Election Verifiability in Receipt-free Voting Protocols

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

Abstract—**Electronic voting is a prominent example of conflicting requirements in security protocols, as the triad of privacy, verifiability and usability is essential for their deployment in practice. Receipt-freeness is a particularly strong notion of privacy, stating that it should be preserved even if voters cooperate with the adversary. While there are impossibility results showing we cannot have receipt-freeness and verifiability at the same time, there are several protocols that aim to achieve both, based on carefully devised trust assumptions. To evaluate their security, we propose a general symbolic definition of election verifiability, extending the state of the art to capture the more complex structure of receipt-free protocols. We apply this definition to analyse, using ProVerif, recent protocols with promising practical features: BeleniosRF and several variants of Selene. Against BeleniosRF, we find several attacks showing that verifiability in Belenios does indeed suffer from the attempt to introduce receipt-freeness. On the other hand, Selene satisfies a weaker notion of receipt-freeness, but we show that it satisfies verifiability in stronger corruption scenarios. We introduce a general framework to compare the verifiability of these protocols in various corruption scenarios and conclude with an analysis of SeleneRF, an attempt to get the best of both that we formalise in this paper. In addition to extending the symbolic model, our results point to foundational gaps in current cryptographic models for election verifiability, as they fail to uncover attacks that we do.**

I. Introduction

Electronic voting brings many advantages to the process of running an election, providing more straightforward procedures and reducing the costs and the potential for human errors. On the other hand, it also brings in possible security weaknesses by increasing the scope and abilities of an adversary against the election. Electronic voting protocols aim to address this problem by incorporating a strong cryptographic foundation, ensuring the security and robustness of the system independently of the corruption of software, hardware or any of the parties involved. There are two main security properties such protocols aim to achieve: *voter privacy* [8], [25] and *election verifiability* [20], [28]. Furthermore, privacy should hold even when voters may cooperate with the adversary. This is to prevent, for example, vote buying or voter coercion. This stronger notion of privacy is typically called receipt-freeness. Both receipt-freeness and verifiability should hold with minimal trust assumptions, reflecting a set of maximally admitted corruption abilities for the adversary.

There is a well-known tension between receipt-freeness and verifiability in electronic voting protocols. On the one hand, there are impossibility results like [14], although one could argue that their underlying assumptions do not apply in all cases. On the other hand, initial protocols attempting to achieve the two, like JCJ/Civitas [15], [27], remained mostly of theoretical interest, due to strong assumptions and weak usability. If we drop the requirement of receipt-freeness, however, there are several deployed practical systems satisfying both privacy and verifiability. Most prominently, Helios [3], [4] and Belenios [2], [21] also enjoy formal security proofs for privacy [16], [22]–[24] and verifiability [6], [17]–[19]. Belenios is an elaboration of Helios aiming to achieve verifiability in stronger corruption scenarios. One step further, BeleniosRF relies on a novel cryptographic primitive (signatures on randomizable ciphertexts) in order to achieve receipt-freeness with trust assumptions similar to Belenios, albeit relying on a less intuitive individual verification procedure [12].

Selene is a recently proposed system that achieves receipt-freeness based on a new mechanism that allows voters to verify their vote directly in the final outcome, while hiding the link between the credentials used for vote casting and the credentials used for vote verification [35]. The idea is that the verification credential (also called tracker) is held in encrypted and committed form next to the voting credential. Using private trapdoor keys associated to each voter, and randomness obtained from trustees after tally, voters can open the commitment and obtain their trackers. Furthermore, this information also allows voters to generate fake randomness in order to resist coercion. Selene has an advantage over BeleniosRF in the usability of its individual verification procedure: voters can directly check their votes associated to trackers in the outcome. The disadvantage is that voters have to generate and register their private trapdoor keys, and receipt-freeness is not immediate like in BeleniosRF. Furthermore, since ballots are posted as such on the bulletin board, the receipt-freeness property is also weaker, as the ballot can be part of a receipt for the adversary.

Formal security proofs have been essential for the development of Helios and Belenios. Many attacks against these systems were found with formal methods and have led to security improvements. In this paper we aim to extend the required formal foundations in order to perform automated verification of election verifiability in receipt-free voting protocols, including BeleniosRF, Selene, and others. A general symbolic definition of election verifiability has been recently proposed in [6] and applied to Helios and Belenios. It considers various adversarial models and ensures strong end-to-end security guarantees. However, it cannot capture an essential feature of some receipt-free voting protocols, like Selene: the verification credential is not publicly linked to the vote casting credential. This is not a mere technicality and may hide real attacks against election verifiability, for example if the adversary manages to map two voting credentials onto the same verification credential, in a variation of clash attacks [29], [30], [33].

Our contributions. We extend the symbolic definition of election verifiability from [6] to also cover the more complex case of receipt-free voting systems. We also improve the definition in [6] in order to more closely match the general definition of election verifiability used in previous works, e.g. in [17]–[19]. We apply the definition to verify with ProVerif the protocols BeleniosRF, Selene and variants of Selene that improve its efficiency or receipt-freeness. We find new attacks on BeleniosRF and provide new proofs for Selene in strong corruption scenarios. We identify various corruption scenarios that pinpoint the trade-offs between BeleniosRF and Selene, and between receipt-freeness and verifiability. We give more details about these contributions in the following.

General vs symbolic election verifiability. The symbolic definition in [6] is generic and can be applied to many corruption scenarios. It has been used to find real attacks and proofs for electronic voting protocols. However, it has one important drawback, in that it was not proved to imply the general definition of election verifiability used in previous work and first introduced in [19]. Conversely, some attacks found with [6] against Belenios do not represent real attacks against election verifiability, so their definition is too strong for some scenarios. The issue in both these cases lies in a departure that was made in [6] from the notion of corrupted voters towards the notion of corrupted credentials, which creates a gap between what is proved (with respect to credentials) and the real execution of the system (with respect to voters). We fix this problem by introducing a stronger formal connection between voter identities and public credentials.

Symbolic election verifiability vs receipt-freeness. End-to-end election verifiability aims to ensure a correspondence between the final outcome and other events that occur during a run of the voting protocol: registration, corruption, voting, verification, etc. Any of these events can be associated to a particular voting credential. In Selene, each vote in the final outcome is also associated to a verification credential. For receipt-freeness, it is important that verification credentials cannot be publicly linked to their corresponding voting credential. This means that, in order to formalise election verifiability in this context, we need a function to *extract* the verification credential associated to a voting credential. Of course, to preserve receipt-freeness, this function should not be efficiently computable. For verifiability, it is sufficient for it to be well-defined and for its arguments to come from the bulletin board. We provide a framework that ensures the extraction function is well-defined and, when plugged into our updated symbolic verifiability definition, does indeed ensure end-to-end verifiability.

Attacks vs proofs for BeleniosRF. Our attacks against BeleniosRF may be surprising, considering that it was formally shown to satisfy election verifiability [12], based on a generic transformation proved secure in [19]. In fact, our attacks show the importance of rigorous adversarial models combined with machine-checked proofs. There are two problems with the definition and proofs in [19]. For the case of a corrupt registrar and honest server, the definition in [19] (relying on a cryptographic model) fails to provide the adversary with the ability to control the signing key of voters. Since security is derived following a hand-based proof, the issue of key generation was easy to oversee. A second problem, common to all current cryptographic models of verifiability (including the machine-checked model proposed later for Belenios in [17]), is that, when the voting server is honest, the ballot is posted directly on the bulletin board, assuming in effect that the communication network and all underlying infrastructure on the voting platform is secure. This defeats the purpose of verifiability, that aims to counter precisely attacks against such infrastructure. We follow the approach standard in symbolic definitions [5], [6], [18], [28], which is more realistic in this respect: the only assumption is that the voting server performs ballot verification and authentication, while the adversary is allowed to control the communication network and block ballots on their way to the bulletin board.

Once the adversary is allowed to control the signing key for honest voters and the communication network, the attack against BeleniosRF follows from the fact that voters do not verify the cast ballot directly on the bulletin board. Instead, when a cast ballot b reaches the server, it is rerandomized into a ballot b' (maintaining a signature with respect to the same key as b) that is posted on the bulletin board. The voter verifies b' by ensuring it is signed with the voter's own key, which ensures that b' is indeed a rerandomization of b. However, the following clash attack can be mounted in the considered adversarial scenario. At registration, the adversary assigns the same key to two honest voters id_1 (voting v_1) and id_2 (voting v_2). During voting, the adversary allows the ballot from id_1 to be cast, and blocks the ballot from id_2, relying on the control of the communication network. Both of these voters will successfully verify the ensuing ballot b' on the bulletin board. For id_2, this will clearly violate individual verifiability (when v_2 is different from v_1) or the no-clash property (when v_2 is equal to v_1). Overall, end-to-end verifiability is violated since not all votes that have been successfully verified are included in the final outcome.

