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

Sevdenur Baloglu   Sergiu Bursuc   Sjouke Mauw   Jun Pang

firstname.lastname@uni.lu
University of Luxembourg

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 Helios, our analysis is the first one to be, at the same time, fully automated (with the Tamarin protocol prover) and to precisely capture its end-to-end verifiability guarantees, allowing us to derive new security proofs and new attacks on deployed versions of it. For Belenios, similarly, we capture precisely the end-to-end verifiability guarantees when all election authorities are corrupted, which is outside the scope of previous formal definitions. We also find new attacks that apply in weaker corruption scenarios that are expected to be secure. In general, our framework allows a unified analysis and comparison of cryptographic voting protocols, corruption scenarios and verifiability procedures towards ensuring the end goal of election integrity.

1 Introduction

Considering the importance of elections in modern societies, it is crucial to ensure the security of voting systems underlying them. More and more, these systems become electronic, bringing numerous benefits, but also increasing the scope for attacks. Indeed, concrete attacks are always being discovered by researchers on deployed electronic voting systems. See for example [62, 61, 59, 41, 35], showing that even recent systems, prepared by governments to run real-world elections, are vulnerable to attacks, in spite of the fact that secure electronic voting is a decades-old problem [39, 24, 17]. A crucial property that is the target of attacks is *election integrity*, where we need to ensure that the choice of each eligible voter is correctly reflected in the final outcome, and that no additional votes can be introduced by attackers.

In order to better protect election integrity, various notions like software independence, ballot casting assurance, voter-verifiable and universally-verifiable elections have been proposed [55, 17, 19, 18, 8, 25, 51, 44, 21]. The main idea is that the voting system should output sufficient information such that election integrity can be anchored in two main assumptions: (i) voters perform basic verification steps to ensure that their ballots are correctly recorded by a public bulletin board (*individual verifiability*); (ii) by running certain audit procedures on the bulletin board, any party can ensure that the published outcome correctly represents the recorded ballots (*universal verifiability*). Security relies on the fact that individual and universal verifiability procedures are public, transparent and can be performed on any platform, independent of the platforms that run the election, or of any particular software implementation. The underlying cryptographic protocol should ensure that, putting together individual and universal verifiability checks, one obtains a strong notion of *end-to-end verifiability*, guaranteeing a global property of election integrity. However, as shown by Küsters et al [48, 49, 47], these local verifiability properties do not always imply a global notion of verifiability, and this gap may be exploited by corrupted parties to compromise election integrity. For example, for Helios [5, 6, 7], a prominent internet voting system used in real-world elections with significant stakes, [49] shows that, relying on so-called clash attacks, an adversary corrupting some election authorities and the voting platforms can replace votes from eligible voters with arbitrary votes without being detected by individual and universal verifiability procedures. Clash attacks are not specific to Helios and have been shown on a series of systems [49, 54].

These attacks do not invalidate the general idea of deriving election integrity from a minimal set of trusted software components and cryptographic verification procedures. Rather, they show the need for usable, systematic and rigorous methods to evaluate if proposed protocols and notions indeed achieve the stated goal of end-to-end verifiability. There are two main aspects to consider, as well as their interplay:

- *protocol security:* given a specification of an e-voting protocol - vote encoding, ballot casting, tally computation, verification procedures, etc - does it formally follow that no adversary from a given class (defined by computational and corruption abilities) can compromise election integrity?
- *software security:* are the verification procedures that are prescribed by the protocol securely implemented?

The attacks from above can be mapped to one of these two classes (against the protocol, or against the software), but sometimes they exploit implicit assumptions in the protocol that are not ensured by the corresponding software implementation, e.g. [35]. Therefore we need a precise specification language, with an appropriate level of abstraction, that can be shared by protocol designers and developers to facilitate complete protocol specifications and minimize inconsistencies between design and implementation.

There are two well-established methods for proving protocol security: computational [23, 16, 13] and symbolic [2, 38, 3] - and soundness results connecting the two [4, 34, 14]. The computational approach captures fine-grained details of the underlying cryptographic primitives, while the symbolic approach assumes perfect cryptography - represented by a set of abstract equations and processes - and aims for automated verification. Prominent automated tools for symbolic verification are for example ProVerif [22] and Tamarin [50]. For election verifiability, there are several computational models that allow precise modelling of the desired properties [31, 30, 26, 47, 19, 45]. The focus of our paper are symbolic models and automated verification. One defining feature of this setting is that, to be handled by automated tools like ProVerif and Tamarin, the security properties need to be expressed as a set of logical properties relating events along the execution trace of a voting system. The main challenge is to identify a set of properties that is at the same time general (allowing to capture a wide range of protocols and corruption scenarios) and sound (it entails a multiset-based notion of end-to-end verifiability).

Symbolic election verifiability. The symbolic model that comes closest to the above-mentioned goals is the one by Cortier et al [28], designed to prove verifiability for a variant of Belenios [33]. While their model achieves the goal of soundness (end-to-end verifiability), it has several limitations in terms of generality:
– distinct corruption scenarios relating to election authorities (server, registrar) are covered by distinct symbolic verifiability definitions, and there is no formal way to connect these, making it hard to directly adopt the model for analysing new protocols.
– to obtain soundness, their model disallows dynamic corruption of voters and, somehow intuitively, it does not provide end-to-end verifiability guarantees for corrupted voters; however, 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.
– their model does not take into account revoting - which, we will show, can be exploited by adversaries.
– the absence of clash attacks is not formally implied by symbolic verifiability in [28]; instead, it follows from additional restrictions on the execution trace, which vary according to the scenario that is considered (corrupted server or corrupted registrar), and are justified mainly by the fact that one of the corresponding parties is trusted. More generally, these restrictions 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 will show that making it explicitly part of symbolic verifiability requirements can entail end-to-end verifiability, without restrictions specific to various corruption scenarios.

An earlier, type-based symbolic model, applied to Helios, is proposed by Cortier et al in [27]. The associated notion of end-to-end verifiability is weaker than [28]: 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, which also limits the multiset of adversarially cast votes, depending on the corruption abilities of the adversary. We stress that this characterisation should be provided 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 as strong as Belenios, if one makes the assumption that neither the voting server nor the registrar is corrupted (Belenios assumes that one of these two parties is not corrupted). A second reason is that, as we show in the paper, even for Helios where both of these parties are corrupted, we can still achieve a meaningful, complete notion of end-to-end verifiability, which does provide significant guarantees in practice. Concerning clash attacks, [27] considers a *no clash* property ensuring that ballots constructed by distinct honest voters are distinct. This captures only a particular aspect of clash attacks introduced in [49], ignoring the ballot verification part. However, we will show that clash attacks are possible even when ballots for distinct voters are distinct, exploiting the structure of the bulletin board and verification mechanisms when revoting is allowed. These attacks are out of the scope of [27] since, like [28], they do not consider revoting and only one ballot per voter may be cast on the bulletin board.

The first general symbolic model for election verifiability was proposed by Kremer et al [46]. However, the scenarios to which their model applies are restricted to the case when all voters are honest and verify their votes. Moreover, although the protocol models in [46] are symbolic - in the sense of being specified in an abstract process algebra - the formulas used for specifying the security properties are, in some sense, not symbolic enough for automated verification with ProVerif/Tamarin. They are global properties referring to an unbounded number of events in a trace, and are closer to multiset-based end-to-end verifiability. The challenge for symbolic verifiability is, relying on universal quantification over events in a trace, to capture end-to-end verifiability within the standard class of trace formulas accepted by ProVerif/Tamarin.

Our contributions. We propose a general symbolic definition for election verifiability that improves on current definitions as follows:
– 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, as we explain below;

– it implies end-to-end verifiability in a generic framework, without requiring scenario-specific trust assumptions as in [28]. 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 demonstrate the application of our definition for multiple scenarios in Helios and Belenios. For Belenios, in addition to the two paradigmatic scenarios mentions above, our analysis is the first one to cover the case when *both* the server and the registrar are corrupted. For Helios, instantiating our definition with different parameters, we perform a refined analysis considering different (i) versions of the bulletin board; (ii) sets of corrupted components and parties; (iii) individual verification procedures. For both Helios and Belenios, we show new attacks and new security proofs, illustrating some tradeoffs between usability and security. Proofs and attacks are automatically completed with the Tamarin prover.

A particular class of attacks that we study are clash attacks, where the individual verifiability mechanism of the system can be subverted so that ballots and the public credentials of two voters *clash*, leading for one single vote to be cast in their name, and leaving room for adversarial ballot stuffing [49]. We show that against some versions of Helios, including the one currently deployed and used in many elections, a clash attack can be performed by a significantly *weaker adversary* than the one of [49]. In particular, we show that a successful adversary neither has to corrupt the voting platform nor (fully control) the voting server. It is sufficient to corrupt the authority that assigns public credentials and the component that performs voter authentication, and to rely on the way revote ballots are processed and verified by honest parties in the system.

In Belenios, when the voting server is honest, it performs some consistency checks between voter identities and public credentials, in order to ensure, in particular, 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 corrupt voters can choose to submit ballots for any public credential. Together with a corrupt registrar, the adversary can then cast any ballot for a desired public credential.

Paper structure. In section 2 we review Helios, present the clash attack of [49], and our new class of clash attacks. We also propose some mitigations against these attacks. Section 3 contains preliminaries for formal verification of (e-voting) protocols with Tamarin. Section 4 contains our formal framework and definitions for election verifiability, which are applied to Helios in section 5 and to Belenios in section 6.

2 Helios voting system and clash attacks

2.1 Protocol description

Our description of Helios follows [49, 7, 6] and also our observations on the version deployed online [5]. Apart from voters (V), the parties in the protocol are:

- *Administrator (*A*):* determines the list eligible candidates and the list of eligible voters.
- *Trustees (*T*):* generate election secret key, publish the corresponding public key, compute the final outcome.
- *Registrar (*VR*):* assigns to every eligible voter a public credential, also called an alias, which is published on the bulletin board; it may also generate passwords for voter authentication.
- *Voting Server (*VS*):* receives ballots cast by authenticated voters and publishes them on BB.
- *Bulletin Board (*BB*):* public ledger containing election information: the public key, the list of candidates, the list of public credentials for eligible voters, the list of cast ballots, the final outcome and proofs of correctness. We denote by BBx the portion of the bulletin board corresponding to a particular element x. For example, BBcast represents the published list of cast ballots.
- *Voting Platform (*VP*):* constructs ballots for voter choices, authenticates and transmits them to VR/VS.
- *Election Auditors (*EA*):* perform audit and verification of proofs on the bulletin board.

VR and VS may be closely connected, possibly maintained by the same party. The voter authentication task may also be shifted from VR to VS. We note that, while the clash attack of [49] assumes $\{\mathsf{VP}, \mathsf{VR}, \mathsf{VS}\}$ are corrupted, it is sufficient to corrupt $\{\mathsf{VP}, \mathsf{VR}\}$ if VR takes care of both alias generation and authentication as in [49, 7].

Setup phase. A determines the list of eligible voters $\mathsf{id}_1, \ldots, \mathsf{id}_n$, and sends it to VR and VS. They choose corresponding public identities, passwords, and communicate these to voters. 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 and each voter id_i obtains credentials $(\mathsf{id}_i, \mathsf{cr}_i, \mathsf{pwd}_i)$ after the registration. We denote this by $\mathsf{Reg}(\mathsf{id}_i, \mathsf{cr}_i, \mathsf{pwd}_i)$.

Voting phase. In this phase, voters interact with their voting platform VP to construct a ballot, authenticate and post it to VR/VS, which cast it on BB. A ballot is simply an encryption of the vote along with its hash to facilitate tracking, and possibly a zero-knowledge proof if homomorphic tally is used. We have:

$$\begin{array}{lcl} \mathsf{VP} & : & \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r});\ \mathsf{b} = \langle \mathsf{h}(\mathsf{c}), \mathsf{c} \rangle;\ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle), \\ \mathsf{VR}/\mathsf{VS} & : & \text{receive } \langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle \text{ and match it with } (\mathsf{id}, \mathsf{pwd}, \mathsf{cr}), \\ \mathsf{BBcast} & : & (\mathsf{cr}, \mathsf{b}). \end{array}$$

Individual verification procedures. They allow voters to ensure that their ballot correctly encodes their vote and it is correctly cast on BB. A first part of individual verifiability allows voters to audit their VP, leading the randomness in their ballot b to be revealed, which allows to check whether b does indeed encode their desired vote v [18]. In that case, a new ballot needs to be constructed for vote casting. In a second part, after casting their vote, voters can verify BBcast, to check wether b is indeed recorded next to their credential cr [8]. For usability purposes, voters actually check the hash associated to the ballot, that we call a receipt, and by public audits it can be ensured that the hash does indeed match the corresponding ciphertext published on BBcast. We note that, in case revoting is allowed, there are a few deployment options for BBcast and individual verification to consider:

– ***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 [5].

– ***Show all ballots cast.*** For a cr as above, BBcast will show all ballots that were cast by cr. We have:

$$\mathsf{BBcast} : (\mathsf{cr}, \mathsf{b}^1), \ldots, (\mathsf{cr}, \mathsf{b}^m),$$

and it should be clear for the voter which ballot is considered to be the last one cast 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 [5].

– ***Verify after voting ends.*** Voters are instructed to verify the bulletin board only after the voting phase has ended. The VS can publish and update BBcast dynamically during the voting phase, or publish it only when the voting phase has ended.

We show that some of these deployment choices, combined with appropriate individual verification procedures, offer better security for election integrity, e.g. better protection against clash attacks, than the currently deployed options.

