsf{cr},\mathsf{v}) \in R$, we have one of two cases: (i) either $\exists \mathsf{id}.\ \tau \vDash \mathsf{Vote}(\mathsf{id},\mathsf{cr},\mathsf{v})$; (ii) or $\tau \vDash \Phi^{\diamond}_{\mathsf{adv}}(\mathsf{cr})$. Therefore, we have $(\mathsf{cr},\mathsf{v}) \in R'_2$ or $(\mathsf{cr},\mathsf{v}) \in R'_3$. Since $R'_2 \cap R'_3 = \emptyset$ and $R'_2, R'_3 \subseteq R$, we can then deduce $R = R'_2 \uplus R'_3$. Next, by definition of R_2, R_3, we have $R'_2 = R_2 \uplus (R'_2 \cap \mathsf{Ver}(\tau))$ and $R'_3 = R_3 \uplus (R'_3 \cap \mathsf{Ver}(\tau))$. Moreover, we have:

$$\begin{aligned} R_1 &= R \cap \mathsf{Ver}(\tau) \\ &= (R'_2 \uplus R'_3) \cap \mathsf{Ver}(\tau) \\ &= (R'_2 \cap \mathsf{Ver}(\tau)) \uplus (R'_3 \cap \mathsf{Ver}(\tau)). \end{aligned}$$

Therefore, we can conclude $R = R_1 \uplus R_2 \uplus R_3$ and $\mathsf{Result}(\tau) = V_1 \uplus V_2 \uplus V_3$.

Next, we show that each of V_1, V_2, V_3 satisfies respectively 1), 2), 3) from Definition 4:

1) We show $V_1 = \mathsf{Ver}_{\mathsf{vote}}(\tau)$. By definition, we have $V_1 \subseteq \mathsf{Ver}_{\mathsf{vote}}(\tau)$. Let us show that $\mathsf{Ver}_{\mathsf{vote}}(\tau) \subseteq V_1$. For any $\mathsf{v} \in \mathsf{Ver}_{\mathsf{vote}}(\tau)$, let s be the number of times v occurs in $\mathsf{Ver}_{\mathsf{vote}}(\tau)$. We show that the number of times v occurs in the multiset V_1 is at least s. Consider the list of all $(\mathsf{id}_1,\mathsf{cr}_1),\ldots,(\mathsf{id}_{\mathsf{s}},\mathsf{cr}_{\mathsf{s}})$ such that $\forall \mathsf{i} \in \{1,\ldots,\mathsf{s}\}$, $\tau \vDash \mathsf{Verified}(\mathsf{id}_{\mathsf{i}},\mathsf{cr}_{\mathsf{i}},\mathsf{v})$. By definition of $\mathsf{Ver}(\tau)$ and by the definition of s, for any $\mathsf{i} \neq \mathsf{j}$, we have $(\mathsf{id}_{\mathsf{i}},\mathsf{cr}_{\mathsf{i}}) \neq (\mathsf{id}_{\mathsf{j}},\mathsf{cr}_{\mathsf{j}})$. Therefore, if $\mathsf{id}_{\mathsf{i}} = \mathsf{id}_{\mathsf{j}}$, we must have $\mathsf{cr}_{\mathsf{i}} \neq \mathsf{cr}_{\mathsf{j}}$. Moreover, from $\tau \vDash \Phi_{\mathsf{cl}}$, we also have $\mathsf{id}_{\mathsf{i}} \neq \mathsf{id}_{\mathsf{j}} \Rightarrow \mathsf{cr}_{\mathsf{i}} \neq \mathsf{cr}_{\mathsf{j}}$. Thus, we can conclude that, for any $\mathsf{i} \neq \mathsf{j}$, we have $\mathsf{cr}_{\mathsf{i}} \neq \mathsf{cr}_{\mathsf{j}}$.

For every $\mathsf{i} \in \{1, \ldots, \mathsf{s}\}$, from $\tau \vDash \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Psi_{\mathsf{tally1}}$ and $(\mathsf{cr_i}, \mathsf{v}) \in \mathsf{Ver}(\tau)$, by Lemma 1, we deduce $\tau \vDash \mathsf{BBtally}(\mathsf{cr_i}, \mathsf{b_i}) \wedge \mathsf{v} = \mathsf{open}(\mathsf{b_i})$. Therefore, by definition of R_1, we have $\{\!|(\mathsf{cr}_1, \mathsf{v}), \ldots, (\mathsf{cr_s}, \mathsf{v})|\!\} \subseteq R_1$. Moreover, from $\mathsf{i} \neq \mathsf{j} \Rightarrow \mathsf{cr_i} \neq \mathsf{cr_j}$, we can then deduce that all $(\mathsf{cr_i}, \mathsf{v})$ are distinct, and therefore, the multiplicity of v in V_1 is at least s. Thus, we can conclude that $V_1 = \mathsf{Ver}_{\mathsf{vote}}(\tau)$ as required.