The case of corrupt voters. We find another weakness of BeleniosRF over Belenios in the case when voter credentials are corrupted. This possibility underlies the stronger notion of verifiability introduced in [6] in order to offer guarantees in the case when voters inadvertently lose their credentials. This case is different from that of a corrupted registrar: on the one hand, it means that all private credentials of a voter are leaked, not only the signing key; on the other hand, the adversary cannot choose the secret key of voters, preventing the attack presented above. For this scenario, Belenios was proved to be secure in [6], while a simple variation of the attack described above leads to a violation of individual verifiability in BeleniosRF: a voter can successfully verify a ballot b' even if it is the rerandomization of a ballot cast by the adversary.

The case of corrupt platforms. The case of a corrupted voting platform can be countered in Helios and Belenios relying on the classic cast-or-audit mechanism known as the Benaloh challenge [7]. This type of challenge can have a usability problem, as confirmed in a recent study [31]. An interesting aspect of Selene is that such a mechanism is not necessary in this case, since voters verify their votes directly in the outcome. We note, however, that voters still need to ensure their private trapdoor keys are secret, e.g. stored on a device that is more trustworthy than the voting platform. A formal model for this corruption scenario has not been considered before, and it is another contribution of this paper. Considering this model, election verifiability is violated in BeleniosRF (as expected, assuming no Benaloh challenge is performed), while we prove that Selene is indeed secure in this case - satisfying however a weaker notion of security that allows the adversary to cast ballots for honest voters not verifying their votes.

Towards stronger security for Selene. Our findings show stronger verifiability properties in Selene over BeleniosRF, while it is known that BeleniosRF satisfies a stronger notion of receipt-freeness due to signatures on rerandomizable ciphertexts. One may read this as an illustration of the trade-off between the two properties. Yet, as suggested in [35], we can also consider a version of Selene that relies on the same primitive as BeleniosRF to rerandomize ballots before posting. We call this version SeleneRF and we prove with ProVerif that it satisfies the same verifiability properties as Selene. We expect it to satisfy the same receipt-freeness notion as BeleniosRF, but this has to be formally proven in future work. Interestingly, in spite of the fact that the ballots are rerandomized, in SeleneRF we do not find attacks similar to the ones described above for BeleniosRF. This is due to the fact that voters directly verify their votes in the outcome, and not their ballots. An advantage of BeleniosRF is that it thwarts a coercer by design, and the voter does not have to follow any coercion-resistance strategy to achieve receipt-freeness. In Selene and its variants, the voter needs to construct a fake verification credential to resist coercion. Another improvement of Selene that we consider is a recently proposed version called Hyperion, showing a simpler way of generating verification credentials [34]. The system description in [34] is quite informal. We provide a formal specification and we prove its election verifiability property with ProVerif.

Paper structure. Section II contains preliminaries for e-voting protocols and formal verification with ProVerif. Section III introduces receipt-free voting protocols, specifically BeleniosRF, Selene, and Hyperion. A general symbolic verifiability definition revised to include receipt-free voting protocols is presented in Section IV. Finally, a comparative verifiability analysis of the considered protocols is presented in Section V.

II. Preliminaries

A. *Verifiable electronic voting protocols*

Bulletin board. Electronic voting protocols utilise a public bulletin board, which we denote by BB, to record information regarding the execution of the protocol for the verifiability and transparency of the election. It can be used by the voters to verify their ballots after the voting procedure (this is also called *individual verifiability*) and by any party to verify the integrity of the election (*universal verifiability*). We can distinguish different parts of the BB by annotations, e.g. BBkey for the public key of the election or BBreg for the registered eligible public credentials.

Setup. A voting protocol assumes a set of eligible voters represented by identities $\mathsf{id}_1, \dots, \mathsf{id}_n$. Election authorities generate their corresponding public credentials $\mathsf{cr}_1, \dots, \mathsf{cr}_n$, that we assume recorded on BBreg. The public key of the election is generated by authorities and posted on BBkey. At registration time, voters obtain their (private) id, their public credential cr, and additional private credentials which will be used for authentication and verification.

Voting procedure. During the voting phase, voters use voting platforms to construct a ballot b, encoding their desired vote v. The ballot is sent to the voting server after authentication. The server validates the ballot b and publishes corresponding information on BB. Typically the ballot itself is published on BB along with the public credential of the voter, but other options are possible. For example, BeleniosRF publishes a rerandomization of the ballot.

Individual verification. Voters can perform certain procedures to ensure that the system correctly records their votes. Typically, voters have to check that information they expect to see (e.g. their ballot or their vote) is present on the bulletin board along with tracking information associated to their public credential.

Tally procedure. The ballots to be tallied are recorded alongside eligible public credentials on the bulletin board: $(\mathsf{cr}_1, \mathsf{b}_1), \dots, (\mathsf{cr}_n, \mathsf{b}_n)$. Each ballot b_i is stripped of any information that is not needed for vote-counting, like signatures and zero-knowledge proofs, resulting in a ciphertext c_i. The tally phase consists in computing the set of votes $\{\mathsf{v}_1, \dots, \mathsf{v}_n\}$ corresponding to $\{\mathsf{c}_1, \dots, \mathsf{c}_n\}$ in an anonymous and verifiable way, typically relying on mixnets or on the homomorphic properties of the encryption scheme. For some voting protocols, ciphertexts are combined with trackers before decryption. In this case, each vote in the outcome is associated to a tracker, like $(\mathsf{tr}_1, \mathsf{v}_1), \dots, (\mathsf{tr}_n, \mathsf{v}_n)$.

Universal verification. External auditors can be involved in verifying that the ballots $\mathsf{b}_1, \dots, \mathsf{b}_n$ are really cast by the corresponding public credentials and satisfy any additional consistency properties expected from the specification of the protocols. For example, they may check that the ballots are signed with the right key, contain valid zero-knowledge proofs, are the last ballots cast for the corresponding credential, there are no duplicates, etc. In addition, auditors also verify the proofs provided in the tally phase, to ensure that $\{v_1, \dots, v_n\}$ corresponds to $\{c_1, \dots, c_n\}$.

Example 1. *Assume three election authorities: a trustee responsible for tallying ballots, a registrar for generating and registering voter keys, and a server for ballot casting.*

Setup. *The trustee generates a key pair* $(\mathsf{pk}, \mathsf{sk})$ *and publishes* pk *on* BBkey. *For each voter* id, *the registrar generates a signing key pair* $(\mathsf{spk}, \mathsf{ssk})$ *and publishes* spk *as* cr *on* BBreg. *Moreover, the server generates a password* pwd *to use for authentication of the voter. Each eligible voter obtains* $\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{ssk} \rangle$.

Voting. *The voter, relying on the voting platform, generates a ciphertext* $\mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r})$ *for some desired vote* v *and fresh randomness* r. *Then, the voter signs it with their* ssk, *obtaining a signature* s *and constructs related zero-knowledge proof* π, *resulting in a ballot* $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \pi \rangle$. *The voter authenticates to the server with their* pwd, *and the ballot* b *is submitted to the server. After server's ballot validation,* $(\mathsf{cr}, \mathsf{b})$ *is published on* BBcast.

Verification. *Voters can check their ballot on* BBcast *anytime during the voting phase, or on* $\mathsf{BBtally}$ *afterwards.*

Tally. *The last ballot corresponding to* cr *on* BBcast *is recorded on* $\mathsf{BBtally}$. *The result is computed by the trustee by decrypting each ciphertext from ballots in* $\mathsf{BBtally}$.

The previous example is a simplified version of Belenios [21], an elaboration of Helios that protects improves election verifiability relying on a relying on a registrar. Auditors ensure that all ballots on BB are signed with keys recorded by the registrar. The zero-knowledge proof on the ballot also helps to protect ballot privacy against copy attacks [24]. Even if the adversary tries to copy any ciphertext cast by any voter, they cannot generate the zero-knowledge proof without knowing the randomness used for the ciphertext.