Tally phase. The ballots which will be tallied are selected and marked as input for the tally procedure. Selection typically chooses the last valid ballot cast by each cr_i. We have:

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

where $\mathsf{b_i} = \bot$ if no ballot was cast for $\mathsf{cr_i}$. The ciphertexts corresponding to non-empty ballots on $\mathsf{BBtally}$ are extracted for decryption by trustees. Before decryption, in order to protect vote-privacy, based on the homomorphic properties of ElGamal encryption [37, 42], the ciphertexts are combined into a ciphertext c containing the sum of the corresponding votes. In this case, each ballot should also contain a proof that the corresponding ciphertext encodes a valid vote. Alternatively, as described in the original Helios paper [6], decryption can be performed after passing ciphertexts through a verifiable mixnet [56, 60]. In both cases, the trustees (and the mixnet if one is used) publish non-interactive cryptographic proofs that can be checked by external parties to ensure that the claimed outcome indeed corresponds to votes which were encoded to the ballots in $\mathsf{BBtally}$. This is traditionally called *universal verifiability*.

2.2 End-to-end verifiability and clash attacks

Intuitively, the combination of individual verification procedures and universal verifiability as above should ensure *end-to-end verifiability*, i.e. each individual vote is correctly captured in the final outcome. However, as shown in [48, 49], this is not always the case, and a more general, global account of verifiability needs to be considered [47]. Moreover, we need to explicitly state what are the corruption abilities of the adversary, and aim towards a set of minimal assumption regarding the parties or infrastructure that needs to be trusted.

Classic clash attack on Helios. The so-called clash attack of [49] works as follows. Assume VR, VS and VP are corrupted (if VR does both the generation of aliases and voter authentication, then VS does not need to be corrupted). By corrupted VP, we mean the adversary corrupts the platforms of a few honest voters. For illustration, we consider a simple scenario with three eligible voters $\mathsf{id_1}, \mathsf{id_2}, \mathsf{id_3}$, where $\mathsf{id_1}$ and $\mathsf{id_2}$ vote for A, while $\mathsf{id_3}$ votes for B. We show that corrupted parties can convince all other parties that votes are correctly counted, while making the outcome be A, B, B instead of A, A, B.

Since ballots may be audited before being cast, the voting platform VP cannot simply change a vote for A to a vote for B without taking the risk of being detected. Therefore, a constraint for the adversary is to mount this attack while computing correct encryptions of votes on VP. Assume $\mathsf{VR/VS}$ publishes the public identities $\mathsf{cr_1}, \mathsf{cr_2}, \mathsf{cr_3}$ as corresponding to eligible voters. Then, corrupted parties do:

VR	:	$\mathsf{Reg(id_1, cr_1, \_), Reg(id_2, cr_1, \_), Reg(id_3, cr_3, \_)}$, that is, $\mathsf{cr_1}$ is communicated to both $\mathsf{id_1}$ and $\mathsf{id_2}$ as their credential,
VP	:	$\mathsf{c_1} = \mathsf{enc}(A, \mathsf{pk}, \mathsf{r})$ with same r for both $\mathsf{id_1}$ and $\mathsf{id_2}$,
$\mathsf{VR/VS}$	:	cast $\mathsf{b_1} = \langle \mathsf{h(c_1), c_1} \rangle$ as corresponding to $\mathsf{cr_1}$,
$\mathsf{VR/VS}$	:	cast a ballot $\mathsf{b_2}$ for $\mathsf{cr_2}$ encoding a vote for B.

Assuming that $\mathsf{b_1}$ is the ballot provided by VP to both $\mathsf{id_1}$ and $\mathsf{id_2}$, and $\mathsf{b_3}$ is the ballot cast by $\mathsf{id_3}$, we have:

$$\mathsf{BBcast} : (\mathsf{cr_1}, \mathsf{b_1}), (\mathsf{cr_2}, \mathsf{b_2}), (\mathsf{cr_3}, \mathsf{b_3}),$$

corresponding to the outcome A, B, B. Moreover, no honest party can detect cheating: although there is only one ballot cast for $\mathsf{id_1}$ and $\mathsf{id_2}$, they cannot detect this, since they have the same pair $(\mathsf{cr_1}, \mathsf{b_1})$ assigned to them, and it is present on BBcast as expected. The adversary does not have to compromise the decryption process or the mixnet: the ciphertexts $\mathsf{c_1}, \mathsf{c_2}, \mathsf{c_3}$ that are to be tallied represent the corrupted outcome desired by the adversary.

New clash attacks. The attack above relies on the fact that the adversary controls the voting platform VP of both $\mathsf{id_1}$ and $\mathsf{id_2}$. Moreover, $\mathsf{id_1}$ and $\mathsf{id_2}$ have to vote for the same candidate A and, if one of $\mathsf{id_1}$ and $\mathsf{id_2}$ audits the ballot, it is assumed that the other party will do the same, or that voters do not notice if the same ballot is repeatedly reused by VP. Given this, it may be considered that the chances of successfully mounting the attack are small. In fact, we show that, if we consider the individual verification procedures that are currently deployed online [5], and if revoting is allowed, then the adversary can mount a successful clash attack without having to control VP and without having to make strong assumptions related to voter behaviour.

Among the deployment options for individual verifiability sketched above, we consider the version where BBcast only shows the last ballot cast for each cr, where voters can check their ballot on BBcast any time after voting. We then consider the following typical voter behaviour: the voters perform individual verification of BBcast immediately after casting their ballot. In the scenario described above, consider the sequence where first id_1 casts a ballot b and then id_2 casts a ballot b'. Then

1. BBcast first includes $(\mathsf{cr}_1, \mathsf{b})$.
2. When id_2 casts a vote for any candidate, $(\mathsf{cr}_1, \mathsf{b})$ is replaced on BBcast with $(\mathsf{cr}_1, \mathsf{b}')$.

The voter id_1 successfully verifies the ballot b after step (1). The voter id_2 also successfully verifies the ballot b' after step (2). To an external observer, the replacement of b by b' looks like revoting performed by cr_1, so the attack is publicly undetectable.

We note that other deployment options 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' we have:

$$\mathsf{BBcast} : \ldots, (\mathsf{cr}_1, \mathsf{b}), \ldots, (\mathsf{cr}_1, \mathsf{b}'), \ldots$$

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, even if all voters verify their ballots at the end of the voting phase.

Clash attack on abstaining voters. 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 to $\mathsf{id}_1, \mathsf{id}_2$ can, again, cast an adversarial ballot. Remarkably, this attack is more difficult to detect and prevent rather than the case where voters cast ballots. In our analysis, we show that some procedures that help against the latter attack are still vulnerable to this empty ballot attack.

Although in hindsight this class of attacks is simple, we note that we discovered it only after developing the formal models that we describe in section 4 and attempting to perform the proofs with Tamarin. In section 5, we instantiate our framework with different scenarios (parameterised by corrupted parties and verification procedures), in order to systematically find such weaknesses in the protocol, or to automatically prove some of its variants secure. Importantly, we show that security with respect to the formal models that we propose indeed entails a global notion of end-to-end verifiability, ensuring that the outcome correctly captures the intent of honest voters.

New individual verification procedures. To obtain stronger E2E security guarantees, we propose two simple ways of improving the individual verifiability procedure, which we prove to be secure. We note that more research is needed to assess their usability and deployment in practice. Formal specifications for each procedure are in subsequent sections. As a first solution, we propose that: (i) the bulletin board should contain the list of all ballots cast by each cr; (ii) the voters should be able to easily access the list of all cast ballots for a given cr, compare them with ballots the voter knows to have cast, and complain if there is a mismatch. In general, if revoting is allowed, then voters should check their ballots only after the voting phase has finished. Otherwise, independently of clash attacks, a dishonest server may replace their ballot with a corrupted one.

The solution above would protect against the new class of clash attacks, where VP is not corrupted. However, if all of $\mathsf{VR}, \mathsf{VS}, \mathsf{VP}$ are corrupted, the original clash attack would still be applicable. The countermeasure that [49] proposes is for voters to contribute to the randomness in the ballot, this way ensuring that two different voters will, with high probability, have different ballots. However, in this case, our new clash attack still holds, since the two different ballots may be verified at different times or different places on BBcast. This does not contradict the security proof of [49]: their proof considers a scenario without revoting where BBcast is published and verified at the end of the election. In that case, it can be publicly ensured that no two distinct ballots can be verified for the same credential.

To prevent both types of clash attack, we propose a solution which ensures that the credentials of two distinct voters who successfully verify their ballots are distinct. This requires to include a credential verification step by voters at some point in the protocol. Instead of being randomly chosen by the registrar, we require that each cr_i is equal to $\mathsf{h}(\langle \mathsf{id}_\mathsf{i}, \mathsf{pwd}_\mathsf{i} \rangle)$, where h is a collision-resistant hash function and pwd_i is the password for id_i. Then, voters verify that the cr communicated to them does indeed satisfy the required relation with respect to their id and pwd. Note that:

- This check should be performed on the same device that voters use to audit or verify their ballots, preventing a potentially corrupted VP to subvert the verification of the cr.
- We assume that the id of each voter is unique, and this relies on a trusted administrator, or a pre-existent identity infrastructure.

Since both $\mathsf{id_i}$ and $\mathsf{pwd_i}$ are private, we assume that their combination contains enough entropy to prevent a brute-force search for $\mathsf{pwd_i}$ from the public $\mathsf{cr_i}$. We leave as future work the search of solutions with better security as well as usability against brute-force attacks.

3 Formal verification preliminaries

Symbolic verification assumes the cryptographic primitives are perfect and represents them relying on a term algebra. First we review the general model of Tamarin, and then we describe specific abstractions that are required for symbolic verification of voting protocols.

3.1 Modeling and verifying protocols in Tamarin

We present the (multiset) rewriting framework as instantiated by Tamarin, referring to [50, 57] 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 1. *Let* $\mathcal{F}_{\mathcal{H}} = \{\mathsf{pk}, \mathsf{enc}, \mathsf{dec}, \mathsf{h}, \mathsf{ver}\}$ *and* $\mathcal{E}_{\mathcal{H}}$*:*

$$\begin{array}{rrcl} & \mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) & = & \mathsf{x}, \\ (\forall\, i) & \mathsf{ver}(\mathsf{enc}(\mathsf{x_i}, \mathsf{y}, \mathsf{z}), \mathsf{y}, \langle \mathsf{x_1}, \dots, \mathsf{x_k} \rangle) & = & \mathsf{ok}. \end{array}$$

A term $\mathsf{enc}(\mathsf{m}, \mathsf{pk}(\mathsf{k}), \mathsf{r})$ *represents a ciphertext where* m *represents the corresponding plaintext,* k *represents a private key, whose public counterpart is* $\mathsf{pk}(\mathsf{k})$*, and* r *represents the randomness used by the encryption algorithm.* k *and* r *are typically fresh (private) names, while* m *may be a public name or a more complex message.*

The first equation in $\mathcal{E}_{\mathcal{H}}$ *specifies that standard decryption property of asymmetric encryption. Cryptography is assumed perfect: there is no other way to derive messages other than applying function symbols and equations. In particular,* $\mathsf{h}(\mathsf{m})$ *does not reveal* m *and we have* $\mathsf{h}(\mathsf{m_1}) \neq \mathsf{h}(\mathsf{m_2})$ *for any* $\mathsf{m_1} \neq \mathsf{m_2}$*. The second set of equations models the verification of a zero-knowledge proof showing that the corresponding plaintext is within a certain range* $\mathsf{x_1}, \dots, \mathsf{x_k}$ *and that the public encryption key is equal to* y*. We assume the proof is part of the encryption, therefore, we do not use an explicit term for it.*

The set of symbols is extended with fact symbols to represent adversarial knowledge, protocol state, freshness information, etc. A fact is represented by $F(t_1, \dots, t_k)$, where F is a fact symbol and $t_1, \dots, t_k$ 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 $!F$ is used in Tamarin to distinguish persistent facts, but all facts in our models can be assumed persistent, so we do not use this notation in the paper to avoid clutter.

A multiset rewriting rule is defined by $[L]\!\!-\!\![\ M\ ]\!\!\rightarrow\![N]$, 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 $[L]\!\!-\!\![\ E, M\ ]\!\!\rightarrow\![N]$ where L may contain epressions $x = t$ to define local variables and E is a set of equations of the form $u \approx v$. For two multisets of facts M_0, M_1 and rule $R = [L]\!\!-\!\![\ E, M\ ]\!\!\rightarrow\![N]$ 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; *protocol rules* allow to specify the behaviour of honest parties.

Example 2. *Consider the set of rules Q_{keys}:*

$$\begin{array}{ll} (i) & [\ \mathsf{Fr(k)}\ ]\!-\!\![\ \mathsf{Key(k)}\ ]\!\!\rightarrow [\ \mathsf{Pk(pk(k)), Sk(k), Out(pk(k))}\ ], \\ (ii) & \mathsf{let}\ \ \mathsf{c = enc(x,y,r)}\ \ \mathsf{in} \\ & [\ \mathsf{Sk(x), Pk(y), Fr(r)}\ ]\!-\!\![\ \mathsf{Enc(c)}\ ]\!\!\rightarrow [\ \mathsf{Out(c)}\ ], \\ (iii) & [\ \mathsf{Sk(x), In(y)}\ ]\!-\!\![\ \mathsf{Dec(y)}\ ]\!\!\rightarrow [\ \mathsf{Out(dec(y,x))}\ ]. \end{array}$$

The first rule models the generation of a fresh secret key k*, which is stored for later use in* $\mathsf{Sk(k)}$*, 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* $\mathsf{Key(k)}$*,* $\mathsf{Enc(c)}$ *and* $\mathsf{Dec(y)}$ *record the respective events in the execution trace. We use the Tamarin* let ... 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 $M_0, M_1, \dots, M_n$ and a sequence of rules $R_1, \dots, R_n \in P$ such that, for every $i \in \{1, \dots, n\}$, 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 the message $v\theta_i$ at timepoint i.