2) We show the required property for V_2. Let $R_2 = \{\!|(\mathsf{cr}_1, \mathsf{v}_1), \ldots, (\mathsf{cr_l}, \mathsf{v_l})|\!\}$. From $\tau \vDash \Psi_{\mathsf{tally2}}$, we know that $\mathsf{cr}_1, \ldots, \mathsf{cr_l}$ are mutually distinct. By definition of R_2, for all $\mathsf{i} \in \{1, \ldots, \mathsf{l}\}$, we have $\mathsf{cr_i} \notin \mathsf{Ver}(\tau) \cup \mathsf{Adv}^{\diamond}(\tau)$ and there exists $\mathsf{id_i}$ such that $\tau \vDash \mathsf{Vote}(\mathsf{id_i}, \mathsf{cr_i}, \mathsf{v_i})$.

3) We show $|V_3| < |\ \mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}_{\mathsf{cr}}(\tau)\ |$. By definition, we have $|V_3| = |R_3|$ and

$$R_3 = (R \cap R_3') \setminus \mathsf{Ver}(\tau) \subseteq R \cap (R_3' \setminus \mathsf{Ver}(\tau)).$$

From $\tau \vDash \Psi_{\mathsf{tally2}}$, we have for all cr, $|\{\!|\mathsf{v} \mid (\mathsf{cr}, \mathsf{v}) \in R_3|\!\}| \leq |\{\!|\mathsf{v} \mid (\mathsf{cr}, \mathsf{v}) \in R|\!\}| \leq 1$. Therefore, by definition of $\mathsf{Adv}^{\diamond}(\tau)$ and R_3', we can deduce $|R_3| < |\ \mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}_{\mathsf{cr}}(\tau)\ |$ and conclude $|V_3| < |\ \mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}_{\mathsf{cr}}(\tau)\ |$. □

B. Helios and Belenios specification details

The details for the Helios specification are in Figure 4 and for the Belenios specification are in Figure 5 on the following pages.

Fig. 4: Protocol components for Helios specification.

(a) Protocol specification $\mathcal{P}_{\mathcal{H}} = (\mathcal{R}_{\mathcal{H}}, \Psi_{\mathcal{H}})$.

Rules are labelled with corresponding protocol parties A, T, V, VR, VS, VP, EA as introduced in Section IV.

SETUP PHASE

$\mathsf{R}^{\mathsf{T}}_{\mathsf{key}}$: **generate election secret and public keys**
$[\ \mathsf{Fr}(\mathsf{sk})\ ] \dashv\{\ \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}))\ \} \mapsto$
$[\ \mathsf{Sk}(\mathsf{sk}), \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk})), \mathsf{Out}(\mathsf{pk}(\mathsf{sk}))\ ]$

$\mathsf{R}^{\mathsf{A}}_{\mathsf{cand}}$: **determine candidates to be elected**
$\mathsf{let}\ \ \mathsf{vlist} = \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle\ \ \mathsf{in}$
$[\ \mathsf{In}(\mathsf{vlist})\ ] \dashv\{\ \mathsf{BBcand}(\mathsf{v}_1), \ldots, \mathsf{BBcand}(\mathsf{v}_k), \mathsf{Vlist}(\mathsf{vlist})\ \} \mapsto$
$[\ \mathsf{BBcand}(\mathsf{v}_1), \ldots, \mathsf{BBcand}(\mathsf{v}_k), \mathsf{Vlist}(\mathsf{vlist})\ ]$

$\mathsf{R}^{\mathsf{A}}_{\mathsf{id}}$: **determine identities eligible to vote**
$[\ \mathsf{In}(\mathsf{id})\ ] \dashv\{\ \} \mapsto [\ \mathsf{Id}(\mathsf{id})\ ]$