B. Formal verification with ProVerif

ProVerif allows automated verification of security properties for cryptographic protocols [9]. Messages are represented in an equational theory where terms are built from a set of types, a set of constants, a set $\mathcal{C}$ of constructors endowed with a set $\mathcal{D}$ of destructors. Types are not essential for describing our protocol specifications, so we will not use them in this paper. A constructor is a function symbol f used to build a term $\mathsf{f}(\mathsf{M}_1, ..., \mathsf{M}_k)$, from terms $\mathsf{M}_1, ..., \mathsf{M}_k$. A destructor is a symbol with associated rewrite rules modeling the properties of cryptographic primitives. Rewrite rules are equations $\mathsf{u} = \mathsf{v}$ that should be read from left to right. For example, we can define a proj_i destructor to fetch the arguments of f: $\mathsf{proj}_i(\mathsf{f}(\mathsf{M}_1, ..., \mathsf{M}_k)) = \mathsf{M}_i$. If a term contains a destructor for which no rewrite rule can be applied, we denote this by $t = \bot$, i.e. the evaluation of the term fails. A term may also be a tuple of the form $(\mathsf{M}_1, ..., \mathsf{M}_n)$, where $\mathsf{M}_1, ..., \mathsf{M}_n$ are terms. We consider a special name $\mathsf{pub} \in \mathcal{N}$ to represent a public communication channel. Unless declared $\mathsf{private}$, names and functions are known by the adversary. A term M can be stored in a table, e.g. $\mathsf{Tb}(\mathsf{M})$. Tables are based on private constructors and are not accessible by the adversary. Terms can be checked with equality or disequality conditions, and such conditions may be combined with logical connectors.

Example 2. *The set of constructors* $\{\mathsf{pk}, \mathsf{spk}, \mathsf{enc}, \mathsf{pr}, \mathsf{sign}\}$ *can model cryptographic primitives from Example 1, along with the corresponding destructors defined by:*

(1) $\mathsf{dec}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}) = \mathsf{x}$,
(2) $\mathsf{check}(\mathsf{sign}(\mathsf{x}, \mathsf{y}), \mathsf{spk}(\mathsf{y})) = \mathsf{x}$,
(3) $\mathsf{ver}(\mathsf{pr}(\mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{x}, \mathsf{z}, \mathsf{w}), \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{y}, \mathsf{w}) = \mathsf{ok}$.

Equations (1) and (2) specify the standard properties of asymmetric encryption and digital signatures. The proof *symbol models a zero-knowledge proof showing that a ciphertext is constructed by a party that knows the corresponding plaintext* m *and randomness* r, *and that party associated another message* w *to this ciphertext. Equation (3) models the corresponding verification procedure for the proof and corresponding arguments.*

$\mathsf{M}, \mathsf{N} ::=$	terms
$\mathsf{a}, \mathsf{b}, \mathsf{c}, ..., \quad \mathsf{x}, \mathsf{y}, \mathsf{z}, ...$	names and variables
$(\mathsf{M}_1, ..., \mathsf{M}_k), \quad \mathsf{f}(\mathsf{M}_1, ..., \mathsf{M}_k)$	tuples and functions
$\mathsf{M} = \mathsf{N}, \quad \mathsf{M} <> \mathsf{N}$	term equality and disequality
$\mathsf{M}\ \&\&\ \mathsf{N}, \quad \mathsf{M}\ \|\|\ \mathsf{N}, \quad \mathsf{not}(\mathsf{M})$	conjunction, disjunction, negation
$\mathsf{P}, \mathsf{Q} ::=$	processes
$0 \quad \mathsf{P} \mid \mathsf{Q} \quad !\mathsf{P} \quad \mathsf{new}\ \mathsf{a}; \mathsf{P}$	parallel, replication, restriction
$\mathsf{in}(\mathsf{N}, \mathsf{x}); \mathsf{P} \quad \mathsf{out}(\mathsf{N}, \mathsf{M}); \mathsf{P}$	message input and output
$\mathsf{if}\ \mathsf{M}\ \mathsf{then}\ \mathsf{P}\ \mathsf{else}\ \mathsf{Q}$	conditional
$\mathsf{let}\ \mathsf{x} = \mathsf{M}\ \mathsf{in}\ \mathsf{P}\ \mathsf{else}\ \mathsf{Q}$	term evaluation
$\mathsf{insert}\ \mathsf{Tb}(\mathsf{M}); \mathsf{P}$	table insertion
$\mathsf{get}\ \mathsf{Tb}(\mathsf{x})\ \mathsf{suchthat}\ \mathsf{M}$	table extraction
$\mathsf{event}(\mathsf{M}_1, ..., \mathsf{M}_k); \mathsf{P}$	event

Fig. 1. Terms and processes in ProVerif grammar.

Processes are denoted by $\mathsf{P}, \mathsf{Q}, \dots$, and generated with the grammar from Figure 1. Recursively, 0 represents the null process, $\mathsf{P} \mid \mathsf{Q}$ the parallel execution of P and Q, and $!\mathsf{P}$ an arbitrary number of parallel executions of P. The restriction $\mathsf{new}\ \mathsf{n}; \mathsf{P}$ generates the name n, which can represent fresh randomness in the process P. The process $\mathsf{in}(\mathsf{pub}, \mathsf{x}); \mathsf{P}$ models a message received from channel pub and stored in x, while the process $\mathsf{out}(\mathsf{pub}, \mathsf{M}); \mathsf{P}$ models a process sending the message M on channel pub and continuing as P. The conditional $\mathsf{if}\ \mathsf{M}\ \mathsf{then}\ \mathsf{P}\ \mathsf{else}\ \mathsf{Q}$ runs P if M evaluates to true, runs Q if $\mathsf{M} = \mathsf{false}$. If a term M contains a destructor where no rewrite rule can be applied, we say that M fails. The term evaluation $\mathsf{let}\ \mathsf{x} = \mathsf{M}\ \mathsf{in}\ \mathsf{P}\ \mathsf{else}\ \mathsf{Q}$, bounds x to M and executes P, when M does not fail. If M fails, it directly executes the process Q. The table insertion $\mathsf{insert}\ \mathsf{Tb}(\mathsf{M}); \mathsf{P}$ stores the term M in Tb. The table extraction $\mathsf{get}\ \mathsf{Tb}(\mathsf{x})\ \mathsf{suchthat}\ \mathsf{M}\ \mathsf{in}\ \mathsf{P}\ \mathsf{else}\ \mathsf{Q}$ selects an entry from Tb that evaluates to M. If there is no such entry, it executes Q.

```
let Voting Platform(v, pk) =
    get Cred(id, cr, ssk) in
    get Pwd(= id, pwd) in
    new r; let c = enc(v, pk, r) in
    let s = sign(c, ssk) in
    let pr = proof(c, v, r, cr) in
    let b = (c, s, pr) in
    let a = h(id, pwd, cr, b) in
    event Vote(id, cr, v);
    insert Voted(id, cr, v, b);
    out(pub, (id, cr, b, a)).
```

```
let Voting Server(pk) =
    get BBreg(cr) in
    get Pwd(id, pwd) in
    in(pub, (= id, = cr, c, s, pr, a));
    let b = (c, s, pr) in
    if h(id, pwd, cr, b) = a then
    if check(s, cr) = c then
    if ver(pr, c, pk, cr) = ok then
    event BBcast(cr, b);
    insert BBcast(cr, b).
```

Fig. 2. Ballot casting processes for voting platform and server.

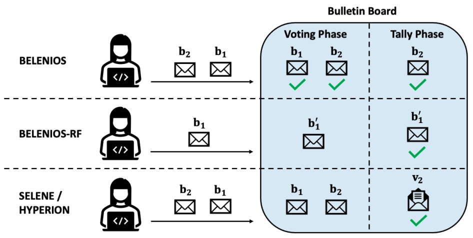

Fig. 3. Illustrating revoting and verifiability in various protocols. For Belenios and Selene: b_2 is a revote ballot with desired vote. For BeleniosRF: b_1 contains the desired vote rerandomized to b'_1.