We consider a set of timepoint variables, denoted by $i, j, l, \dots$, which will be interpreted over rational numbers. A *trace atom* is either a term equality $t_1 = t_2$, or a timepoint ordering $i \prec j$, or a timepoint equality $i = j$, or an action fact $F@i$ 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 all variables are bounded, is defined recursively as expected, with the following notable case: $\tau \vDash F@i$ if and only if $F \in facts(\tau, i)$. 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:

$$\begin{array}{rcl} P \vDash \Phi & \text{iff} & \forall \tau \in \mathsf{tr}(P).\ \tau \vDash \Phi, \\ P;\Psi \vDash \Phi & \text{iff} & \forall \tau \in \mathsf{tr}(P).\ \tau \vDash \Psi \Rightarrow \Phi. \end{array}$$

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 3. *Continuing Example 2, the formula* $\Phi_{\mathsf{sec}} : \forall x, i.\ \mathsf{Key}(x)@i \Rightarrow \neg\exists j.\ \mathsf{K}(x)@j$ *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* $Q_{\mathsf{keys}} \nvDash \Phi_{\mathsf{sec}}$*, since the third rule in* Q_{keys} *may decrypt messages published by the second rule. The restriction* $\Psi_{\mathsf{keys}} : \forall x, i.\ \mathsf{Dec}(x)@i \Rightarrow \neg(\exists j.\ \mathsf{Enc}(x)@j)$ *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 $F_1(x_1, x_2) \Rightarrow F_2(y_1, y_2)$ represents $\forall x_1, x_2, i.\ F_1(x_1, x_2)@i \Rightarrow \exists y_1, y_2, j.\ F_2(y_1, y_2)@j$. 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 facts. For example, $F(x) \Rightarrow F_1(y) \vee \neg F_2(y)$ represents $\forall x, i.\ F(x)@i \Rightarrow \exists y, j.\ F_1(y)@j \vee \neg(\exists y, j.\ F_2(y)@j)$. 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.

3.2 Voting related abstractions

We extend the perfect cryptography assumption to cover two specific operations of e-voting protocols: ballot construction and vote counting. We explain and justify the corresponding abstractions next.

Ballot construction. Voting platform audits in Helios allow voters to ensure that the ballot to be cast correctly encodes their vote [8, 18]. The underlying security argument is probabilistic and is based on expectations regarding the voter audit behaviour. For our model of Helios, we assume that the voting platform correctly encrypts the voter's choice, therefore assuming that the ballot audit perfectly achieves its goals and does not affect other parts of the system. We note that the same assumption is used in previous symbolic models of Helios and for symbolic analysis of electronic voting in general [10, 46, 58, 27, 32, 28]. However, a malicious voting platform may still choose the randomness required for the encryption in an arbitrary way, even in presence of audits, and we allow this in our models. In this, our model of the voting platform is more general than previous symbolic models.

Vote counting. We consider voting systems that use a public bulletin board BB to record the execution of the election. We can distinguish the following important parts:

$$\begin{aligned}
\mathsf{BBcast} &: (\mathsf{cr}_1, [\mathsf{b}_1^1, \mathsf{b}_1^2, \ldots]),\ (\mathsf{cr}_2, [\mathsf{b}_2^1, \mathsf{b}_2^2, \ldots]),\ \ldots,\\
\mathsf{BBtally} &: (\mathsf{cr}_1, \mathsf{b}_1), \ldots, (\mathsf{cr}_n, \mathsf{b}_n).
\end{aligned}$$

BBcast records the ballots cast during the voting phase. It can be used by voters to verify that their ballots correctly reach the bulletin board, and also by public auditors to ensure the behaviour of BB follows the specification. Above, BBcast records for each cr_i the list of all cast ballots $[\mathsf{b}_\mathsf{i}^1, \mathsf{b}_\mathsf{i}^2, \ldots]$. The revoting policy and ballot validity checks determine which ballot from $\mathsf{BBcast}(\mathsf{cr}, \_)$, if any, to keep for the tally, and these are stored in $\mathsf{BBtally}$. The first part of an universally verifiable tally (called *good sanitization* in [27]), allows any external observer to ensure that $\mathsf{BBtally}$ is correctly obtained from BBcast.

The second part of an universally verifiable tally (called *good counting* in [27]) takes as input the multiset of ballots $\{\!|\mathsf{b}_1, \ldots, \mathsf{b}_n|\!\}$ from $\mathsf{BBtally}$ and outputs a result ρ, which corresponds to a multiset of votes $\{\!|\mathsf{v}_1, \ldots, \mathsf{v}_n|\!\}$, and a zero-knowledge proof π showing that these votes correctly correspond to $\{\!|\mathsf{b}_1, \ldots, \mathsf{b}_n|\!\}$. The proof π is, in general, a complex cryptographic primitive whose security should also be analysed independently of the voting protocol. It may consist of two components, one for a verifiable mixnet [56, 52, 43, 60], and one for verifiable decryption [42, 53, 29]. As typically done in symbolic models [28, 27], in order to capture the properties of this second part of the tally, we restrict the class of evoting protocols to those where the public information present on BB before the final tally is sufficient to determine the vote v_i that is encoded by a ballot b_i. This is in general true for verifiable protocols, in particular for those where b_i contains an encryption of v_i, like Helios or Belenios. We note, however, that there are verifiable protocols where this assumptions is not true, for example when the ballot b_i contains an information theoretic commitment to v_i [51, 36].

Having made this assumption, we consider a function open (like in [28]) that, when applied to a ballot b, returns the vote encoded by b according to public information of the election; similar notions are the wrap predicate in [27] and the $\mathsf{extract}$ function in [20]. Then the final outcome corresponding to $\mathsf{b}_1, \ldots, \mathsf{b}_n$ from $\mathsf{BBtally}$ is considered to be $\mathsf{open}(\mathsf{b}_1), \ldots, \mathsf{open}(\mathsf{b}_n)$. For Helios, when pk is the public key of the election, we define open as follows:

$$\begin{aligned}
\mathsf{b} = \langle \mathsf{h}(\mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r})), \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r}) \rangle \quad &\Longrightarrow \quad \mathsf{open}(\mathsf{b}) = \mathsf{v},\\
\mathsf{b} = \bot \quad &\Longrightarrow \quad \mathsf{open}(\mathsf{b}) = \bot.
\end{aligned}$$

We stress that this abstraction is justified by the fact that: (i) $\mathsf{BBtally}$ is public and fixed at the end of the election; (ii) ρ is claimed by election authorities to be the result corresponding to $\mathsf{BBtally}$; (iii) the expected correspondence between $\mathsf{BBtally}$, the public key pk and the result ρ, as defined by open, can be formalised as an NP relation. This defines a precise cryptographic task for a zero-knowledge proof, and dedicated cryptographic primitives can be used to implement it. We note that there are symbolic methods for ensuring that software correctly implements the underlying cryptographic constructions [40, 9], and also for explicitly incorporating zero-knowledge proofs within symbolic models for security protocols [12, 11].

4 Formal models for election verifiability

We need two specifications for formal verification:

- $\mathcal{S}$ - the specification of the considered e-voting system, including the procedures for honest parties and the corruption abilities of the adversary;
- Φ - the specification of 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 $\mathcal{S}$. In order to have a general security definition Φ that can be applied to any desired $\mathcal{S}$, we assume, similar to previous symbolic definitions, e.g. [10, 27, 28], that $\mathcal{S}$ defines a set of basic events, that are standard for electronic voting protocols: a voter is registered; a candidate is electable; a vote is cast; a ballot is verified; the result is recorded; a voter is corrupted. Formally, relying on fact symbols

$$\mathsf{BBreg}, \mathsf{Cand}, \mathsf{Vote}, \mathsf{Verified}, \mathsf{BBtally}, \mathsf{Corr},$$

these events will be represented by action facts associated to the corresponding rules in the specification $\mathcal{S}$. We refer to Figure 1 for an example specification showing how these events are added to rules. In more detail:

- ***BBreg and Cand.*** A voting system assumes a set of eligible voters, represented by the identities $\mathsf{id}_1, \dots, \mathsf{id}_\mathsf{n}$. They are determined by a trusted administrator or other election authorities. The number n of eligible voters is in general public, however there are elections where it is required that voter identities remain private, and in that case public credentials $\mathsf{cr}_1, \dots, \mathsf{cr}_\mathsf{n}$ are used to record and track voter ballots on the bulletin board. We assume that for each eligible credential cr_i there is a corresponding event $\mathsf{BBreg}(\mathsf{cr}_\mathsf{i})$ recorded in the trace. We note that cr can be equal to id, if the specification of the voting system prescribes it. The set of candidates to be elected is represented by the set of events $\mathsf{Cand}(\mathsf{v}_1), \dots, \mathsf{Cand}(\mathsf{v}_\mathsf{k})$ in the trace.
- ***Vote.*** Formally, when a voter with private identity id and public credential cr casts a vote v, we assume the action fact $\mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ is recorded in the corresponding execution trace.
- ***Verified.*** Individual verification procedures allow voters to verify that their cast vote is correctly recorded by the voting system, possibly relying on special section of the bulletin board. If a voter with identity id and public credential cr has successfully performed the verification steps related to a vote for v, we assume the action fact $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ is recorded in the execution trace.
- ***BBtally.*** In a symbolic execution trace, the multiset of ballots to be tallied is formally defined by events of the form $\mathsf{BBtally}(\mathsf{cr}, \mathsf{b})$ recording that the ballot b is tallied for the respective cr. According to the vote counting abstraction discussed in subsection 3.2, if all $\mathsf{BBtally}$ events in a trace are $\{\mathsf{BBtally}(\mathsf{cr}_1, \mathsf{b}_1), \dots, \mathsf{BBtally}(\mathsf{cr}_\mathsf{n}, \mathsf{b}_\mathsf{n})\}$, then we consider the result of the election to be $\{\mathsf{open}(\mathsf{b}_1), \dots, \mathsf{open}(\mathsf{b}_\mathsf{n})\}$.
- ***Corrupted Voter.*** A registered voter obtains a public credential and possibly private credentials to be used for the authentication. The action fact $\mathsf{Corr}(\mathsf{id}, \mathsf{cr})$ denotes the event that all these credentials are leaked to the adversary. The cause of the leak may be the voter or the infrastructure used to transmit or store credentials. Note that other components of the voting system - e.g. the trustees, the registrar, the voting server, the voting platform - may also be corrupted, but we do not require these corruption events to be recorded in the execution trace. The reason this is required for voters is that, if private credentials are leaked, in general we cannot prevent the adversary from casting a ballot for the respective voter, and we need to account for this in the security property.

Figure 1: 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, VR, VS, VP, EA as introduced in section 2.

SETUP PHASE

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

$\mathsf{R}^{\mathsf{A}}_{\mathsf{cand}}$: **determine candidates to be elected**
$\mathsf{let}\ \ \mathsf{vlist} = \langle \mathsf{v}_1, \dots, \mathsf{v}_k \rangle\ \ \mathsf{in}$
$[\ \mathsf{In}(\mathsf{vlist})\ ]$
$-\!\![\ \mathsf{Cand}(\mathsf{v}_1), \dots, \mathsf{Cand}(\mathsf{v}_k), \mathsf{BBcand}(\mathsf{vlist})\ ]\!\!\rightarrow$
$[\ \mathsf{Cand}(\mathsf{v}_1), \dots, \mathsf{Cand}(\mathsf{v}_k), \mathsf{Out}(\mathsf{vlist})\ ]$

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

$\mathsf{R}^{\mathsf{VR}}_{\mathsf{reg}}$: **register voter with credential and password**
$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{cr}), \mathsf{Fr}(\mathsf{pwd})\ ]\!\!-\!\![\ \mathsf{BBreg}(\mathsf{cr})\ ]\!\!\rightarrow$
$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{pwd}), \mathsf{BBreg}(\mathsf{cr}), \mathsf{Out}(\mathsf{cr})\ ]$

$\mathsf{R}^{\mathsf{VS}}_{\mathsf{bb}}$: **setup initial** BBcast **for registered voters**
$[\ \mathsf{BBreg}(\mathsf{cr})\ ]\!\!-\!\![\ \mathsf{BBcast}(\mathsf{cr}, \perp)\ ]\!\!\rightarrow [\ \mathsf{BBcast}(\mathsf{cr}, \perp)\ ]$

VOTING PHASE

$\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$: **construct a ballot, authenticate and send it to** VR
$\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r});\ \ \mathsf{b} = \langle \mathsf{h}(\mathsf{c}), \mathsf{c} \rangle;$
$\quad \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle)\ \ \mathsf{in}$
$[\ \mathsf{Cand}(\mathsf{v}), \mathsf{BBkey}(\mathsf{pkey}), \mathsf{Fr}(\mathsf{r}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{pwd})\ ]$
$-\!\![\ \mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow$
$[\ \mathsf{Out}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{h}(\mathsf{c}))\ ]$

$\mathsf{R}^{\mathsf{VR}}_{\mathsf{auth}}$: **authenticate voter and forward ballot to** VS
$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{hc}, \mathsf{c} \rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{b} \rangle)\ \ \mathsf{in}$
$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{pwd})\ ]\!\!-\!\![\ \mathsf{a}' \approx \mathsf{a}\ ]\!\!\rightarrow$
$[\ \mathsf{Cast}(\mathsf{cr}, \mathsf{b})\ ]$

$\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$: **effectively cast the ballot on BB**
$[\ \mathsf{Cast}(\mathsf{cr}, \mathsf{b})\ ]\!\!-\!\![\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow [\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$

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

TALLY PHASE

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

$\Psi^{\mathsf{VS/EA}}_{\mathsf{tally}}$: **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 receipt on** BBcast
$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{hc}, \mathsf{c} \rangle\ \ \mathsf{in}$
$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{hash}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$
$-\!\![\ \mathsf{hash} \approx \mathsf{hc}, \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow [\ ]$

$\mathsf{R}^{1}_{\mathsf{ver}}$: **same as $\mathsf{R}^{0}_{\mathsf{ver}}$, but performed in the tally phase**

$\mathsf{R}^{2}_{\mathsf{ver}}$: **voter verifies the receipt on** BBtally
$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{hc}, \mathsf{c} \rangle\ \ \mathsf{in}$
$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{hash}), \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$
$-\!\![\ \mathsf{hash} \approx \mathsf{hc}, \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})\ ]\!\!\rightarrow [\ ]$

$\mathsf{R}^{3}_{\mathsf{ver}}$: **voter verifies there is no ballot on** BBtally
$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{pwd}), \mathsf{BBtally}(\mathsf{cr}, \mathsf{x})\ ]$
$-\!\![\ \mathsf{x} \approx \perp, \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \perp)\ ]\!\!\rightarrow [\ ]$

$\mathsf{R}^{0}_{\mathsf{ver}}$ and $\mathsf{R}^{1}_{\mathsf{ver}}$ 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
$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @\ j\ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @\ i\ \wedge\ i \prec j$
$\Rightarrow \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @\ l$

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

$\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}$: **corrupt trustee to reveal the secret key**
$[\ \mathsf{Sk}(\mathsf{sk})\ ]\!\!-\!\![\ ]\!\!\rightarrow [\ \mathsf{Out}(\mathsf{sk})\ ]$

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

$\mathcal{C}^{\mathsf{VR}}_{\mathsf{auth}}$: **corrupt registrar to not authenticate voter**
$[\ \mathsf{In}(\langle \mathsf{cr}, \mathsf{b} \rangle)\ ]\!\!-\!\![\ ]\!\!\rightarrow [\ \mathsf{Cast}(\mathsf{cr}, \mathsf{b})\ ]$

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

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

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

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

4.1 Voting protocols, procedures and adversary

A *protocol component* is a pair $(\mathcal{R}, \Psi)$, where $\mathcal{R}$ is a set of protocol rules and Ψ is a set 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 $\mathcal{S} = (\ (\mathsf{R}_\mathcal{P}, \Psi_\mathcal{P}),\ (\mathsf{R}_\mathcal{V}, \Psi_\mathcal{V}),\ (\mathsf{R}_\mathcal{A}, \Psi_\mathcal{A})\ )$, then we let $\mathcal{S} \vDash \Phi$ iff $\mathsf{R}_\mathcal{P} \cup \mathsf{R}_\mathcal{V} \cup \mathsf{R}_\mathcal{A};\ \Psi_\mathcal{P} \wedge \Psi_\mathcal{V} \wedge \Psi_\mathcal{A} \vDash \Phi$.