$\mathsf{R}^{\mathsf{VR/VS}}_{\mathsf{reg}}$: **register voter with credential and password**
$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{cr}), \mathsf{Fr}(\mathsf{pwd})\ ] \dashv\{\ \mathsf{BBreg}(\mathsf{cr})\ \} \mapsto$
$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Out}(\mathsf{cr})\ ]$

$\mathsf{R}^{\mathsf{VS}}_{\mathsf{bb}}$: **setup initial** BBcast **for registered voters**
$[\ \mathsf{BBreg}(\mathsf{cr})\ ] \dashv\{\ \mathsf{BBcast}(\mathsf{cr}, \bot)\ \} \mapsto [\ \mathsf{BBcast}(\mathsf{cr}, \bot)\ ]$

VOTING PHASE

$\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$: **construct a ballot, authenticate and send it to** VR/VS
$\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r});\ \ \mathsf{pr}_1 = \mathsf{proof}_1(\mathsf{c}, \mathsf{r}, \mathsf{vlist});$
$\mathsf{b} = \langle \mathsf{c}, \mathsf{pr}_1 \rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle)\ \ \mathsf{in}$
$[\ \mathsf{BBcand}(\mathsf{v}), \mathsf{BBkey}(\mathsf{pkey}), \mathsf{Fr}(\mathsf{r}), \mathsf{Vlist}(\mathsf{vlist}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}),$
$\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ] \dashv\{\ \mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ \} \mapsto$
$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle)\ ]$

$\mathsf{R}^{\mathsf{VR/VS}}_{\mathsf{cast}}$: **authenticate voter, verify and publish ballot**
$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{pr}_1 \rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle)\ \ \mathsf{in}$
$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{BBkey}(\mathsf{pkey}), \mathsf{Vlist}(\mathsf{vlist}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}),$
$\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ] \dashv\{\ \mathsf{a}' = \mathsf{a}, \mathsf{ver}_1(\mathsf{pr}_1, \mathsf{c}, \mathsf{pkey}, \mathsf{vlist}) = \mathsf{ok},$
$\mathsf{VScast}(\mathsf{id}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ \} \mapsto [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$

TALLY PHASE

$\mathsf{R}^{\mathsf{VS/EA}}_{\mathsf{tally}}$: VS **selects ballots for tally; can be audited by** EA
$[\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ] \dashv\{\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ \} \mapsto [\ \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$

$\Psi^{\mathsf{VS/EA}}_{\mathsf{cast}}$: **ensure ballot validity; can be audited by** EA
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \Rightarrow \mathsf{BBreg}(\mathsf{cr}) \ \wedge\ (\ \mathsf{b} \neq \bot \Rightarrow \mathsf{b} = \langle \mathsf{c}, \mathsf{pr}_1 \rangle \ \wedge$
$\mathsf{BBkey}(\mathsf{pkey}) \ \wedge\ \mathsf{Vlist}(\mathsf{vlist}) \ \wedge\ \mathsf{ver}_1(\mathsf{pr}_1, \mathsf{c}, \mathsf{pkey}, \mathsf{vlist}) = \mathsf{ok}\ )$

$\Psi^{\mathsf{VS/EA}}_{\mathsf{tally}}$: **the last ballot added to BB is selected for tally**
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i \ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \ \wedge$
$\mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ @l \Rightarrow j \prec i \ \vee\ \mathsf{b} = \mathsf{b}'$

(b) Individual verification procedures for $\mathcal{P}_{\mathcal{H}}$.

$\mathsf{R}^{0}_{\mathsf{ver}}$: **voter verifies the ballot on** BBcast
$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$
$\dashv\{\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ \} \mapsto [\ \ ]$

$\mathsf{R}^{1}_{\mathsf{ver}}$: **voter verifies the ballot on** BBtally
$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$
$\dashv\{\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})\ \} \mapsto [\ \ ]$

$\mathsf{R}^{2}_{\mathsf{ver}}$: **voter verifies there is no ballot on** BBtally
$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{BBtally}(\mathsf{cr}, \bot)\ ]$
$\dashv\{\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \bot)\ \} \mapsto [\ \ ]$

$\mathsf{R}^{0}_{\mathsf{ver}}$ **can be combined with restrictions below:**

Ψ_{last} : **the verified ballot is currently the last on BB**
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @i \ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \ \wedge$
$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @l \ \wedge\ i \prec l \ \wedge\ j \prec l \Rightarrow j \prec i \ \vee\ \mathsf{b} = \mathsf{b}'$

Ψ_{mine} : **all ballots currently on BB were cast by** id
$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i \ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @j \ \wedge\ j \prec i$
$\Rightarrow \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @l$

(c) Adversarial corruption rules against $\mathcal{P}_{\mathcal{H}}$.