Example 3. *The processes in Figure 2 model basic operations of a voting protocol.* BB *is modelled by tables. Voting credentials are retrieved from* Cred *and* Pwd*, and a vote* v *is chosen. A signature* s*, a proof* pr*, and authentication information* a *are computed. Information relevant for individual verifiability is stored in the table* Voted(id, cr, v, b)*, and an event* Vote(id, cr, v) *is recorded. The tuple* (id, cr, b, a) *is sent to the public channel. The voting server receives* b *and publishes it on* BB *if validation checks are ok.*

Security properties. Security properties in this paper are modelled by correspondence assertions, capturing relations between events. For example, the formula $\mathsf{e}(\mathsf{M}) \Rightarrow \mathsf{e}'(\mathsf{M}')$ states that if the event $\mathsf{e}(\mathsf{M})$ is executed in the trace, then the event $\mathsf{e}'(\mathsf{M}')$ has been executed before in that trace. Queries can be more complex, combining events with relations between terms inside the events, e.g. we can add $\mathsf{M} = \mathsf{M}'$ on the right-hand side of the previous query. In general, we consider a set of timepoint variables, denoted by $i, j, l, \ldots$, which will be interpreted over rational numbers. A trace atom is either a term equality $\mathsf{t}_1 = \mathsf{t}_2$, or a timepoint ordering $i \prec j$, or a timepoint equality $i = j$, or an event atom $\mathsf{e}(\mathsf{M})@i$ - the timepoint can be omitted from the event atom if it is not relevant for the property. A trace formula is a first-order logic formula obtained from trace atoms by applying the usual quantification and logical connectives.

Example 4. *Considering the processes from Figure 2 and assuming a destructor* open *that reveals the vote* v *inside a ballot* b*, we can check whether any ballot from* BBcast *comes from a corresponding* Vote *event in the voting platform, using formula:* $\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \implies \mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \wedge \mathsf{open}(\mathsf{b}) = \mathsf{v}$. *This formula allows multiple cast ballots per credential, and for all of them it requires a corresponding voting event.*

An execution trace τ of a process P is defined as a sequence of events $\mathsf{E}_1, ..., \mathsf{E}_n$ that occurred during the execution of P. The satisfaction relation $\tau \vDash \Phi$, for a trace τ and a trace formula Φ, whose variables are all bounded, is defined recursively as expected, with the following notable case: $\tau \vDash \mathsf{e}(\mathsf{M})@i$ if and only if $\mathsf{e}(\mathsf{M})$ occurs on the i-th position of the trace. For a process P, we let tr(P) be the set of traces of P. For trace formulas Ψ, Φ, we let:

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

For verification of security properties, $(\mathsf{P}; \Psi)$ is typically a protocol specification and Φ the property to be verified. Having the component Ψ, also called a restriction, in a protocol specification can help to express in a concise way some properties that protocol parties should ensure along an execution trace. If the equational theory $\mathcal{E}$ is not clear from the context, we write $(P, \Psi, \mathcal{E}) \vDash \Phi$.

III. Receipt-free Voting Protocols

We can distinguish three techniques that are instrumental for receipt-free voting protocols. First, like in JCJ/Civitas [15], [27], voters can be given the ability to construct a fake credential, which can be used to cast a ballot according to the request of the adversary. Second, like in BeleniosRF [12], ballots published on the bulletin board can be derived from cast ballots with advanced cryptography that allows to ensure the published ballots encode the same votes as cast ballots, while hiding the link between them. Finally, like in Selene [35] and Hyperion [34], voters can be given the ability to derive a tracker showing their vote in the final outcome; to resist coercion, they have the ability to derive a fake tracker pointing to a vote desired by the adversary, relying on their private trapdoor keys. In Figure 3, we show how these protocols relate to a classic protocol like Belenios. In Belenios, the ballot is verified on the bulletin board in the same form as it was cast. In BeleniosRF, the ballot is randomized by the voting server before being posted on BB: it cannot be used as a coercion tool. BeleniosRF disallows revoting; otherwise, a corrupt server could choose which of the ballots from a given voter to cast.

In Selene and Hyperion, the vote is verified directly in the outcome after trackers are revealed to voters. Nobody can associate a credential to a tracker on BB, except the voter and dedicated election authorities. In case of coercion, a private trapdoor key allows voters to choose any tracker from the outcome that conforms to the instructions of the coercer. We illustrate this approach with a simple example first. A *trapdoor commitment scheme* consists of two main operations: com(m, tpk, r) allows to commit a message m for trapdoor public key tpk with randomness r; open(cm, tsk, r) allows to obtain the message m from a commitment relying on the corresponding trapdoor secret key and randomness. This key may also allow to produce fake randomness that opens the commitment to a different message.

Example 5. *Assume each voter has a trapdoor key pair* $\langle \mathsf{tpk}, \mathsf{tsk} \rangle$ *for a commitment scheme. Voters publish their* tpk *next to their credential* cr *on* BB*. Election authorities generate a tracker* tr *for each voter and publish encryption of* tr *and a commitment to* tr *for the trapdoor public key of the corresponding voter. After voting, we have the following information on* BB *for each credential:* $\mathsf{cr_i}, \mathsf{enc}(\mathsf{tr_i}, \mathsf{pk}), \mathsf{com}(\mathsf{tr_i}, \mathsf{tpk_i}, \mathsf{r_i}), \mathsf{enc}(\mathsf{v_i}, \mathsf{pk})$*. Votes and trackers are decrypted at tally:* $\mathsf{BB} : (\mathsf{tr_1}, \mathsf{v_1}), \ldots, (\mathsf{tr_n}, \mathsf{v_n})$.

For individual verification, election authorities send the commitment randomness $\mathsf{r_i}$ *to each voter. Voters obtain their tracker by applying* $\mathsf{open}(\mathsf{cm}, \mathsf{tsk_i}, \mathsf{r_i})$*, where* cm *is the commitment recorded next to their credential. For receipt-freeness, voters can choose any tracker* $\mathsf{tr_j}$ *from the outcome and construct fake randomness that opens their commitment to this tracker.*

A. BeleniosRF

BeleniosRF [12] is a variant of Belenios [21], which aims for receipt-freeness in addition to end-to-end verifiability. It relies on a cryptographic primitive called signatures on randomizable ciphertexts [10], instantiated from Groth-Shai proofs [26]. Like in Belenios, the public credential of a voter is the public part of the signing key generated by the registrar, and the ballots cast are of the form $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \pi \rangle$, where c is an encryption of the desired vote, s is the signature of c, and π is a zero-knowledge proof showing the validity of the ciphertext and associating it to the public credential of the voter. Unlike Belenios, the ballot is rerandomized before being posted next to cr on BBcast. More precisely: the voting server re-encrypts the ciphertext c and, relying on [10], adapts the signature s and the zero-knowledge proof π for the new ciphertext c'. The randomized ballot $\mathsf{b}' = \langle \mathsf{c}', \mathsf{s}', \pi' \rangle$ is now a valid ballot for the same credential cr and is published next to it on BBcast.

The individual verification procedure for a voter with credential cr consists in verifying that there is a ballot b' associated to cr on BBcast, and verifying that b' contains a valid signature with respect to cr. Intuitively, this should be sufficient to ensure election verifiability if either the registrar or the server is not corrupt. If the server is honest, it will log the identity of voters together with their public credentials, so even if a voter's secret key is corrupted, the adversary should not be able to cast a ballot for the credential of an honest voter. Symmetrically, if the registrar is honest, the only party who can construct valid ballots for a public credential is the corresponding voter - verifiability follows even for a corrupt server since revoting is not allowed. In addition, BeleniosRF satisfies receipt-freeness: a coerced voter cannot prove how they voted, as there is no way for the adversary to determine whether any ballot provided by the voter corresponds to the one cast on the bulletin board - relying on the assumption that the voting server is honest.

B. Selene

The main feature of Selene [35] is that it enables intuitive end-to-end verifiability by allowing voters to verify their vote directly in the outcome of the election. For this purpose voters are provided with trackers that point to their votes in the outcome. For receipt-freeness, trackers can be obtained by voters only at the end of the election: relying on trapdoor commitments like in Example 5, election authorities send voters the information that is required to open their tracker from a commitment, and also to construct a fake tracker for coercion-resistance. Selene relies on ElGamal encryption, a digital signature scheme, non-interactive zero-knowledge proofs and Pedersen trapdoor commitments. Mixnets are used for anonymising ciphertexts containing votes and trackers before decryption.