While $\mathcal{V}$ may also be considered as part of the protocol specification $\mathcal{P}$, we treat it separately because we will analyse the security properties that are ensured by various verification procedures $\mathcal{V}$ in the context of the same basic protocol $\mathcal{P}$ and 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. Next, we discuss and illustrate in more detail each component of a specification $(\mathcal{P}, \mathcal{V}, \mathcal{A})$.

Protocol model $\mathcal{P}$. The component $\mathcal{P}_\mathcal{H} = (\mathcal{R}_\mathcal{H}, \Psi_\mathcal{H})$ from Figure 1a corresponds to the Helios protocol as introduced in section 2, omitting the individual verification procedures. Rules can be formally separated into phases with additional restrictions in Tamarin (omitted from picture). The fact $\mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{pwd})$ represents a voter with identity id that was communicated the public credential cr and the password pwd at registration. The rule $\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$ models the actions of the voting platform when asked to construct a ballot for the given credentials and desired vote. For casting to the bulletin board, the ballot needs to be authenticated with pwd. The mechanism used for authentication is abstracted with a simple construction based on a hash function. The voting server verifies the validity of ballots to be posted on BBcast, checking the hash of the encryption and the range proof showing that the encryption is for an eligible candidate (we model this with equations from Example 1). For tally, VS selects the last ballot from BBcast for each cr. These actions of the server are universally verifiable by auditors, since BBcast and $\mathsf{BBtally}$ are public. That is why the corresponding restrictions are labeled by both VS and EA.

Individual verification procedures $\mathcal{V}$. In Figure 1b, we present rules and restrictions that can be combined to obtain individual verification components for Helios. The premises of rules $\mathsf{R}^{\mathsf{i}}_{\mathsf{ver}}$ typically contain two facts: one which includes the expected vote and tracking information for a posted ballot; and one which includes a particular cast ballot as seen on the bulletin board. A basic verification check performed in the rule ensures that the tracking information for the posted ballot matches the one of the ballot b shown as cast on the bulletin board. Further verification checks on b are enabled by the action fact VerB, which records some parameters relevant to the verification operation. The restrictions $\Psi_\mathcal{V} \in \{\Psi_{\mathsf{last}}, \Psi_{\mathsf{mine}}\}$ can then express additional trace properties that should be true for these parameters. If and only if all tests performed by $\mathsf{R}^{\mathsf{i}}_{\mathsf{ver}}$ and $\Psi_\mathcal{V}$ are true, the rule $\mathsf{R}^{\mathsf{i}}_{\mathsf{ver}}$ can be executed and a corresponding event $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ can be recorded in the trace to represent a successful verification event.

For example, $(\mathsf{R}^{0}_{\mathsf{ver}}, \emptyset)$ represents the basic verification procedure where the voter simply checks whether the obtained receipt is present on BBcast, with no constraint. In addition, $(\mathsf{R}^{0}_{\mathsf{ver}}, \Psi_{\mathsf{last}})$ ensures that the verified ballot is the last one currently cast for that cr on BBcast. In addition, $(\mathsf{R}^{0}_{\mathsf{ver}}, \Psi_{\mathsf{last}} \wedge \Psi_{\mathsf{mine}})$ requires that all ballots previously cast should be recognised as originating from the voter who performs the verification. The specification of the protocol may require that verification is performed after the end of the voting phase, which can be handled by a rule $\mathsf{R}^{1}_{\mathsf{ver}}$ similar to $\mathsf{R}^{0}_{\mathsf{ver}}$. The rule $\mathsf{R}^{2}_{\mathsf{ver}}$ shows that ballots can also be verified on $\mathsf{BBtally}$ after it was computed from BBcast (which is different from verifying BBcast with $\mathsf{R}^{1}_{\mathsf{ver}}$). The rule $\mathsf{R}^{3}_{\mathsf{ver}}$ used in a component $(\{\mathsf{R}^{0}_{\mathsf{ver}}, \mathsf{R}^{3}_{\mathsf{ver}}\}, \Psi)$ means that voters who did not vote can verify that there is no ballot tallied in their name on the bulletin board. We discuss more about these procedures in section 5.

Adversarial corruption rules $\mathcal{A}$. Election verifiability should also take into account the ability of the adversary to corrupt certain participants and infrastructure. It is standard to assume that the adversary may corrupt voters, the holders of secret keys (also known as trustees) and the communication network. In addition, the adversary may control other parties (e.g registrars), voting platforms, election servers, etc. We model all such abilities by a set of (adversarial) corruption rules $\mathcal{A}$, which is a parameter of our definition, and is an addition to the standard network and intruder deduction rules. For our case study, we will consider different scenarios where $\mathcal{A}$ consists of a subset of rules from Figure 1c with respect to the Helios specification (and we have a similar set for Belenios). The corruption rules are of two sorts: either leaking information to the adversary (rules $\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}$), or allowing the adversary to control parts of the voting infrastructure corresponding to parties VR, VS or VP. The rule $\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}$ affects the electoral roll: a corrupted registrar takes instructions on how to construct and allocate public credentials to voters. For example, the same credential may be allocated to two different voters, in order to mount a clash attack. Note that adversarial inputs are not constrained, letting the adversary choose the desired attack strategy. The rule $\mathcal{C}^{\mathsf{VR}}_{\mathsf{auth}}$ (respectively $\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$), models a corrupted VR (respectively VS) that may forgo the prescribed way of casting ballots to the bulletin board. The rule $\mathcal{C}^{\mathsf{VP}}_{\mathsf{vote}}$ represents a corrupted voting platform which allows the adversary to select the randomness used in the encryption of the vote. This will generate a valid ballot with adversarially chosen randomness. As discussed in subsection 3.2, we assume the ballot is valid because voters have the option to audit their platform.

Restrictions can be used to reduce the adversarial power, when it is justified by the considered scenario. For example, if one makes the assumption that the ballot-casting infrastructure is honest, it follows that, if a voter casts multiple ballots, we can ensure that they are delivered to the bulletin board in the right order. We can model this by the restriction Ψ_{order} used in conjunction with the corresponding adversarial rules.

4.2 Symbolic election verifiability

Our definition of election verifiability is a conjunction of trace formulas that restricts the possible outcome of the election and the possible outcome of individual verification procedures. The goal is to guarantee that all successfully verified votes are correctly counted in the final result, and also to restrict the set of additional, potentially malicious, votes in the result.

Revoting policy. Assume a voter with credentials $(\mathsf{id}, \mathsf{cr}, \mathsf{pwd})$ has cast several ballots corresponding to votes $\mathsf{v}_1 \ldots, \mathsf{v}_k$. 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 respective 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, we would be able to ensure that all successfully verified votes are contained in the final outcome. However, some systems may prescribe weaker verification procedures, which can only ensure for example that the ballot made it to the bulletin board. In that case, the formal verifiability property needs to add 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 $(\mathsf{id}, \mathsf{cr}, \mathsf{v})$. We note that, in this paper, we are mainly interested in verification procedures that are strong enough to imply that the verified vote is counted, thus in general we may not need $\Omega(\mathsf{id}, \mathsf{cr}, \mathsf{v})$; formally, this formula is set to true in that case. Another natural policy is Ω^{last} from Figure 2, specifying that the vote v should be the last one cast by the given voter. Although Φ_{iv} and $\Phi^{\diamond}_{\mathsf{E2E}}$ defined below depend on the revoting policy Ω, it will usually be clear from the context and we omit it from notation.

Definition 2. 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 an e-voting specification $\mathcal{S}$ satisfies symbolic election verifiability iff $\mathcal{S} \vDash \Phi^{\diamond}_{\mathsf{E2E}}$.

• Φ_{iv} corresponds intuitively to ***individual verifiability***, i.e. *successful verification* should imply that the corresponding vote is indeed *counted as intended* in the final outcome. For successful verification, we have seen that $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$ can require various properties to be ensured on an execution trace, as in Figure 1b. To obtain a precise, practical notion of counted as intended, the formula Φ_{iv} needs to be complemented with the rest. Recall that events and variables to the right of $\Rightarrow$ in a (simplified) trace formula are universally quantified. Therefore Φ_{iv} requires that any ballot b claimed by election authorities to represent a voter's choice satisfies the required constraint $\mathsf{v} = \mathsf{open}(\mathsf{b})$. We also note that, by public audits, it can be ensured that there is always a ballot recorded for every registered credential; this can be the empty ballot $\perp$ if there was no ballot cast for that credential.

Figure 2: Symbolic election verifiability

Formulas defining symbolic E2E

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

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

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

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

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

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

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

Additional property for homomorphic tally

Φ_{cand} : $\mathsf{BBtally}(\mathsf{cr},\mathsf{b}) \wedge \mathsf{b} \neq \perp \Rightarrow \mathsf{Cand}(\mathsf{open}(\mathsf{b}))$

Examples of revote policies Ω

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

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

Possible weakening of individual verifiability

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

Restrictions to achieve multiset-based E2E

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

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

• 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, all credentials accounted for in the final tally should correspond to registered credentials.

• Φ_{cl} specifies that ***no clash*** should occur on public identities used in successful individual verification procedures. In other words, if there is such a clash, then the individual verification procedure should detect it and return a negative result. In subsection 2.2, we have shown a few scenarios where a violation of this property leads to practical attacks on verifiability of Helios; therefore this property is, in some sense, necessary. On the other hand, it is also a general property, and it has a significant contribution towards a minimal set of properties and restrictions that are sufficient for end-to-end verifiability.

• $\Phi^{\diamond}_{\mathsf{adv}}$. For each ballot to be tallied, we can make the following distinction: either there was a corresponding instance of the voting procedure executed by an eligible voter, or else this is a vote cast by the adversary. The property $\Phi^{\diamond}_{\mathsf{adv}}$ specifies precisely what votes are expected to be cast by the adversary. This may depend on trust assumptions related to participants in the voting system, thus the property is parametrised by a formula $\Phi^{\diamond}_{\mathsf{adv}}$. At least, we expect that the adversary is able to cast votes for corrupted voters. The strongest version of $\Phi^{\diamond}_{\mathsf{adv}}$, obtained by using $\Phi^{\bullet}_{\mathsf{adv}}$ in place of $\Phi^{\diamond}_{\mathsf{adv}}$, specifies that no other votes are expected to be cast by the adversary. If we do not make any trust assumptions about the election authorities then, to our knowledge, there is no voting system that ensures this property. It can be achieved by Helios when the registrar and the voting server are honest, and by Belenios when at least one of these parties is honest, but we note that the registrar plays a different role in Belenios, providing secret signing keys to voters [33].

To evaluate the degree of security that can be achieved in stronger corruption scenarios, for example when all election authorities are corrupted, we consider a slightly weaker version of $\Phi^{\diamond}_{\mathsf{adv}}$. The formula $\Phi^{\circ}_{\mathsf{adv}}$ specifies that, in addition to corrupted voters, the adversary may cast ballots for voters who do not verify their votes. To illustrate this case, we consider a few scenarios where votes from honest voters may not reach the final tally: (i) a voter does not vote; (ii) a voter does vote but the corresponding ballot is dropped by corrupted election infrastructure; (iii) revoting is allowed, and a first vote submitted by the voter is replaced by corrupted election infrastructure. In all these cases, the voter has the option to perform the verification procedure associated to the voter choice (abstention or desired candidate) and to the corresponding public credential. $\Phi^{\circ}_{\mathsf{adv}}$ says that the adversary is able to take over the respective credential if and only if the voter did not perform the verification procedure, or if its outcome was negative, in which case the voter should complain to the relevant authority. We note that accountability and dispute resolution mechanisms should complement election verifiability for this case [47, 15].

• Φ_{cand} is required for the particular case of systems whose tally method is based on homomorphic encryption. Then we need to ensure that ballots entering the final tally 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 simply removed from the final 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 will see examples of procedures that achieve strong as well as examples that only achieve weak individual verifiability.