$\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}$: **corrupt trustee to control the secret key**
$[\ \mathsf{In}(\mathsf{sk})\ ] \dashv\{\ \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}))\ \} \mapsto [\ \mathsf{Sk}(\mathsf{sk}), \mathsf{BBkey}(\mathsf{pk}(\mathsf{sk}))\ ]$

$\mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}$: **corrupt voter to reveal credentials**
$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})\ ] \dashv\{\ \mathsf{Corr}(\mathsf{id}, \mathsf{cr})\ \} \mapsto$
$[\ \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{pwd} \rangle)\ ]$

$\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$: **corrupt server to stuff ballots**
$[\ \mathsf{In}(\langle \mathsf{cr}, \mathsf{b} \rangle)\ ] \dashv\{\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ \} \mapsto [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$

$\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}$: **corrupt registration of public credential**
$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{cr}' \rangle), \mathsf{Fr}(\mathsf{pwd})\ ] \dashv\{\ \mathsf{BBreg}(\mathsf{cr}')\ \} \mapsto$
$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}), \mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBreg}(\mathsf{cr}')\ ]$

$\mathcal{C}^{\mathsf{VP}}_{\mathsf{vote}}$: **corrupt platform to choose randomness**
rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ where $\mathsf{Fr}(\mathsf{r})$ is replaced by $\mathsf{In}(\mathsf{r})$

Ψ_{order} : **ensure ballots are delivered in the right order**
$\mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @i \ \wedge\ \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @j \ \wedge$
$\mathsf{VScast}(\mathsf{id}, \mathsf{b})\ @k \ \wedge\ \mathsf{VScast}(\mathsf{id}, \mathsf{b}')\ @l \ \wedge\ i \prec j \Rightarrow k \prec l$

Fig. 5: Protocol components for Belenios specification.

(a) Protocol specification $\mathcal{P}_{\mathcal{B}} = (\mathcal{R}_{\mathcal{B}}, \Psi_{\mathcal{B}})$.

SETUP PHASE

R^{T}_{key} : **generate election secret and public keys**
[Fr(sk)]⊣{ BBkey(pk(sk)) }↦
[Sk(sk), BBkey(pk(sk)), Out(pk(sk))]

R^{A}_{cand} : **determine candidates to be elected**
let vlist = $\langle v_1, \ldots, v_k \rangle$ in
[In(vlist)]⊣{ BBcand(v_1), … , BBcand(v_k), Vlist(vlist) }↦
[BBcand(v_1), … , BBcand(v_k), Vlist(vlist)]

R^{A}_{id} : **determine identities eligible to vote**
[In(id)]⊣{ }↦[Id(id)]

R^{VR}_{reg} : **register voter with signature pair**
let cr = pk(skey) in
[Id(id), Fr(skey)]⊣{ BBreg(cr) }↦
[Reg(id, cr, skey), BBreg(cr), Out(cr)]

R^{VS}_{pwd} : **generate password for voter authentication**
[Id(id), Fr(pwd)]⊣{ }↦[Pwd(id, pwd)]

R^{VS}_{bb} : **setup initial** BBcast **for registered voters**
[BBreg(cr)]⊣{ BBcast(cr, ⊥) }↦[BBcast(cr, ⊥)]

VOTING PHASE

R^{VP}_{vote} : **construct a ballot, authenticate and send it to** VS
let c = enc(v, pkey, r); s = sign(c, skey);
$pr_1 = proof_1(c, r, vlist)$; $pr_2 = proof_2(c, r, cr)$;
b = ⟨c, s, pr_1, pr_2⟩; a = h(⟨id, pwd, cr, b⟩) in
[BBcand(v), BBkey(pkey), Fr(r), Vlist(vlist), Reg(id, cr, skey)
Pwd(id, pwd)]⊣{ Vote(id, cr, v), VoteB(id, cr, b) }↦
[Voted(id, cr, v, b), Out(⟨id, cr, b, a⟩)]