Encrypted trackers $\mathsf{enc}(\mathsf{tr_1}, \mathsf{pk}), \ldots, \mathsf{enc}(\mathsf{tr_n}, \mathsf{pk})$ are published on BB, where pk is the public key of the election (with the secret key held by election trustees) and $\mathsf{tr_1}, \ldots, \mathsf{tr_n}$ is a random permutation obtained by encrypting each element of a public list of trackers and passing it through a re-encryption mixnet [13], [32]. Like in Belenios, the public credential of each voter is a public signing key. In addition, voters generate a trapdoor key pair $\langle \mathsf{tpk}, \mathsf{tsk} \rangle$, and register its public part next to their credential: $\mathsf{BB} : (\mathsf{cr_1}, \mathsf{tpk_1}), \ldots, (\mathsf{cr_n}, \mathsf{tpk_n})$. Trustees associate each credential with an encrypted tracker. In addition, based on the homomorphic properties of ElGamal and on its interaction with the Pedersen commitment scheme, the trustees can compute a commitment to the tracker for the public trapdoor key associated to the corresponding voter. The commitment along with a zero knowledge proof of correctness $\pi^{\mathsf{i}}_{\mathsf{ptr}}$ is recorded for each credential $\mathsf{cr_i}$ on the bulletin board:

$$\mathsf{BBptr} : \overbrace{\mathsf{cr_i}, \mathsf{tpk_i}, \mathsf{enc}(\mathsf{tr_i}, \mathsf{pk}), \mathsf{com}(\mathsf{tr_i}, \mathsf{tpk_i}, \mathsf{r_i}), \pi^{\mathsf{i}}_{\mathsf{ptr}}, \mathsf{pk}}^{\text{tracking data}}$$

Ballots cast by the voter are of the form $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \pi \rangle$, which is the same as in Belenios, and are posted alongside the public credential on BB. Before the tally, zero-knowledge proofs and signatures are verified. For computing the outcome, the pairs of ciphertexts $(\mathsf{enc}(\mathsf{tr_1}, \mathsf{pk}), \mathsf{c_1}), \ldots, (\mathsf{enc}(\mathsf{tr_n}, \mathsf{pk}), \mathsf{c_n})$ corresponding to each voter are passed through a parallel re-encryption mixnet. Such a mixnet will produce a list of pairs that encode the same information as the input list of pairs, where each element is unlinkable to the corresponding element in the input. Each pair is verifiably decrypted, resulting in the outcome $\mathsf{BBres} : (\mathsf{tr_1}, \mathsf{v_1}), \ldots, (\mathsf{tr_n}, \mathsf{v_n}), \pi_{\mathsf{res}}$, where π_{res} is an overall proof of re-encryption and decryption, ensuring that the outcome is correctly computed from the list of input ciphertext pairs. After the outcome is computed, each voter receives the randomness r for their corresponding commitment from trustees. Using their trapdoor secret key tsk and the randomness r, they can open the commitment and obtain their trackers for verification. If voters are coerced, they can construct fake randomness to open their commitment to a different tracker. Suppose the coercer has forced the voter to vote for v'. The voter finds a tracker-vote pair $(\mathsf{tr}', \mathsf{v}')$ from the outcome on BB, and computes the fake randomness r' using their tsk, tr and r that will open their commitment to tr'. The trapdoor signing key tsk of the voter and fake randomness r'

can be provided to the coercer, which opens their commitment to a different tracker tr', showing that the voter voted as the coercer desired.

We also consider SeleneRF, that adopts the primitive for signatures on rerandomizable ciphertexts from BeleniosRF. The protocol is exactly the same as Selene, the only difference being that the signature and proofs on the ballot are performed with the primitives from [12]. The tally and verification procedures remain the same as in Selene, as a rerandomized ballot b' can be tallied in the same way as b and the trackers can be extracted in the same way.

C. Hyperion

Hyperion improves the tracker management in Selene by deriving the tracker for each voter directly from their public trapdoor key [34]. This removes the need to generate, encrypt and anonymise trackers in advance. Moreover, computing pre-tracking information and the trackers in the final outcome is also more efficient. The main idea in Hyperion is that, instead of encrypted and committed trackers like in Selene, the pre-tracking information is simply tpk raised to a power r, where tpk is the trapdoor public key of the voter, and r is randomness chosen by trustees, different for each voter. We call $\mathsf{tpk}^{\mathsf{r}}$ a pre-tracker. Once the voting phase has ended, each pre-tracker $\mathsf{tpk}^{\mathsf{r}}$ is paired with the ciphertexts c encoding the vote of the corresponding voter. The list of all such pairs $(\mathsf{tpk}_1^{\mathsf{r}_1}, \mathsf{c}_1), \ldots, (\mathsf{tpk}_n^{\mathsf{r}_n}, \mathsf{c}_n)$ is passed through a parallel mixnet where the second component of each pair is re-encrypted to a different ciphertext, as in a classical mixnet, while the first component is exponentiated to a power s, where s is the same for all the elements of the list. The mixnet for the first component is called an exponentiation mix. After decrypting the ciphertexts in the resulting outcome, we obtain:

$$\begin{array}{lllll} \mathsf{BBres}: & (\mathsf{tr}_1, \mathsf{v}_1), & \ldots, & (\mathsf{tr}_n, \mathsf{v}_n), & \pi_{\mathsf{res}} \\ \text{where} & \mathsf{tr}_1 = \mathsf{tpk}_1^{\mathsf{r}_1\mathsf{s}}, & \ldots, & \mathsf{tr}_n = \mathsf{tpk}_n^{\mathsf{r}_n\mathsf{s}} & \end{array}$$

The trustees know $\mathsf{g}^{\mathsf{r}_i}$ and s, from which they compute $\mathsf{g}^{\mathsf{r}_i\mathsf{s}}$ and send it to the corresponding voter. Furthermore, we should note that $\mathsf{tpk}_i = \mathsf{g}^{\mathsf{spk}_i}$, where spk_i is the trapdoor secret key of the voter. Therefore, by exponentiating $\mathsf{g}^{\mathsf{r}_i\mathsf{s}}$ to the power spk_i, voters can obtain their trackers $\mathsf{tr}_i = \mathsf{tpk}_i^{\mathsf{r}_i\mathsf{s}}$. As for Selene, we gather all data relevant for tracking on BBptr:

$$\mathsf{BBptr} : \overbrace{\mathsf{cr}_i, \mathsf{tpk}_i, \mathsf{tpk}_i^{\mathsf{r}_i}, \pi}^{\text{tracking data}}$$

The proof $\pi = \mathsf{prmix}(\mathsf{M}_{\mathsf{in}}, \mathsf{s}, \mathsf{M}_{\mathsf{out}})$ represents the (exponentiation) mixnet proof showing that the final set of trackers $\mathsf{M}_{\mathsf{out}}$ is obtained by exponentiating each element in the set of pre-trackers M_{in} to the same secret power s. A further enhancement proposed in [34] is to derive a separate bulletin board for each voter. This improves receipt-freenes, since any fake tracker that the voter may derive cannot clash with trackers owned by the coercer. This is especially pertinent in elections with a small number of voters. We do not consider this option in this paper, but it should be straightforward to extend our analysis for this case, as the individual bulletin board for each voter is derived from a further exponentiation mix, with a separate s_i for each voter, and the corresponding tracker is $\mathsf{tpk}_i^{\mathsf{r}_i\mathsf{s}\mathsf{s}_i}$. We need to record $\mathsf{g}^{\mathsf{s}_i}$ on BB and update the tracker extraction function accordingly.

IV. Election verifiability definitions

Before introducing our symbolic definition for election verifiability, we recall the definition of election verifiability from [19], that has been adapted both to computational [17] and symbolic models [18]. Furthermore, we introduce some notation that will be useful for bridging the gap from the general definition to the symbolic model, and we also extend the definition from [19], in order to have a generic notion that can cover stronger or weaker guarantees of election verifiability depending on considered voting protocols or trust assumptions.

A. General definition of election verifiability

Definition 1. *A* voting specification *is a tuple $(\mathcal{P}, \Psi, \mathcal{E}, \Omega)$, for a process $\mathcal{P}$, a restriction Ψ, an equational theory $\mathcal{E}$ and a trace formula Ω called the revote policy. Furthermore, we assume $\mathcal{P}$ relies on fact symbols* $\mathsf{Vote}, \mathsf{BBres}, \mathsf{Verified}, \mathsf{Corr}$ *to record the following events:*

- $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$*: vote* v *is cast by the voter with identity* id*;*
- $\mathsf{BBres}(\mathsf{tr}, \mathsf{v})$*: vote* v *is in the result next to a counter* tr*;*
- $\mathsf{Verified}(\mathsf{id}, \mathsf{cr}, \mathsf{v})$*: vote* v *is verified by a voter with identity* id *and public credential* cr*;*
- $\mathsf{Corr}(\mathsf{id})$*: the voter with identity* id *is corrupt.*

The process $\mathcal{P}$ and restrictions Ψ model the procedures of an electronic voting protocol: setup, voting, tally, individual and public verification; as well as the abilities of the adversary to corrupt parties and control the network. The revote policy $\Omega(\mathsf{id}, \mathsf{v})$ should say under what conditions the vote v that was cast by id should be counted. For example, it may say that v is the first or the last vote cast by the respective voter. Some protocols like Helios, Belenios, and BeleniosRF do not have counters (or trackers) associated to votes in the outcome. For such systems, the result of the election is considered to be the multiset of votes encoded by the ballots recorded on a special portion of the bulletin board called $\mathsf{BBtally}$, where each ballot b is recorded next to a public credential cr, resulting in events $\mathsf{BBtally}(\mathsf{cr}, \mathsf{b})$. The votes inside these ballots are determined by an extraction function open, as in e.g. [6], [18]. To apply Definition 1 to such protocols, we can therefore take $\mathsf{tr} = \mathsf{cr}$ and define: $\mathsf{BBres}(\mathsf{cr}, \mathsf{v}) \equiv \mathsf{BBtally}(\mathsf{cr}, \mathsf{b}) \wedge \mathsf{open}(\mathsf{b}) = \mathsf{v}$.