Comparison with [28]. The properties defining $\Phi^{\diamond}_{\mathsf{E2E}}$ are similar in nature to properties defining symbolic verifiability in [28]. There are, however, important technical differences that allow us to overcome the limitations of [28] discussed in the introduction:
– we introduce cr as an argument in $\mathsf{Verified}$, Vote and we remove id as an argument in $\mathsf{BBtally}$. In [28] these are $\mathsf{Verified}(\mathsf{id}, \mathsf{v})$, $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$ and $\mathsf{BBtally}(\mathsf{id}, \mathsf{cr}, \mathsf{b})$. Our version of $\mathsf{Verified}$, Vote makes a stronger connection between these events and public information on BB. In the same spirit, removing id from $\mathsf{BBtally}$ makes the verifiability property more transparent (the connection to id is only known to the voting server in [28]).
– the hidden connection between id and cr in $\mathsf{BBtally}$ requires [28] to come up with additional artefacts and 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 $\mathsf{BBtally}$ and voter information in $\mathsf{Verified}$ and Vote. 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 the *no clash* property 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 $\mathsf{Verified}$ events to other events in the execution trace.
– we allow a more liberal handling of corruption and 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.

4.3 Multiset-based election verifiability

Following the approach from [26, 28, 27], we consider a definition of end-to-end verifiability based on multisets. The advantage of this definition is that it captures the desired verifiability property with mathematical notions (like set theory and natural number theory) that are more closely related to the real-world execution of a vote-counting election. The drawback is that automated verification modulo these theories is more problematic than for the symbolic approach. The goal of this section is to show that symbolic verifiability entails multiset-based verifiability, based on minimal restrictions concerning the execution of the voting system. We aim for restrictions that can be enforced by public audits on the bulletin board, and do not require additional trust assumptions about the voting infrastructure. We consider the two restrictions from Figure 2, where Ψ_{tally1} requires that for each registered credential there has to be a corresponding ballot (possibly $\perp$) 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 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 $\{\!\!\{a_1, \ldots, a_n\}\!\!\}$, 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:

$$\begin{array}{rcl}
\mathsf{Ver}(\tau) & = & \{\!\!\{(\mathsf{cr}, \mathsf{v}) \mid \exists \mathsf{id}.\ \tau \vDash \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \ \wedge\ \Omega(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \ \wedge\ \mathsf{v} \neq \bot\}\!\!\}, \\
\mathsf{Ver}_{\mathsf{vote}}(\tau) & = & \{\!\!\{\mathsf{v} \mid \exists \mathsf{cr}.\ (\mathsf{cr}, \mathsf{v}) \in \mathsf{Ver}(\tau)\}\!\!\}, \\
\mathsf{Ver}_{\mathsf{cr}}(\tau) & = & \{\mathsf{cr} \mid \exists \mathsf{v}.\ (\mathsf{cr}, \mathsf{v}) \in \mathsf{Ver}(\tau)\}, \\
\mathsf{Adv}^{\diamond}(\tau) & = & \{\mathsf{cr} \mid \tau \vDash \Phi^{\diamond}_{\mathsf{adv}}(\mathsf{cr})\}, \\
\mathsf{Result}(\tau) & = & \{\!\!\{\mathsf{open}(\mathsf{b}) \mid \exists \mathsf{cr}.\ \tau \vDash \mathsf{BBtally}(\mathsf{cr}, \mathsf{b}) \ \wedge\ \mathsf{b} \neq \bot)\}\!\!\}.
\end{array}$$

Our definition of multiset-based end-to-end verifiability is a slight generalisation of the one in [26, 28]: we account for revoting; we extend the notion of corrupted voters (covering $\Phi^{\bullet}_{\mathsf{adv}}$) to adversarial credentials defined by $\Phi^{\diamond}_{\mathsf{adv}}$ (covering both $\Phi^{\bullet}_{\mathsf{adv}}$ and $\Phi^{\circ}_{\mathsf{adv}}$); we require that corrupted voters that have verified their votes should also benefit from the underlying end-to-end verifiability guarantees. In summary, the definition shows that the final result can be partitioned into: votes that have been verified by voters, votes that have not been verified, and additional votes that may be cast by the adversary. Moreover, for each of these sets, there are constraints related to their content and size.

Definition 3. Let Ω be a revoting policy. 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 $\mathsf{Result}(\tau) = V_1 \uplus V_2 \uplus V_3$, where

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

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. □

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

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

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

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

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 3. 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

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

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 3:

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 $(\mathsf{id}_1, \mathsf{cr}_1), \dots, (\mathsf{id}_\mathsf{s}, \mathsf{cr}_\mathsf{s})$ such that $\forall \mathsf{i} \in \{1, \dots, \mathsf{s}\},\ \tau \vDash \mathsf{Verified}(\mathsf{id}_\mathsf{i}, \mathsf{cr}_\mathsf{i}, \mathsf{v})$. 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 $\mathsf{i} \in \{1, \dots, \mathsf{s}\}$, from $\tau \vDash \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Psi_{\mathsf{tally1}}$ and $(\mathsf{cr}_\mathsf{i}, \mathsf{v}) \in \mathsf{Ver}(\tau)$, by Lemma 1, we deduce $\tau \vDash \mathsf{BBtally}(\mathsf{cr}_\mathsf{i}, \mathsf{b}_\mathsf{i}) \ \wedge\ \mathsf{v} = \mathsf{open}(\mathsf{b}_\mathsf{i})$. Therefore, by definition of R_1, we have $\{\!\{(\mathsf{cr}_1, \mathsf{v}), \dots, (\mathsf{cr}_\mathsf{s}, \mathsf{v})\}\!\} \subseteq R_1$. Moreover, from $\mathsf{i} \neq \mathsf{j} \Rightarrow \mathsf{cr}_\mathsf{i} \neq \mathsf{cr}_\mathsf{j}$, we can then deduce that all $(\mathsf{cr}_\mathsf{i}, \mathsf{v})$ 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 $R_2 = \{\!\{(\mathsf{cr}_1, \mathsf{v}_1), \dots, (\mathsf{cr}_\mathsf{l}, \mathsf{v}_\mathsf{l})\}\!\}$. From $\tau \vDash \Psi_{\mathsf{tally2}}$, we know that $\mathsf{cr}_1, \dots, \mathsf{cr}_\mathsf{l}$ are mutually distinct. By definition of R_2, for all $\mathsf{i} \in \{1, \dots, \mathsf{l}\}$, we have $\mathsf{cr}_\mathsf{i} \notin \mathsf{Ver}(\tau) \cup \mathsf{Adv}^{\diamond}(\tau)$ and there exists id_i such that $\tau \vDash \mathsf{Vote}(\mathsf{id}_\mathsf{i}, \mathsf{cr}_\mathsf{i}, \mathsf{v}_\mathsf{i})$.

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 $|V_3| < |\ \mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}_{\mathsf{cr}}(\tau)\ |$. □

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

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

5 Helios verification: attacks and security proofs

We put together the symbolic models introduced previously to derive several scenarios for the verification of election verifiability in Helios (in the next section we apply them to Belenios). We perform automated verification, finding both security proofs and attacks with Tamarin. The Tamarin code is available as additional material [1]. Each scenario is defined by assembling elements from Figure 1. The protocol component $\mathcal{P}_{\mathcal{H}}$ is combined with an individual verification procedure $\mathcal{V}_i$ and adversarial corruption rules $\mathcal{A}_j$, resulting in an e-voting specification $\mathcal{H}[\mathcal{V}_i, \mathcal{A}_j] = (\mathcal{P}_{\mathcal{H}}, \mathcal{V}_i, \mathcal{A}_j)$. The set of $\mathcal{V}_i$ and of $\mathcal{A}_j$ that we consider are in Table 1 and Table 2, respectively.

The procedure $\mathcal{V}_1$ represents the scenario where the bulletin board only displays the last ballot cast by each voter, as in the current deployment of Helios [5], or where all ballots are shown but voters are instructed to verify the last ballot cast for their credential on the bulletin board. $\mathcal{V}_2$ represents the same procedure, but performed after the voting phase has ended, so no more ballots can be accepted on the bulletin board. $\mathcal{V}_1$ and resp. $\mathcal{V}_2$ can be augmented to $\mathcal{V}_3$ and resp. $\mathcal{V}_4$, using an additional test, modelled by Ψ_{mine}, ensuring that all ballots cast on the bulletin board belong to the voter performing the verification (note that this is in fact not possible on the current deployment, where only the last ballot is shown). $\mathcal{V}_5$ and $\mathcal{V}_6$ are performed directly on $\mathsf{BBtally}$ after it is computed from BBcast.

Table 1: Verification procedures for Helios and Belenios

Specification	**Voter instructions**
$\mathcal{V}_1 : (\mathsf{R}^0_{\mathsf{ver}}, \Psi_{\mathsf{last}})$	verify the last ballot in BBcast
$\mathcal{V}_2 : (\mathsf{R}^1_{\mathsf{ver}}, \Psi_{\mathsf{last}})$	verify the last ballot in BBcast, and do it in tally phase
$\mathcal{V}_3 : (\mathsf{R}^0_{\mathsf{ver}}, \Psi_{\mathsf{last}} \wedge \Psi_{\mathsf{mine}})$	as $\mathcal{V}_1$, and ensure all ballots are recognised as own
$\mathcal{V}_4 : (\mathsf{R}^1_{\mathsf{ver}}, \Psi_{\mathsf{last}} \wedge \Psi_{\mathsf{mine}})$	as $\mathcal{V}_2$, and ensure all ballots are recognised as own
$\mathcal{V}_5 : (\mathsf{R}^2_{\mathsf{ver}})$	verify the ballot directly on $\mathsf{BBtally}$
$\mathcal{V}_6 : (\mathsf{R}^3_{\mathsf{ver}})$	verify abstention directly on $\mathsf{BBtally}$

Table 2: Adversary models for Helios

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

* : only the authentication component is corrupted
* : only the registration component is corrupted

Verification results are presented in Table 3, where $\mathcal{H}[\mathcal{V}_i, \mathcal{A}_j]$ is represented by $[\mathcal{V}_i, \mathcal{A}_j]$. Recall that

$$\begin{aligned}\Phi^{\bullet}_{\mathsf{E2E}} &= \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\bullet}_{\mathsf{res}},\\ \Phi^{\circ}_{\mathsf{E2E}} &= \Phi_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\circ}_{\mathsf{res}}.\end{aligned}$$

and note that for each adversary, except for $\mathcal{A}_5$ where we have the clash attack of [49], there is a corresponding procedure such that either $\Phi^{\bullet}_{\mathsf{E2E}}$ or $\Phi^{\circ}_{\mathsf{E2E}}$ is satisfied. A particular case is individual verifiability for the scenarios $[\mathcal{V}_i, \mathcal{A}_1]$, for $i \in \{1, 3\}$, that can only be proved for voters that are not corrupted (i.e. we use $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ instead of Φ_{iv}). Moreover, since $\mathcal{V}_1, \mathcal{V}_3$ allow a voter to verify multiple distinct votes, we also use the revote policy Ω^{last} to specify that only the last of these should be counted. Note that we can prove $\Phi^{\bullet}_{\mathsf{E2E}}$ for $\mathcal{V}_1, \mathcal{V}_3$ only in the weakest adversarial model $\mathcal{A}_1$. We also prove Φ_{cand} in all corruption scenarios, which is true since the auditors check the range proofs on the bulletin board to ensure legitimate votes. We have a '?' in the table when Tamarin does not terminate within 10 minutes; for all other cases it does so within 5 minutes. In several scenarios we find attacks that we discuss next. Some of them are expected; some of them are new.

Violations of Φ_{iv} ***- (VIV).*** In some cases, verification procedures do not satisfy the basic requirement of individual verifiability, i.e. that the verified vote should be part of the outcome. If revoting is allowed and a party responsible for vote casting, that is VS or VR, is corrupted, then any verification procedure performed during the voting phase is vulnerable to an instance of the ballot stuffing attack: even if the expected ballot is present on the bulletin board at some point during the voting phase, it may be later replaced by malicious parties with a ballot of their choice. The vulnerable scenarios are $\{[\mathcal{V}_i, \mathcal{A}_j] \mid i \in \{1,3\}, j \in \{2,4,5\}\}$. Some additional, less expected scenarios are also vulnerable: we have $\mathcal{H}[\mathcal{V}_i, \mathcal{A}_3] \not\models \Phi^{\mathsf{h}}_{\mathsf{iv}}$, for $i \in \{1, 3\}$.

Table 3: Verifiability analysis for $\mathcal{H}[\mathcal{V}_i, \mathcal{A}_j]$

$[\mathcal{V}_\mathbf{i}, \mathcal{A}_\mathbf{j}]/\mathbf{\Phi}_{\mathsf{type}}$	Φ_{iv}	Φ_{eli}	Φ_{cl}	$\Phi^{\bullet}_{\mathsf{res}}$	$\Phi^{\circ}_{\mathsf{res}}$
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{1,3\}$	✓⋆	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{2,4,5,6\}$	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{1,3\}$	✗	✓	✓	✗	✗
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{2,4,5,6\}$	✓	✓	✓	✗	✓
$[\mathcal{V}_1, \mathcal{A}_3]$	✗	✓	✗	✓	✓
$[\mathcal{V}_3, \mathcal{A}_3]$	✗	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_3],\ i \in \{2,4,5\}$	✓	✓	✓	✓	✓
$[\mathcal{V}_6, \mathcal{A}_3]$	✓	✓	✗	✓	✓
$[\mathcal{V}_1, \mathcal{A}_4]$	✗	✓	✗	✗	✗
$[\mathcal{V}_3, \mathcal{A}_4]$	✗	✓	✓	✗	✗
$[\mathcal{V}_i, \mathcal{A}_4],\ i \in \{2,4,5\}$	✓	✓	✓	✗	✓
$[\mathcal{V}_6, \mathcal{A}_4]$	✓	✓	✗	✗	✓
$[\mathcal{V}_1, \mathcal{A}_5]$	✗	✓	✗	✗	✗
$[\mathcal{V}_3, \mathcal{A}_5]$	✗	✓	?*	✗	✗
$[\mathcal{V}_4, \mathcal{A}_5]$	✓	✓	?*	✗	✓
$[\mathcal{V}_i, \mathcal{A}_5],\ i \in \{2,5,6\}$	✓	✓	✗	✗	✓

⋆ : verification is for $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ when used with revote policy Ω^{last}.
* : there should be an attack but Tamarin takes > 10mins.