R^{VS}_{cast} : **authenticate voter, verify and publish ballot**
let b = ⟨c, s, pr_1, pr_2⟩; a′ = h(⟨id, pwd, cr, b⟩) in
[In(⟨id, cr, b, a⟩), BBkey(pkey), Vlist(vlist), BBreg(cr),
Pwd(id, pwd)] ⊣{ a′ = a, ver(s, c, cr) = ok,
$ver_1(pr_1, c, pkey, vlist) = ok$, $ver_2(pr_2, c, cr) = ok$,
Log(id, cr), VScast(id, b), BBcast(cr, b) }↦ [BBcast(cr, b)]

TALLY PHASE

$R^{VS/EA}_{tally}$: VS **selects ballots for tally; can be audited by** EA
[BBcast(cr, b)]⊣{ BBtally(cr, b) }↦[BBtally(cr, b)]

Ψ^{VS}_{log} : **logs are checked to ensure consistency**
Log(id, cr) @i ⇒ ¬(Log(id, cr′) @j ∧ cr ≠ cr′) ∧
¬(Log(id′, cr) @j ∧ id ≠ id′)

$\Psi^{VS/EA}_{cast}$: **ensure ballot validity; can be audited by** EA
BBcast(cr, b) ⇒ BBreg(cr) ∧ (b ≠ ⊥ ⇒ b = ⟨c, s, pr_1, pr_2⟩
∧ BBkey(pkey) ∧ Vlist(vlist) ∧ ver(s, c, cr) = ok
∧ $ver_1(pr_1, c, pkey, vlist) = ok$ ∧ $ver_2(pr_2, c, cr) = ok$)

$\Psi^{VS/EA}_{tally}$: **the last ballot added to BB is selected for tally**
BBcast(cr, b) @i ∧ BBcast(cr, b′) @j ∧
BBtally(cr, b) @l ⇒ $j \prec i$ ∨ b = b′

(b) Individual verification procedures for $\mathcal{P}_{\mathcal{B}}$.

R^{0}_{ver} : **voter verifies the receipt on** BBcast
[Voted(id, cr, v, b), BBcast(cr, b)]
⊣{ Verified(id, cr, v), VerB(id, cr, b) }↦[ ]

R^{1}_{ver} : **voter verifies the receipt on** BBtally
[Voted(id, cr, v, b), BBtally(cr, b)]
⊣{ Verified(id, cr, v) }↦[ ]

R^{2}_{ver} : **voter verifies there is no ballot on** BBtally
[Reg(id, cr, skey), BBtally(cr, ⊥)]
⊣{ Verified(id, cr, ⊥) }↦[ ]

R^{0}_{ver} **can be combined with restrictions below:**

Ψ_{last} : **the verified ballot is currently the last on BB**
BBcast(cr, b) @i ∧ BBcast(cr, b′) @j ∧
VerB(id, cr, b) @l ∧ $i \prec l$ ∧ $j \prec l$ ⇒ $j \prec i$ ∨ b = b′

Ψ_{mine} : **all ballots currently on BB are cast by** id
VerB(id, cr, b) @i ∧ BBcast(cr, b′) @j ∧ $j \prec i$
⇒ VoteB(id, cr, b′) @l

(c) Adversarial corruption rules against $\mathcal{P}_{\mathcal{B}}$.

C^{T}_{key} : **corrupt trustee to control the secret key**
[In(sk)]⊣{ BBkey(pk(sk)) }↦[Sk(sk), BBkey(pk(sk))]

C^{V}_{corr} : **corrupt voter to reveal credentials**
[Reg(id, cr, skey), Pwd(id, pwd)]
⊣{ Corr(id, cr) }↦[Out(⟨id, cr, skey, pwd⟩)]

C^{VS}_{cast} : **corrupt server to stuff ballots**
[In(⟨cr, b⟩)]⊣{ BBcast(cr, b) }↦[BBcast(cr, b)]

C^{VR}_{reg} : **corrupt registration of public / secret credentials**
[In(⟨id, cr, skey, cr′⟩)]⊣{ BBreg(cr′) }↦
[Reg(id, cr, skey), BBreg(cr′)]

C^{VP}_{vote} : **corrupt platform to choose randomness**
rule R^{VP}_{vote} where Fr(r) is replaced by In(r)

Ψ_{order} : **ensure ballots are delivered in the right order**
VoteB(id, cr, b) @i ∧ VoteB(id, cr, b′) @j ∧
VScast(id, b) @k ∧ VScast(id, b′) @l ∧ $i \prec j$ ⇒ $k \prec l$