The definition of election verifiability from [19] ensures that votes in the final result can be partitioned into three multisets V_1, V_2, V_3, where

1) V_1 are votes cast by honest voters who verified their votes;
2) V_2 is a subset of the votes from honest voters who did not verify their votes; it contains at most one vote for each such voter, but some votes may be dropped under the influence of the adversary;
3) V_3 represents the votes cast by the adversary, and its size should be bounded by the number of corrupt voters.

Point 1) can be strengthened by requiring that all of the verified votes should be present in the outcome, and not only those of honest voters. On the other hand, point 3) can be weakened by allowing the adversary to also cast votes for honest voters who did not verify their vote. This motivates our definition of the following sets associated to a trace:

- $\mathsf{Ver}^\bullet$ - the multiset of verified votes satisfying the revote policy;
- Ver° - as above, but only including votes of honest voters;
- $\mathsf{Adv}^\bullet$ - the set of corrupt voter identities;
- Adv° - the set of corrupt voter identities for which the verification procedure has not been performed.
- $\mathsf{Vote}^{\triangledown,\diamond}$ - the set containing all subsets of the votes from *non-adversarial voters* who have not *verified* their votes; the definition of this set varies according to the scenario $(\triangledown, \diamond) \in \{\bullet, \circ\} \times \{\bullet, \circ\}$ which is used for instantiating the notions of non-adversarial voters and verified votes.
- Res - the multiset of votes in the result of the election.

Definition 2. *Let τ be a trace produced by a voting specification having Ω as revote policy. We consider the formulas:*

$$\begin{aligned}
\mathsf{Ver}^\bullet(\mathsf{id},\mathsf{cr},\mathsf{v}) &\equiv \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \Omega(\mathsf{id},\mathsf{v})\\
\mathsf{Ver}^\circ(\mathsf{id},\mathsf{cr},\mathsf{v}) &\equiv \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \Omega(\mathsf{id},\mathsf{v}) \wedge \neg\mathsf{Corr}(\mathsf{id})\\
\mathsf{Adv}^\bullet(\mathsf{id}) &\equiv \mathsf{Corr}(\mathsf{id})\\
\mathsf{Adv}^\circ(\mathsf{id}) &\equiv \mathsf{Corr}(\mathsf{id}) \vee \neg\mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}')
\end{aligned}$$

and define the following sets, for $(\triangledown, \diamond) \in \{\circ, \bullet\} \times \{\circ, \bullet\}$:

$$\begin{aligned}
\mathsf{Ver}^\triangledown(\tau) &= \{\!\!\{\mathsf{v} \mid \exists \mathsf{id},\mathsf{cr}.\ \tau \vDash \mathsf{Ver}^\triangledown(\mathsf{id},\mathsf{cr},\mathsf{v}) \wedge \mathsf{v} \neq \bot\}\!\!\}\\
\mathsf{Ver}^\triangledown_{\mathsf{id}}(\tau) &= \{\mathsf{id} \mid \exists \mathsf{id}',\mathsf{cr},\mathsf{v}.\ \tau \vDash \mathsf{Ver}^\triangledown(\mathsf{id}',\mathsf{cr},\mathsf{v})\\
&\qquad\qquad \wedge \mathsf{Reg}(\mathsf{id},\mathsf{cr}) \wedge \mathsf{v} \neq \bot\}\\
\mathsf{Adv}^\diamond(\tau) &= \{\mathsf{id} \mid \tau \vDash \mathsf{Adv}^\diamond(\mathsf{id})\}\\
\mathsf{Res}(\tau) &= \{\!\!\{\mathsf{v} \mid \exists \mathsf{tr}.\ \tau \vDash \mathsf{BBres}(\mathsf{tr},\mathsf{v})\ \wedge\ \mathsf{v} \neq \bot\}\!\!\}
\end{aligned}$$

Additionally, we let $\mathsf{Vote}^{\triangledown,\diamond}(\tau)$ to be the set of multisets $\{\!\!\{\mathsf{v}_1,...,\mathsf{v}_s\}\!\!\}$ s.t. there are distinct $\mathsf{id}_1,...,\mathsf{id}_s$ and for all $\mathsf{i} \in \{1,\dots,\mathsf{s}\}$ we have:

$$\tau \vDash \mathsf{Vote}(\mathsf{id}_\mathsf{i},\mathsf{v}_\mathsf{i})\ \wedge\ \mathsf{id}_\mathsf{i} \notin (\,\mathsf{Ver}^\triangledown_{\mathsf{id}}(\tau) \cup \mathsf{Adv}^\diamond(\tau)\,).$$

Definition 3. *Let $(\triangledown, \diamond) \in \{\circ, \bullet\} \times \{\circ, \bullet\}$. A trace τ of a voting specification $\mathcal{S}$ satisfies $[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$-election verifiability iff there exist multisets V_1, V_2, V_3 of votes such that $\mathsf{Res}(\tau) = V_1 \uplus V_2 \uplus V_3$ and*

(1) $V_1 = \mathsf{Ver}^\triangledown(\tau)$,

(2) $V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau)$,

(3) $|V_3| \leq |\mathsf{Adv}^\diamond(\tau) \setminus \mathsf{Ver}^\triangledown_{\mathsf{id}}(\tau)|$.

We denote this by $\tau \Vdash \mathsf{E2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$. If all traces of $\mathcal{S}$ satisfy it, we have $\mathcal{S} \Vdash \mathsf{E2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$.

Definition 3 gives four different notions of verifiability:

- $\mathsf{E2E}[\mathsf{iv}_\bullet, \mathsf{res}_\bullet]$ is the strongest notion, ensuring that verified votes of corrupt voters are also included in the final tally (this is strong individual verifiability).
- $\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$ corresponds to the definition from [19], providing individual verifiability only for honest voters.
- $\mathsf{E2E}[\mathsf{iv}_\bullet, \mathsf{res}_\circ]$: in addition to corrupt voters, this notion allows the adversary to cast votes for voters who have not successfully performed the verification procedure (this is sometimes called ballot stuffing). However, for corrupt voters who successfully verified their votes, we are certain these votes are correctly counted.
- $\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$ is the weakest notion, only allowing individual verifiability for honest voters and allows ballot stuffing for honest voters who did not verify their votes.

B. *Symbolic definition of election verifiability*

We extend the symbolic verifiability definition from [6] by introducing two symbolic notions: one to make a stronger link between voter identities and their public credentials, and a second one to make a link between the public credentials and trackers used for verification. The former helps making a stronger connection to the general notion of election verifiability from Definition 3 and [6]. The latter helps modelling receipt-free voting systems like Selene.

The registration events. Universal verifiability of the tally guarantees that all the votes in the final outcome correspond to ballots recorded next to public credentials before the tally phase, i.e. we have:

$$\mathsf{BBtally} : (\mathsf{cr}_1, \mathsf{b}_1), \dots, (\mathsf{cr}_\mathsf{n}, \mathsf{b}_\mathsf{n}) \quad \mathsf{BBres} : (\mathsf{tr}_1, \mathsf{v}_1), \dots, (\mathsf{tr}_\mathsf{n}, \mathsf{v}_\mathsf{n})$$

and we can ensure that $\mathsf{v}_1, \dots, \mathsf{v}_\mathsf{n}$ is a permutation of votes encoded in $\mathsf{b}_1, \dots, \mathsf{b}_\mathsf{n}$. However, how can we ensure that $\mathsf{b}_1, \dots, \mathsf{b}_\mathsf{n}$ are ballots cast by eligible voters, i.e. that the adversary cannot do ballot stuffing? The election verifiability definition should precisely limit the influence of the adversary over the ballots in the final tally. The definition in [6] considers a notion of *corrupt credentials* in order to specify which ballots among $\mathsf{b}_1, \dots, \mathsf{b}_\mathsf{n}$ should be allowed to be cast by the adversary. This notion turns out to be distinct from the notion of *corrupt voters*, which is the more natural notion used in the general definition of election verifiability. As a consequence, the symbolic definition in [6] cannot be used to prove election verifiability. On the other hand, although it can find real attacks on verifiability in voting protocols, it can also detect false attacks. For example, if there is one corrupt voter, one corrupt registrar and the result contains one vote cast by the adversary, this should not be counted as an attack on verifiability, since the number of adversarial votes is bounded by the number of corrupt voters. However, the definition in [6] finds an attack against Belenios in this scenario, due to the fact that the corrupt registrar associates a so-called honest credential to the corrupt voter. We solve this problem by introducing events $\mathsf{Reg}(\mathsf{id},\mathsf{cr})$ that make a formal link between voter identities and the corresponding public credentials, and adding a consistency requirement for these events in the security definition.