In this case, the adversary only controls the registration authority. The explanation for the attack in this case is that there can be a clash on public credentials, so the ballot of a voter who verified may be replaced with a ballot of a second voter that has the same credential. Note that, if the second voter verifies their ballot according to $\mathcal{V}_3$, the clash mounted by the adversary would be detected, and that is why we have $\mathcal{H}[\mathcal{V}_3, \mathcal{A}_3] \models \Phi_{\mathsf{cl}}$.

Violations of $\Phi^{\bullet}_{\mathsf{res}}$ - classic ballot stuffing (BS1). Ballot stuffing is possible in Helios for honest voters who did not vote or, more generally, for those who did not verify their vote. For this attack, the adversary has to corrupt either the registrar or the voting server. Since the attack applies for voters who did not verify their vote, no verification procedure can help here, so we obtain the following vulnerable scenarios: $\{[\mathcal{V}_i, \mathcal{A}_j] \mid i \in \{1, \ldots, 6\}, j \in \{2,4,5\}\}$.

Violations of $\Phi^{\circ}_{\mathsf{res}}$ - strong ballot stuffing (BS2). The goal of $\Phi^{\circ}_{\mathsf{res}}$ is to formally prove the claim from above, i.e. that there can be no ballot stuffing for voters that verified their (absence of) vote. This is true for $\mathcal{V}_2, \mathcal{V}_4, \mathcal{V}_5, \mathcal{V}_6$, but not for $\mathcal{V}_1, \mathcal{V}_3$.

Violations of Φ_{cl} - clash attack of [49] (CL1). The clash attack of [49] occurs with the adversarial model $\mathcal{A}_5$, and none of the current verification procedures can prevent it.

Violations of Φ_{cl} - new clash attacks (CL2). To derive the new class of clash attacks discussed in subsection 2.2, we consider the weaker adversaries $\mathcal{A}_3$ - corrupting the registration of credentials, and $\mathcal{A}_4$ - also corrupting a party responsible for ballot casting. Assume voters perform verification according to $\mathcal{V}_1$, i.e. the standard procedure, and that the corrupted registrar assigns the same credential to a second voter. Then both voters may be able to successfully verify their ballot, since the bulletin board will display the latest ballot cast by each of them, and verifying at different times they can both be happy. Yet, for these two ballots, there will only be one vote counted at the end of election. This attack shows that the adversary $\mathcal{A}_3$ can cause damage already. However, it may not allow large scale corruption of results, since too many instances of this attack would have a noticeable impact on the total number of votes in the outcome. For large scale corruption, the adversary needs to control a ballot casting authority - either VS or VR, leading to $\mathcal{A}_4$ - so that the missing votes can be replaced with the ones chosen by the adversary. The vulnerable scenarios, excluding the ones also vulnerable to ***(CL1)***, are $\{[\mathcal{V}_i, \mathcal{A}_j]) \mid i \in \{1, 6\}, j \in \{3, 4\}\}$.

This attack can be prevented by $\mathcal{V}_5$ - verifying ballots directly on $\mathsf{BBtally}$ (where there is only one single ballot per cr) - or using the restriction Ψ_{mine} to ensure all ballots are recognised as theirs by voters. Note that the violation of Φ_{cl} for $\mathcal{V}_6$ corresponds to the clash attack on abstaining voters. Ψ_{mine} does not apply in that case. Verifying on $\mathsf{BBtally}$ also does not help, since two abstaining voters would both be happy with the $\perp$ ballot.

5.1 Resisting clash attacks with verifiable aliases

We have seen the none of the current procedures can protect against clash attacks by the strongest adversary $\mathcal{A}_5$, or against weaker adversaries that target abstaining voters. The solution proposed by [49] to counter the former attacks is that the randomness used for encryption should be a combination of one part r^{V} that is provided by the voter and another part r^{VP} that is generated by the voting platform. In this case, even if VP is corrupted, it will not be possible that two voters will have the same ballot. However, our new class of clash attacks can still be mounted without a corrupted VP, and having control of encryption randomness would not prevent them. Another solution suggested by [49] is that voters should have an efficient strategy for auditing a number of ballots and check whether two ballots are generated similarly. However, it is in general difficult to anticipate and counteract the adversarial strategy of the platform, and this solution also raises usability challenges.

Our observation is that the source of attacks (by all $\mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5$) is the distribution of corrupted credentials by the registrar, which provides a number of voters with the same alias, and there is no direct way to detect it. To improve on this, we propose a solution which allows voters to verify that their alias is unique to them, so that clashes can be detected independently of ballot casting. We believe this solution is potentially more usable than some of the options sketched above, and it could also be improved with future research, for example by making the aliases publicly verifiable, not just voter-verifiable. We also note that some of the more complex checks that we proposed to protect against $\mathcal{A}_3, \mathcal{A}_4$, like Ψ_{mine}, are also not needed: a simple individual verification procedure combined with alias verification would be enough to prevent clash attacks. Finally, verifiable aliases would also prevent clash attacks against abstaining voters.

Figure 3: Verifiable aliases for Helios

$\mathsf{R}^{\mathsf{VR}}_{\mathsf{reg}}$: **register voter with verifiable public credential**

$\mathsf{let}\ \ \mathsf{cr} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd} \rangle)\ \ \mathsf{in}$

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

$\mathsf{R}^{\mathsf{V}}_{\mathsf{cr}}$: **verify that** cr **satisfies the expected relation**

$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{pwd})\ ]\!\!-\!\![\ \mathsf{cr} \approx \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd} \rangle)\ ]\!\!\rightarrow [\ \mathsf{VerC}(\mathsf{id}, \mathsf{cr})\ ]$

$\mathsf{R}^{0}_{\mathsf{ver}}$: **verification succeeds only if** cr **has been previously verified**

$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{hc}, \mathsf{c} \rangle\ \ \mathsf{in}$

$[\ \mathsf{VerC}(\mathsf{id}, \mathsf{cr}), \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{hash}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!-\!\![\ \mathsf{hash} \approx \mathsf{hc}, \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow [\ \ ]$

Verifiable aliases can be provided by a function that generates distinct outputs for distinct inputs. In this paper, we evaluate a simple solution based on a (colision resistant) hash function h which takes the unique identity of the voter id and an additional value that the voter knows, e.g. the password of this voter pwd, and generates a unique alias $\mathsf{h}(\langle \mathsf{id}, \mathsf{pwd} \rangle)$. This alias will be verifiable by each voter, as modelled by the additional rule $\mathsf{R}^{\mathsf{V}}_{\mathsf{cr}}$ given in Figure 3. We also note that individual verification procedures need to be updated so that they require alias verification in order to succeed. We give one example. Formal verification results for all cases are in Table 4.

Table 4: Verifiability analysis for verifiable aliases

$[\mathcal{V}_\mathbf{i}, \mathcal{A}_\mathbf{j}]/\mathbf{\Phi}_{\mathsf{type}}$	Φ_{iv}	Φ_{eli}	Φ_{cl}	$\Phi^{\bullet}_{\mathsf{res}}$	$\Phi^{\circ}_{\mathsf{res}}$
$[\mathcal{V}_1, \mathcal{A}_3]$	✗	✓	✓	✓	✓
$[\mathcal{V}_6, \mathcal{A}_3]$	✓	✓	✓	✓	✓
$[\mathcal{V}_1, \mathcal{A}_4]$	✗	✓	✓	✗	✗
$[\mathcal{V}_6, \mathcal{A}_4]$	✓	✓	✓	✗	✓
$[\mathcal{V}_i, \mathcal{A}_5],\ i \in \{1,3\}$	✗	✓	✓	✗	✗
$[\mathcal{V}_4, \mathcal{A}_5]$	✓	✓	✓	✗	✓
$[\mathcal{V}_i, \mathcal{A}_5],\ i \in \{2,5,6\}$	✓	✓	✓	✗	✓

6 Belenios verification: security proofs and attacks

The Belenios voting protocol is an elaboration of Helios in order to ensure stronger security properties when some election authorities may be corrupted [33, 26]. Specifically, the aim is to distribute trust, so that a stronger version of E2E ($\Phi^{\bullet}_{\mathsf{E2E}}$ in our models) can be achieved even when either the registrar or the server is corrupted. The most important change with respect to Helios is the addition of signatures that can publicly authenticate voter ballots, in addition to the id-password mechanism provided by the voting server.

Cryptographic primitives. The symbolic model needs to be extended to capture the additional cryptographic primitives of signing and zero-knowledge proofs. Belenios uses a zero-knowledge proof relation that is more complex than in Helios. It has two features: (i) like in Helios, it ensures the vote is in a valid range; and (ii) it ensures the encryption is linked to a given signature verification key, and so it cannot be reused to cast a ballot for a different voter. We model (i) as in Helios, with equations that we add to the theory of encryption. To model (ii), we have a dedicated proof symbol, as shown below:

$$\begin{aligned}
&& \mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) &= \mathsf{x}, \\
(\forall i) && \mathsf{ver}(\mathsf{enc}(\mathsf{x_i}, \mathsf{y}, \mathsf{z}), \mathsf{y}, \langle \mathsf{x_1}, \ldots, \mathsf{x_k} \rangle) &= \mathsf{ok}, \\
&& \mathsf{verify}(\mathsf{sign}(\mathsf{m}, \mathsf{k}), \mathsf{m}, \mathsf{pk}(\mathsf{k})) &= \mathsf{ok}, \\
&& \mathsf{verifypr}(\mathsf{proof}(\mathsf{v}, \mathsf{r}, \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r}), \mathsf{pkey}, \mathsf{vk}), \mathsf{pkey}, \mathsf{vk}) &= \mathsf{ok}.
\end{aligned}$$

Protocol description. The registrar and the voting server have special roles:

- *Registrar (*VR*):* generates a signing key pair for each eligible voter - a signing key skey and a verification key vkey - and distributes it to the voter, while publishing the verification key vkey on the bulletin board.
- *Voting Server (*VS*):* generates a password for each voter to authenticate them when they are casting a ballot and after authentication publishes the received ballot on the bulletin board. At ballot casting time, it also records a log of pairs $(\mathsf{id}, \mathsf{cr})$, in order to ensure consistency, i.e. that no voter uses two different credentials, and that no credential is used by two different voters.

The remaining parties: administrator (A), voters (V), voting platform (VP), trustee (T) and election auditors (EA) are similar to Helios. The phases of the protocol and the role of parties are as follows:

Setup phase. A determines the list of eligible voters $\mathsf{id_1}, \ldots, \mathsf{id_n}$, and sends it to VR and VS. They choose corresponding signature key pairs, public identities, passwords, and communicate these to voters. 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, where $\mathsf{cr_i} = \mathsf{vkey_i}$. Each voter $\mathsf{id_i}$ obtains the signature key pair $(\mathsf{id_i}, \mathsf{cr_i}, \mathsf{skey_i})$ from VR and credential $(\mathsf{id_i}, \mathsf{pwd_i})$ from VS after the registration.

Voting phase. In this phase, voters interact with their voting platform VP to construct a ballot, authenticate and post it to VS, which casts it on BB. A ballot is simply an encryption of the vote along with the signature of the voter, and the zero-knowledge proof. It is important to note that the voters communicate the pair $(\mathsf{id}, \mathsf{cr})$ to the server at this point. This is the first time it learns the association of id and cr. We have:

$$\begin{array}{lll}
\mathsf{VP} & : & \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r}); \quad \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{skey}); \quad \mathsf{pr} = \mathsf{proof}(\mathsf{v}, \mathsf{r}, \mathsf{c}, \mathsf{pk}, \mathsf{cr}); \\
 & & \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle; \quad \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b} \rangle), \\
\mathsf{VS} & : & \text{receive } \langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle, \text{ match it with } (\mathsf{id}, \mathsf{pwd}), \text{ verify zero-knowledge proof } \mathsf{pr} \\
 & & \text{and verify the signature using } \mathsf{cr}; \text{ also ensure } (\mathsf{id}, \mathsf{cr}) \text{ is consistent with the log} \\
\mathsf{BBcast} & : & (\mathsf{cr}, \mathsf{b}), \text{ where } \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle.
\end{array}$$

Tally phase. As in the homomorphic variant of Helios, decryption and vote counting is performed by T after ciphertexts from valid ballots are combined homomorphically.

Protocol specification. The protocol specification $\mathcal{P}_{\mathcal{B}}$ of Belenios is similar to the specification $\mathcal{P}_{\mathcal{H}}$ of Helios, following the differences sketched above: the ballot structure, the registrar and the server behaviour. It is depicted in Figure 4. For applying the security definition, we also need to define the predicate open according to the ballot structure in Belenios. If pk is the public key of the election, we have:

$$\begin{array}{rcl}
\mathsf{b} = \langle \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r}), \mathsf{s}, \mathsf{pr} \rangle & \Longrightarrow & \mathsf{open}(\mathsf{b}) = \mathsf{v}, \\
\mathsf{b} = \bot & \Longrightarrow & \mathsf{open}(\mathsf{b}) = \bot.
\end{array}$$

Individual verification procedures $\mathcal{V}$. The rules and restrictions to obtain individual verification procedures for Belenios are similar to the ones for Helios, and are presented in Figure 4b. To obtain specific procedures $\mathcal{V}_i$, we combine them as in Table 1.