Depending on the trust assumptions in the protocol and the corruption scenario for which verification is performed, the placement of the registration events may vary. For example, Belenios and BeleniosRF are supposed to provide verifiability if either the registrar or the voting server is honest. In the former case, we place $\mathsf{Reg}(\mathsf{id},\mathsf{cr})$ in the specification of the registrar at registration time, and in the latter case we place it in the specification of the voting server when it receives a ballot from that voter. This way we can obtain a general symbolic definition that does not have to vary according to the corruption scenario, but only requires to place the registration facts according to it. In a previous framework for symbolic election verifiability [18], distinct definitions and assumptions where devised for distinct corruption scenarios in Belenios (corrupt server or corrupt registrar). We are able to obtain security proofs of election verifiability for both of these scenarios using the same definition and assumptions. Finally, we note that in general $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$ may be recorded as an event without requiring trust assumptions, but this would require additional cryptographic techniques to hide the link between id and cr.

The linking events. To account for the fact that voters may verify their votes directly in the final tally, we need to further link their public credentials to the credentials used for verification, i.e. to their corresponding trackers. Since the tally process is universally verifiable, this link can be established without any trust assumptions, relying only on the guarantees provided by the zero-knowledge proofs. To formalise this link, we require the existence of a function that can extract a tracker from the public data that is associated to a credential if the corresponding zero-knowledge proofs are valid. Then, if $\mathsf{u}_1, \dots, \mathsf{u}_n$ is all the public data associated to a credential cr and $\mathsf{tr} = \mathsf{extract}(\mathsf{u}_1, \dots, \mathsf{u}_n)$, we add an event $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$ to record the fact that tr should be considered as associated to cr. The symbolic definition of election verifiability will ensure that this link is consistent, i.e. any credential has a unique associated tracker, and conversely. Definition 4 introduces the additional events that are required for registration and linking of credentials.

Definition 4. *A voting specification $\mathcal{S}$ is enhanced if its rules record the following events:*

- $\mathsf{Reg}(\mathsf{id}, \mathsf{cr})$*: a voter with identity* id *is associated with the public credential* cr*;*
- $\mathsf{BBreg}(\mathsf{cr})$*: the public credential* cr *is recorded on* BB *as corresponding to an eligible voter;*
- $\mathsf{BBptr}(\mathsf{cr}, \mathsf{u}_1, \dots, \mathsf{u}_n)$*: records auxiliary information relevant for tracking votes for* cr*.*

The event $\mathsf{BBptr}(\mathsf{cr}, \mathsf{u}_1, \dots, \mathsf{u}_n)$ puts together any pre-tracking data and zero-knowledge proofs related to a given credential cr. This allows for a simpler specification of the tracker extraction function in the following. For a sequence of terms $\mathsf{u}_1, \dots, \mathsf{u}_n$, we denote $\mathsf{ver}(\mathsf{u}_1, \dots, \mathsf{u}_n) = \mathsf{ok}$ if, for any term in $\mathsf{u}_1, \dots, \mathsf{u}_n$ that represents a zero-knowledge proof, the corresponding verification equation evaluates to true. We assume, implicitly, that all arguments required for the verification of proofs are also part of $\mathsf{u}_1, \dots, \mathsf{u}_n$. If all proofs related to a credential are valid, an admissible extraction function should always succeed in computing a corresponding tracker from pre-tracking data, as formalised by the following definition.

Definition 5. *An enhanced voting specification* $(\mathcal{P}, \Psi, \mathcal{E}, \Omega)$ *has an* admissible extraction function *if there is a destructor* $\mathsf{extract}$ *defined in* $\mathcal{E}$ *that, for any terms* $\mathsf{cr}, \mathsf{u}_1, \dots, \mathsf{u}_n$*, satisfies:*

$$[\ (\mathcal{P}, \Psi, \mathcal{E}, \Omega) \models \mathsf{BBptr}(\mathsf{cr}, \mathsf{u}_1, \dots, \mathsf{u}_n) \wedge \mathsf{ver}(\mathsf{u}_1, \dots, \mathsf{u}_n) = \mathsf{ok}\ ] \implies \mathsf{extract}(\mathsf{u}_1, \dots, \mathsf{u}_n) \neq \bot.$$

The satisfaction of Definition 5 can be checked automatically with ProVerif, as we do for Selene and Hyperion. For Belenios and BeleniosRF, the extraction function is trivial, as it needs to associate cr to itself.

Example 6. *In Selene, we have* $\mathsf{BBptr}(\mathsf{cr}, \langle \mathsf{tpk}, \mathsf{etr}, \mathsf{cm}, \pi_{\mathsf{ptr}}, \mathsf{pk}\rangle)$ *where* π_{ptr} *shows, in particular, that* etr *is of the form* $\mathsf{enc}(\mathsf{tr}, \mathsf{pk}, \mathsf{r})$*, for some terms* tr *and* r*. Therefore, we can define the extraction function by* $\mathsf{extract}(\mathsf{x}_1, \mathsf{enc}(\mathsf{x}, \mathsf{y}, \mathsf{z}), \mathsf{x}_2, \mathsf{x}_3, \mathsf{y}) = \mathsf{x}$.

For Selene, the tracker is a term encoded inside a ciphertext in pre-tracking information, and the extraction function simply returns this term. In Hyperion, the extraction function needs to be more involved to account for the computation that trustees should perform in order to derive trackers.

Example 7. *In Hyperion, we have* $\mathsf{BBptr}(\mathsf{cr}, \langle \mathsf{rtpk}, \pi\rangle)$*, where* $\pi = \mathsf{prmix}(\mathsf{M}_{\mathsf{in}}, \mathsf{s}, \mathsf{M}_{\mathsf{out}})$ *shows that the set of final trackers is equal to the set of pre-trackers where each element* rtpk *is raised to the same power* s*. Therefore, we can define the extraction function by* $\mathsf{extract}(\mathsf{x}, \mathsf{prmix}(\mathsf{x}_1, \mathsf{y}, \mathsf{x}_2)) = \mathsf{x}^{\mathsf{y}}$.

One may wonder if, in some sense, a wrong extraction function could be defined, that is still admissible for a protocol. Our symbolic definition of verifiability, if satisfied by a protocol and extraction function specification, will ensure that the extraction function determines a bijection between the set of credentials and the set of trackers, and furthermore that voters derive trackers that are consistent with this function. We show that these two properties (together with others from our definition) are sufficient to ensure end-to-end election verifiability - regardless of the actual definition of the extraction function. Another observation is that the extraction function is not efficiently computable in general, as can be seen from the previous examples. Indeed, as the corresponding tracker points directly to the vote in Selene and Hyperion, this would violate privacy. In Belenios and BeleniosRF this is not a problem, since the tracker points to the ballot, and not to the vote.

Definition 6. *Let $\mathcal{S}$ be an enhanced voting specification with an admissible extraction function. We define the event* $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$ *to record that* tr *can be extracted from pre-tracking information associated to* cr*. Formally, for a trace τ of $\mathcal{S}$, we have* $\tau \models \mathsf{BBptr}(\mathsf{cr}, \mathsf{u}_1, \dots, \mathsf{u}_n) \wedge \mathsf{ver}(\mathsf{u}_1, \dots, \mathsf{u}_n) = \mathsf{ok} \wedge \mathsf{extract}(\mathsf{u}_1, \dots, \mathsf{u}_n) = \mathsf{tr} \implies \tau \models \mathsf{Link}(\mathsf{cr}, \mathsf{tr})$.