Adversary models. The corruption rules allow the adversary to control voters, trustees, the server or the registrar - as given in Figure 4c. The rules $\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}$ leak all the private information from trustees and voters to the adversary. To model corrupted election infrastructure, we consider the roles of each party and give complete control of the corresponding process to the adversary. The role of the registrar is to assign and publish voter signing keys, and the rule $\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}$ allows the adversary full control of that process: it takes the signature key pair from the adversary to distribute to the voters; it may, in particular, lead to clash attacks if the same signature key pair is allocated to two different voters. The rule $\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$, modelling a corrupted server, adds any ballot to the bulletin board without password authentication or verifying logs. We perform verification for the adversarial models listed in Table 5: all-honest case ($\mathcal{A}_1$), corrupted server case ($\mathcal{A}_2$), corrupted registrar case ($\mathcal{A}_3$), and all-corrupted case ($\mathcal{A}_4$). To obtain an e-voting specification, we combine $\mathcal{P}_{\mathcal{B}}$ with an adversary $\mathcal{A}_i$ and an individual verification procedure $\mathcal{V}_j$.

Table 5: Adversary models for Belenios

Specification	**Corrupted parties and infrastructure**
$\mathcal{A}_1 : \mathcal{C}^{\mathsf{T}}_{\mathsf{key}}, \mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}, \Psi_{\mathsf{order}}$	trustees, 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, server and registrar

Attacks. Verification results for these models, obtained with Tamarin, are presented in Table 6. We find some unexpected attacks in cases $\mathcal{A}_1$ and $\mathcal{A}_3$ that we explain next. In case of $\mathcal{A}_1$, we expect weak individual verifiability to hold with revote policy Ω^{last}, as in the case of Helios. However, we find $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_1, \mathcal{A}_1) \not\models \Phi^{\mathsf{h}}_{\mathsf{iv}}$. In case of $\mathcal{A}_3$, the server is honest and it checks the consistency of the assignment of signing keys to voters, as modelled by the restriction $\Psi^{\mathsf{VS}}_{\mathsf{log}}$. That is why clash attacks should not be possible. Ballot stuffing should also

Figure 4: Protocol components for Belenios specification

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

SETUP PHASE

$\mathsf{R}^{\mathsf{T}}_{\mathsf{key}}$: **generate election secret and public keys**

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

$\mathsf{R}^{\mathsf{A}}_{\mathsf{cand}}$: **determine candidates to be elected**

$\mathsf{let}\ \ \mathsf{vlist} = \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle\ \ \mathsf{in}$

$[\ \mathsf{In}(\mathsf{vlist})\ ]\!\!-\!\![\ \mathsf{Cand}(\mathsf{v}_1), \ldots, \mathsf{Cand}(\mathsf{v}_k), \mathsf{BBcand}(\mathsf{vlist})\ ]\!\!\rightarrow$
$[\ \mathsf{Cand}(\mathsf{v}_1), \ldots, \mathsf{Cand}(\mathsf{v}_k), \mathsf{Out}(\mathsf{vlist})\ ]$

$\mathsf{R}^{\mathsf{A}}_{\mathsf{id}}$: **determine identities eligible to vote**

$[\ \mathsf{In}(\mathsf{id})\ ]\!\!-\!\![\ ]\!\!\rightarrow[\ \mathsf{Id}(\mathsf{id})\ ]$

$\mathsf{R}^{\mathsf{VR}}_{\mathsf{reg}}$: **register voter with signature pair**

$\mathsf{let}\ \mathsf{cr} = \mathsf{pk}(\mathsf{skey})\ \ \mathsf{in}$

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

$\mathsf{R}^{\mathsf{VS}}_{\mathsf{cred}}$: **generate password for voter authentication**

$[\ \mathsf{Id}(\mathsf{id}), \mathsf{Fr}(\mathsf{pwd})\ ]\!\!-\!\![\ ]\!\!\rightarrow[\ \mathsf{Cred}(\mathsf{id}, \mathsf{pwd})\ ]$

$\mathsf{R}^{\mathsf{VS}}_{\mathsf{bb}}$: **setup initial** BBcast **for registered voters**

$[\ \mathsf{BBreg}(\mathsf{cr})\ ]\!\!-\!\![\ \mathsf{BBcast}(\mathsf{cr}, \perp)\ ]\!\!\rightarrow[\ \mathsf{BBcast}(\mathsf{cr}, \perp)\ ]$

VOTING PHASE

$\mathsf{R}^{\mathsf{VP}}_{\mathsf{vote}}$: **construct a ballot, authenticate and send it to** VS

$\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{skey});$
$\quad \mathsf{pr} = \mathsf{proof}(\mathsf{v}, \mathsf{r}, \mathsf{c}, \mathsf{pk}, \mathsf{cr});$
$\quad \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b} \rangle)\ \ \mathsf{in}$

$[\ \mathsf{Cand}(\mathsf{v}), \mathsf{BBkey}(\mathsf{pkey}), \mathsf{Fr}(\mathsf{r}), \mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{skey}),$
$\ \mathsf{Cred}(\mathsf{id}, \mathsf{pwd})\ ]\!\!-\!\![\ \mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow$
$[\ \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b})\ ]$

$\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$: **authenticate voter, verify ballot, publish**

$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b} \rangle)\ \ \mathsf{in}$

$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a} \rangle), \mathsf{Cred}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBreg}(\mathsf{cr})\ ]$
$-\!\![\ \mathsf{a}' \approx \mathsf{a}, \mathsf{Log}(\mathsf{id}, \mathsf{cr}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow[\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$

$\Psi^{\mathsf{VS}}_{\mathsf{log}}$: **logs are checked to ensure consistency**

$\mathsf{Log}(\mathsf{id}, \mathsf{cr})\ @\ i \Rightarrow \neg(\mathsf{Log}(\mathsf{id}, \mathsf{cr}')\ @\ j\ \wedge\ \mathsf{cr} \neq \mathsf{cr}') \wedge$
$\neg(\mathsf{Log}(\mathsf{id}', \mathsf{cr})\ @\ j\ \wedge\ \mathsf{id} \neq \mathsf{id}')$

$\Psi^{\mathsf{VS/EA}}_{\mathsf{cast}}$: **ensure ballot validity; can be audited by** EA

$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \Rightarrow \mathsf{BBreg}(\mathsf{cr})\ \wedge\ (\ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle \Rightarrow$
$\mathsf{BBkey}(\mathsf{pkey})\ \wedge\ \mathsf{BBcand}(\mathsf{vlist})\ \wedge\ \mathsf{verify}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) = \mathsf{ok}$
$\wedge\ \mathsf{ver}(\mathsf{c}, \mathsf{pkey}, \mathsf{vlist}) = \mathsf{ok}\ \wedge\ \mathsf{verifypr}(\mathsf{pr}, \mathsf{pkey}, \mathsf{cr}) = \mathsf{ok}\ )$

TALLY PHASE

$\mathsf{R}^{\mathsf{VS/EA}}_{\mathsf{tally}}$: VS **selects ballots for tally; can be audited by** EA

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

$\Psi^{\mathsf{VS/EA}}_{\mathsf{tally}}$: **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 $\mathcal{P}_{\mathcal{B}}$

$\mathsf{R}^{0}_{\mathsf{ver}}$: **voter verifies the receipt on** BBcast

$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle\ \ \mathsf{in}$

$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]$
$-\!\![\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow[\ \ ]$

$\mathsf{R}^{1}_{\mathsf{ver}}$: **same as** $\mathsf{R}^{0}_{\mathsf{ver}}$**, but performed in the tally phase**

$\mathsf{R}^{2}_{\mathsf{ver}}$: **voter verifies the receipt on** BBtally

$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle\ \ \mathsf{in}$

$[\ \mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{BBtally}(\mathsf{cr}, \mathsf{b})\ ]$
$-\!\![\ \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})\ ]\!\!\rightarrow[\ \ ]$

$\mathsf{R}^{3}_{\mathsf{ver}}$: **voter verifies there is no ballot on** BBtally

$[\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{skey}), \mathsf{BBtally}(\mathsf{cr}, \mathsf{x})\ ]$
$-\!\![\ \mathsf{x} \approx \perp, \mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \perp)\ ]\!\!\rightarrow[\ \ ]$

$\mathsf{R}^{0}_{\mathsf{ver}}$ **and** $\mathsf{R}^{1}_{\mathsf{ver}}$ **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

$\mathsf{VerB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @\ j\ \wedge\ \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @\ i\ \wedge\ i \prec j$
$\Rightarrow \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @\ l$

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

$\mathcal{C}^{\mathsf{T}}_{\mathsf{key}}$: **corrupt trustee to reveal the secret key**

$[\ \mathsf{Sk}(\mathsf{sk})\ ]\!\!-\!\![\ ]\!\!\rightarrow[\ \mathsf{Out}(\mathsf{sk})\ ]$

$\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}$: **corrupt registration of signature pair**

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

$\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$: **corrupt server to stuff ballots**

$\mathsf{let}\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr} \rangle\ \ \mathsf{in}$

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

$\mathcal{C}^{\mathsf{V}}_{\mathsf{corr}}$: **corrupt voter to reveal credentials**

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

Ψ_{order} : **ensure ballots are delivered in the right order**

$\mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ @\ i\ \wedge\ \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b}')\ @\ j\ \wedge$
$\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ @\ k \wedge \mathsf{BBcast}(\mathsf{cr}, \mathsf{b}')\ @\ l\ \wedge$
$i \prec j \Rightarrow k \prec l$

not be possible, since it is both a valid signature and the password that are required by the server. However, we find results contradicting these properties: $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_3) \not\models \Phi_{\mathsf{cl}}$ for $i = 1, 3$ and $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_3) \not\models \Phi^{\bullet}_{\mathsf{res}}$, for any i. Moreover, the weaker property $\Phi^{\circ}_{\mathsf{res}}$ is also not satisfied for $i \in \{1, 3\}$. In the following, for illustrating attacks, we consider a scenario with two voters id_1 and id_2. The main source of attacks is that revoting is allowed and that voters can choose the public credential under which they submit ballots.

Table 6: Verifiability analysis of Belenios

$[\mathcal{V}_\mathbf{i}, \mathcal{A}_\mathbf{j}]/\mathbf{\Phi}_{\mathsf{type}}$	Φ_{iv}	Φ_{eli}	Φ_{cl}	$\Phi^{\bullet}_{\mathsf{res}}$	$\Phi^{\circ}_{\mathsf{res}}$
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{1,3\}$	✗*	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_1],\ i \in \{2,4,5,6\}$	✓	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{1,3\}$	✗	✓	✓	✓	✓
$[\mathcal{V}_i, \mathcal{A}_2],\ i \in \{2,4,5,6\}$	✓	✓	✓	✓	✓
$[\mathcal{V}_1, \mathcal{A}_3]$	✗	✓	✗	✗	✗
$[\mathcal{V}_3, \mathcal{A}_3]$	?*	✓	✓	✗	✗
$[\mathcal{V}_i, \mathcal{A}_3],\ i \in \{2,4,5\}$	✓	✓	✓	✗	✓
$[\mathcal{V}_6, \mathcal{A}_3]$	✓	✓	✗	✗	✓
$[\mathcal{V}_1, \mathcal{A}_4]$	✗	✓	✗	✗	✗
$[\mathcal{V}_3, \mathcal{A}_4]$	✗	✓	✓	✗	✗
$[\mathcal{V}_i, \mathcal{A}_4],\ i \in \{2,4,5\}$	✓	✓	✓	✗	✓
$[\mathcal{V}_6, \mathcal{A}_4]$	✓	✓	✗	✗	✓

$\star$: $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ is also violated.
$*$: Tamarin execution exceeds system capacities.

Ballot reordering - violation of $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ by $\mathcal{A}_1$. Assume id_1 has public credential cr_1 and id_2 has public credential cr_2. The adversary corrupts id_2 and controls the communication network. The attack scenario is as follows:

Voter id_1:	posts a ballot b_1 followed by b_2. Subsequently, the voter only verifies b_2.
Adversary $\mathcal{A}$:	blocks both ballots from id_1 and posts the ballot b_2 using credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
Server VS:	accepts adversary's input since it has no way of verifying that b_2 was constructed by id_1. We obtain $\mathsf{BBcast}(\mathsf{cr}_1, \mathsf{b}_2)$, and thus the voter id_1 can successfully verify the ballot b_2.
Adversary $\mathcal{A}$:	posts the second ballot b_1 using credentials $(\mathsf{id}_2, \mathsf{cr}_1)$.
Server VS:	accepts the ballot b_1, and we have $\mathsf{BBcast}(\mathsf{cr}_1, \mathsf{b}_1)$.
Outcome:	we obtain $\mathsf{BBtally}(\mathsf{cr}_1, \mathsf{b}_1)$, when we expect to have $\mathsf{BBtally}(\mathsf{cr}_1, \mathsf{b}_2)$.

We note that this attack is not possible in Helios since the registrar and the server agree on the correspondence between identities and credentials, and therefore corrupt voters cannot cast ballots for credentials that are not assigned to them.

Ballot stuffing - violation of $\Phi^{\bullet}_{\mathsf{res}}$ by $\mathcal{A}_3$. Assume id_1 has public credential cr_1 and id_2 has public credential cr_2. The adversary corrupts id_2 and controls the communication network. The attack scenario is as follows:

Registrar VR:	registers voters normally, resulting in $\mathsf{Reg}(\mathsf{id}_1, \mathsf{cr}_1, \mathsf{skey}_1)$ and $\mathsf{Reg}(\mathsf{id}_2, \mathsf{cr}_2, \mathsf{skey}_2)$.
Server VS:	registers voters normally, resulting in $\mathsf{Cred}(\mathsf{id}_1, \mathsf{pwd}_1)$ and $\mathsf{Cred}(\mathsf{id}_2, \mathsf{pwd}_2)$.
Adversary $\mathcal{A}$:	corrupts the registrar and the voter id_2; it obtains $\mathsf{cr}_1, \mathsf{skey}_1, \mathsf{pwd}_2$. It casts a ballot b associated to credentials $(\mathsf{id}_2, \mathsf{cr}_1)$. We have $\mathsf{BBcast}(\mathsf{cr}_1, \mathsf{b})$.
Server VS:	based on log checks, it will not accept further ballots with credentials $(\mathsf{id}_1, \mathsf{cr}_1)$.

In this way, the honest voter id_1 can be prevented by corrupt parties (registrar and voter) from casting a ballot. Formally, the formula $\Phi^{\bullet}_{\mathsf{res}}$ is not satisfied since we obtain $\mathsf{BBtally}(\mathsf{cr}_1, \mathsf{b})$ and cr_1 does not correspond to a corrupted voter and b does not come from an honest execution of the voting procedure. We note that, formally, this is not an attack on multiset-based verifiability as specified in Definition 3 or in [26, 28]. This is because the number of adversarial votes in the final result is bounded by the number of corrupted voters, and the multiset-based definition allows ballot-dropping for honest voters. However, we think this attack is more serious than mere ballot-dropping and should be considered a real attack on election integrity; the honest voter is permanently prevented from casting a ballot, even if completely trusted infrastructure is used. Our analysis for the other cases shows that a stronger property is possible, which could be aimed for in this case as well.

Ballot stuffing - violation of $\Phi^{\circ}_{\mathsf{res}}$ ***by*** $\mathcal{A}_3$. We also have $(\mathcal{P}_{\mathcal{B}}, \mathcal{V}_i, \mathcal{A}_3) \not\models \Phi^{\circ}_{\mathsf{res}}$ for $i = 1, 3$. Indeed, in a variation of the previous attack, assume that the voter id_1 casts a ballot b_1, which is intercepted by $\mathcal{A}$ and cast under credentials $(\mathsf{id}_2, \mathsf{cr}_1)$. Subsequently, the voter id_1 can successfully verify b_1 if we rely on $\mathcal{V}_1$ or $\mathcal{V}_3$. However, at some point later, $\mathcal{A}$ can submit an adversarial ballot b_2 with credentials $(\mathsf{id}_2, \mathsf{cr}_1)$, without being detected by the log checks on the server. This violates $\Phi^{\circ}_{\mathsf{res}}$, since the adversary managed to submit a ballot even if the voter is not corrupted and performed successful verification. We note that, in this case, the attack does violate multiset-based verifiability as specified in Definition 3 or in [26, 28], since a verified vote disappears from the final tally.

Clash attack - violation of Φ_{cl} ***by*** $\mathcal{A}_3$***.*** Similar to individual verifiability, we require resistance to clash attacks even if some of the targeted voters are corrupted. We show a clash attack against two voters, where one is honest and one is corrupted:

Registrar VR:	clashes id_1 and id_2 at registration, resulting in $\mathsf{Reg}(\mathsf{id}_1, \mathsf{cr}, \mathsf{skey})$ and $\mathsf{Reg}(\mathsf{id}_2, \mathsf{cr}, \mathsf{skey})$.
Voters:	the voter id_1 posts b_1 and the voter id_2 posts b_2.
Adversary $\mathcal{A}$:	blocks b_1 from id_1, corrupts id_2 and submits the ballot b_1 with credentials $(\mathsf{id}_2, \mathsf{cr})$.
Server VS:	sees submissions $(\mathsf{id}_2, \mathsf{cr}, \mathsf{b}_1), (\mathsf{id}_2, \mathsf{cr}, \mathsf{b}_2)$, showing no inconsistency in logs.
Voters:	id_1 sees $\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}_1)$; id_2 sees $\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}_2)$; verification is successful for both.
Outcome:	only one ballot will be tallied for id_1 and id_2; adversary can determine which.

We note that this attack is possible even if $\mathcal{A}$ does not know the secret keys of voters. It only needs to cause a clash on verification keys.

Other attacks. Reminiscent from Helios, we find the clash attack on empty ballots, i.e. against $\mathcal{V}_6$ when the registrar is corrupted by $\mathcal{A}_3$ and $\mathcal{A}_4$. We also have the generic attacks against individual verifiability of $\mathcal{V}_1, \mathcal{V}_3$ that are related to the fact that revoting is allowed and ballots can be added by a corrupted server or registrar after the voters have verified. If there are no corrupted voters, these attacks are less serious than in Helios, since the only ballots corrupted parties can add are the ones constructed by the respective voters, similar to the reordering attack when election authorities are honest. We also find the expected attacks in the case of $\mathcal{A}_4$ when both the server and the registrar are corrupted.

Security proofs. We derive proofs of strong end-to-end verifiability, that is $\Phi^{\bullet}_{\mathsf{E2E}}$, for the two scenarios where the registrar is honest, i.e. $\mathcal{A}_1$ and $\mathcal{A}_2$. For $\mathcal{A}_3$, when the server is honest and the registrar is corrupt, the verification of logs could potentially be improved so that the same property holds, which is expected in Belenios. Currently, we can only prove the weaker notion of end-to-end verifiability, that is $\Phi^{\circ}_{\mathsf{E2E}}$, for this case. Finally, for $\mathcal{A}_4$, when both the server and the registrar are corrupted, Belenios satisfies the weaker notion of end-to-end verifiability, like Helios.

7 Conclusion and future work

A limitation of our definition is that we cannot handle protocols where there are no public credentials associated to ballots on the bulletin board. This excludes for example a version of Helios with detached names that is mentioned in [49]. We need public credentials in order to make a strong (injective) correspondence between tallied ballots and earlier events in the trace. A way towards lifting this restriction may be to rely on injective correspondence assertions, which are supported by ProVerif, but not by Tamarin to our knowledge. A more general notion of opening for a ballot is also needed, to handle systems that offer everlasting privacy, where one cannot extract the votes from the public bulletin board [51, 36]. For Helios or Belenios to be deployed in elections with even higher stakes than currently, we need to further reduce the trust assumptions required for (strong) end-to-end verifiability - since attackers would be even more motivated to break them. We also need to improve the usability of some proposed solutions, like our proposal for verifiable aliases.

References

[1] *Additional material: Tamarin code for verification*. `https://www.dropbox.com/sh/xpvqun0k9ruuu07/AAAGBijof03WA9jugu_3XIrxa?dl=0`.

[2] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In John Launchbury and John C. Mitchell, editors, *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*, pages 33–44. ACM, 2002.

[3] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 65(1):1:1–1:41, 2018.

[4] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 20(3):395, 2007.

[5] Ben Adida. The Helios Voting System. `https://heliosvoting.org/`.

[6] Ben Adida. Helios: Web-based open-audit voting. In *17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348, 2008.

[7] Ben Adida, Olivier De Marneffe, Olivier Pereira, and Jean-Jacques 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.

[8] Ben Adida and C. Andrew Neff. Ballot casting assurance. In *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06*, 2006.

[9] José Bacelar Almeida, Manuel Barbosa, Endre Bangerter, Gilles Barthe, Stephan Krenn, and Santiago Zanella Béguelin. Full proof cryptography: verifiable compilation of efficient zero-knowledge protocols. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 488–500. ACM, 2012.

[10] Michael Backes, Cătălin Hrițcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, Pittsburgh, Pennsylvania, USA, 23-25 June 2008*, pages 195–209. IEEE Computer Society, 2008.

[11] Michael Backes, Cătălin Hrițcu, and Matteo Maffei. Type-checking zero-knowledge. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 357–370. ACM, 2008.

[12] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 202–215. IEEE Computer Society, 2008.

[13] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[14] Gilles Barthe, Benjamin Grégoire, Charlie Jacomme, Steve Kremer, and Pierre-Yves Strub. Symbolic methods in computational cryptography proofs. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 136–151, 2019.

[15] David A. Basin, Sasa Radomirovic, and Lara Schmid. Dispute resolution in voting. *CoRR*, abs/2005.03749, 2020.

[16] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.

[17] Josh Benaloh. Simple verifiable elections. In Dan S. Wallach and Ronald L. Rivest, editors, *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*. USENIX Association, 2006.

[18] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07*, 2007.

[19] Josh Daniel Cohen Benaloh. *Verifiable Secret-ballot Elections*. PhD thesis, Yale University, New Haven, CT, USA, 1987. AAI8809191.

[20] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 499–516. IEEE Computer Society, 2015.

[21] Matthew Bernhard, Josh Benaloh, J. Alex Halderman, Ronald L. Rivest, Peter Y. A. Ryan, Philip B. Stark, Vanessa Teague, Poorvi L. Vora, and Dan S. Wallach. Public evidence from secret ballots. In Robert Krimmer, Melanie Volkamer, Nadja Braun Binder, Norbert Kersting, Olivier Pereira, and Carsten Schürmann, editors, *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, volume 10615 of *Lecture Notes in Computer Science*, pages 84–109. Springer, 2017.

[22] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.

[23] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptol. ePrint Arch.*, 2000:67, 2000.

[24] David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors. *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*. Springer, 2010.

[25] David Chaum, Peter Y. A. Ryan, and Steve 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*, pages 118–139, 2005.

[26] Véronique Cortier, Constantin Cătălin Drăgan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: Privacy and verifiability for Belenios. In *Proceedings of the 31st IEEE Computer Security Foundations Symposium, Oxford, United Kingdom, July 9-12, 2018*, pages 298–312. IEEE Computer Society, 2018.

[27] Véronique Cortier, Fabienne Eigner, Steve Kremer, Matteo Maffei, and Cyrille Wiedling. Type-based verification of electronic voting protocols. In Riccardo Focardi and Andrew C. Myers, editors, *Proceedings of the 4th International Conference on Principles of Security and Trust, London, UK, April 11-18, 2015*, volume 9036 of *Lecture Notes in Computer Science*, pages 303–323. Springer, 2015.

[28] Véronique Cortier, Alicia Filipiak, and Joseph Lallemand. BeleniosVS: Secrecy and verifiability against a corrupted voting device. In *Proceedings of the 32nd IEEE Computer Security Foundations Symposium, Hoboken, NJ, USA, June 25-28, 2019*, pages 367–381, 2019.

[29] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed ElGamal à la Pedersen: Application to Helios. In Ahmad-Reza Sadeghi and Sara Foresti, editors, *ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 131–142. ACM, 2013.

[30] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for Helios under weaker trust assumptions. In *Proceedings of the 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014*, pages 327–344, 2014.

[31] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz 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*, pages 779–798. IEEE Computer Society, 2016.

[32] Véronique Cortier, David Galindo, and Mathieu Turuani. A formal analysis of the Neuchatel e-voting protocol. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 430–442. IEEE, 2018.

[33] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Belenios: A simple private and verifiable electronic voting system. In Joshua D. Guttman, Carl E. Landwehr, José Meseguer, and Dusko Pavlovic, editors, *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows*, volume 11565 of *Lecture Notes in Computer Science*, pages 214–238. Springer, 2019.

[34] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.

[35] Chris Culnane, Aleksander Essex, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. Knights and knaves run elections: Internet voting and undetectable electoral fraud. *IEEE Secur. Priv.*, 17(4):62–70, 2019.

[36] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election verifiability or ballot privacy: Do we need to choose? In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 481–498. Springer, 2013.

[37] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 10–18, 1984.

[38] Andrew D. Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, pages 77–91. IEEE Computer Society, 2002.

[39] D. Gritzalis. *Secure Electronic Voting*. Advances in Information Security. Springer US, 2003.

[40] Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 685–702. ACM, 2019.

[41] J. Alex Halderman and Vanessa Teague. The new south wales ivote system: Security failures and verification flaws in a live online election. In Rolf Haenni, Reto E. Koenig, and Douglas Wikström, editors, *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*, volume 9269 of *Lecture Notes in Computer Science*, pages 35–53. Springer, 2015.

[42] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology - EUROCRYPT 2000*, pages 539–556. Springer, 2000.

[43] Markus Jakobsson, Ari Juels, and Ronald 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*, pages 339–353, 2002.

[44] Chris Karlof, Naveen Sastry, and David A. Wagner. Cryptographic voting protocols: A systems perspective. In Patrick D. McDaniel, editor, *Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, July 31 - August 5, 2005*. USENIX Association, 2005.

[45] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-end verifiable elections in the standard model. In Elisabeth Oswald and Marc Fischlin, editors, *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*, volume 9057 of *Lecture Notes in Computer Science*, pages 468–498. Springer, 2015.

[46] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Proceedings of the 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[47] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010.

[48] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*, pages 538–553. IEEE Computer Society, 2011.

[49] Ralf Küsters, Tomasz Truderung, and Andreas 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*, pages 395–409. IEEE Computer Society, 2012.

[50] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification, Saint Petersburg, Russia, July 13-19, 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[51] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer, 2006.

[52] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *CCS 2001, ACM Conference on Computer and Communications Security*, pages 116–125, 2001.

[53] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology - EUROCRYPT '91*, pages 522–526, 1991.

[54] Olivier Pereira and Dan S. Wallach. Clash attacks and the star-vote system. In Robert Krimmer, Melanie Volkamer, Nadja Braun Binder, Norbert Kersting, Olivier Pereira, and Carsten Schürmann, editors, *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, volume 10615 of *Lecture Notes in Computer Science*, pages 228–247. Springer, 2017.

[55] Ronald L Rivest. On the notion of 'software independence'in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759–3767, 2008.

[56] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 21-25, 1995, Proceeding*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 1995.

[57] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *25th IEEE Computer Security Foundations Symposium, (CSF'12)*, pages 78–94. IEEE Computer Society, 2012.

[58] Ben Smyth, Mark Ryan, Steve Kremer, and Mounira Kourjieh. Towards automatic analysis of election verifiability properties. In Alessandro Armando and Gavin Lowe, editors, *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security - Joint Workshop, ARSPA-WITS 2010, Paphos, Cyprus, March 27-28, 2010. Revised Selected Papers*, volume 6186 of *Lecture Notes in Computer Science*, pages 146–163. Springer, 2010.

[59] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the estonian internet voting system. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 703–715. ACM, 2014.

[60] Douglas Wikström. The Open Verificatum Project. `https://www.verificatum.org/`.

[61] Scott Wolchok, Eric Wustrow, J. Alex Halderman, Hari K. Prasad, Arun Kankipati, Sai Krishna Sakhamuri, Vasavya Yagati, and Rop Gonggrijp. Security analysis of india's electronic voting machines. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 1–14. ACM, 2010.

[62] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. Attacking the washington, D.C. internet voting system. In Angelos D. Keromytis, editor, *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, volume 7397 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2012.