In ProVerif, the event Link satisfying the definition can be specified in a process that applies $\mathsf{extract}$ and adds Link as soon as the required information from BBptr is available. Definition 6 formalises the association between credentials and trackers that is ensured by any admissible extraction function. As discussed above, our symbolic definition of verifiability, relying on trace formulas from Figure 4, will ensure additional constraints for the event Link.

Basic properties (for $\triangledown, \diamond \in \{\circ, \bullet\}$)

$\Phi^{\triangledown}_{\mathsf{iv1}}$: $\mathsf{Ver}^{\triangledown}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \wedge \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{BBres}(\mathsf{tr}, \mathsf{v}') \Rightarrow \mathsf{v} = \mathsf{v}'$

$\Phi^{\triangledown}_{\mathsf{iv2}}$: $\mathsf{Ver}^{\triangledown}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \wedge \mathsf{Ver}^{\triangledown}(\mathsf{id}', \mathsf{cr}, \mathsf{v}') \Rightarrow \mathsf{id} = \mathsf{id}'$

$\Phi^{\triangledown}_{\mathsf{iv3}}$: $\mathsf{Ver}^{\triangledown}(\mathsf{id}, \mathsf{cr}, \mathsf{v}) \Rightarrow \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{BBreg}(\mathsf{cr})$

Φ_{eli} : $\mathsf{BBres}(\mathsf{tr}, \mathsf{v})$
$\quad \Rightarrow \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{BBreg}(\mathsf{cr}) \wedge (\ \mathsf{Reg}(\mathsf{id}, \mathsf{cr}) \vee \mathsf{v} = \bot\ )$

$\Phi^{\diamond}_{\mathsf{res}}$: $\mathsf{BBres}(\mathsf{tr}, \mathsf{v}) \wedge \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{Reg}(\mathsf{id}, \mathsf{cr})$
$\quad \Rightarrow \mathsf{Vote}(\mathsf{id}, \mathsf{v}) \vee \mathsf{Adv}^{\diamond}(\mathsf{id}) \vee \mathsf{v} = \bot$

(formulas $\mathsf{Ver}^{\triangledown}$ and $\mathsf{Adv}^{\diamond}$ introduced in Definition 2)

Consistency properties

Φ_{reg1} : $\mathsf{Reg}(\mathsf{id}, \mathsf{cr}) \wedge \mathsf{Reg}(\mathsf{id}', \mathsf{cr}) \Rightarrow \mathsf{id} = \mathsf{id}'$

Φ_{reg2} : $\mathsf{Reg}(\mathsf{id}, \mathsf{cr}) \wedge \mathsf{Reg}(\mathsf{id}, \mathsf{cr}') \Rightarrow \mathsf{cr} = \mathsf{cr}'$

Φ_{link1} : $\mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{Link}(\mathsf{cr}', \mathsf{tr}) \Rightarrow \mathsf{cr} = \mathsf{cr}'$

Φ_{link2} : $\mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{Link}(\mathsf{cr}, \mathsf{tr}') \Rightarrow \mathsf{tr} = \mathsf{tr}'$

Φ_{one} : $\mathsf{BBres}(\mathsf{tr}, \mathsf{v})\ @i \wedge \mathsf{BBres}(\mathsf{tr}, \mathsf{v}')\ @j \Rightarrow i = j$

Fig. 4. Formulas for symbolic election verifiability (Definition 7).

Definition 7. *Consider the formulas from Figure 4. For* $(\triangledown, \diamond) \in \{\circ, \bullet\} \times \{\circ, \bullet\}$, *we let*

$$\begin{aligned}\mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}] \quad = \quad & \Phi^{\triangledown}_{\mathsf{iv1}} \wedge \Phi^{\triangledown}_{\mathsf{iv2}} \wedge \Phi^{\triangledown}_{\mathsf{iv3}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi^{\diamond}_{\mathsf{res}} \wedge \\ & \Phi_{\mathsf{reg1}} \wedge \Phi_{\mathsf{reg2}} \wedge \Phi_{\mathsf{link1}} \wedge \Phi_{\mathsf{link2}} \wedge \Phi_{\mathsf{one}}\end{aligned}$$

An enhanced e-voting specification $\mathcal{S}$ satisfies $[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$*-symbolic election verifiability iff it has an admissible extraction function and* $\mathcal{S} \vDash \mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$.

The formula $\Phi^{\triangledown}_{\mathsf{iv1}}$ represents individual verifiability, ensuring that all verified votes should be part of the final outcome, i.e. the vote recorded for that tracker is the expected one. The formula $\Phi^{\triangledown}_{\mathsf{iv2}}$ requires that no clash should occur on the public credentials of voters who verified their vote. The formula $\Phi^{\triangledown}_{\mathsf{iv3}}$ makes a link from the voter credential to a vote verification tracker tr. The formula Φ_{eli} ensures that trackers on BBres are connected through Link to registered eligible credentials. The formula $\Phi^{\diamond}_{\mathsf{res}}$ ensures that there is no ballot stuffing. It relies on Link and Reg events to determine what is the voter identity associated to each tracker in the outcome. Our constraints related to Link and Reg are completed by the consistency properties, ensuring that each credential has a unique associated tracker and voter identity, and conversely. Finally, the property Φ_{one} ensures that there is at most one vote counted for each tracker, and therefore for each credential and each voter.

C. Symbolic E2E verifiability implies general E2E verifiability

In this section we prove that, if an enhanced voting specification satisfies $\mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$, then it also satisfies $\mathsf{E2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$. We recall the general flow of electronic voting protocols that we consider in this paper: a set of public credentials is determined at registration and published on BBreg; ballots from corresponding voters are collected, and the ballot to be tallied for each credential is recorded on BBtally; the final result is computed from the ballots in BBtally. Assuming there are n public credentials, a snapshot of the end-to-end run of the election can then be represented by:

$$\begin{array}{rccc} \mathsf{BBreg}: & \mathsf{cr}_1 & \ldots & \mathsf{cr}_n \\ \mathsf{BBtally}: & (\mathsf{cr}_1, \mathsf{b}_1) & \ldots & (\mathsf{cr}_n, \mathsf{b}_n) \\ \mathsf{BBres}: & (\mathsf{tr}_1, \mathsf{v}_1) & \ldots & (\mathsf{tr}_n, \mathsf{v}_n) \end{array}$$

where $\mathsf{b}_i = \bot$ and $\mathsf{v}_i = \bot$ in case of abstention, invalid ballot casting or ballot blocking by the adversary. Recall that in Helios, Belenios and BeleniosRF, BBres is defined implicitly from BBtally taking $\mathsf{tr}_i = \mathsf{cr}_i$ and $\mathsf{v}_i = \mathsf{open}(\mathsf{b}_i)$. In order to prove our result, we will make one additional assumption about the execution trace of a voting system, namely that it contains the final state of the bulletin board. According to our observations above, the fact that a trace contains the final state of the bulletin board can be formalised by the formula

$$\Psi_{\mathsf{E2E}}: \quad |\{\mathsf{cr} \mid \tau \vDash \mathsf{BBreg}(\mathsf{cr})\}| = |\{\mathsf{tr} \mid \exists \mathsf{v}.\ \tau \vDash \mathsf{BBres}(\mathsf{tr}, \mathsf{v})\}|$$

stating that the number of trackers (or credentials) recorded on the final result bulletin board is equal to the number of registered credentials. A trace that satisfied Ψ_{E2E} is called an *end-to-end voting trace*. We note that the assumption of a complete trace is inherent to the property of election verifiability, which cannot be evaluated until the election authorities commit to the final result.

Theorem 1. *For every voting specification $\mathcal{S}$, symbolic election verifiability implies general election verifiability for end-to-end voting traces. That is, for any trace τ of $\mathcal{S}$, we have*

$$\tau \vDash \mathsf{SE2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}] \wedge \tau \vDash \Psi_{\mathsf{E2E}} \Longrightarrow \tau \Vdash \mathsf{E2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$$

The proof is given in Section A of the appendix.

V. Verifiability Analysis of BeleniosRF and Selene

In this section, we present the verifiability analysis with ProVerif of BeleniosRF, Selene, SeleneRF, and Hyperion, which are introduced in Section III, with respect to the definition introduced in Section IV. We start by introducing the common components used for the formalisation of these protocols (voting server, registrar, voting platform, etc.) and the corruption models that we consider. We also present the equational theories for cryptographic primitives. Then, we provide comparative verification results for these systems obtained with ProVerif, and explain the attacks and proofs found. We show some parts from the specification of each protocol and refer to the ProVerif code attached as supplementary material for full details.

A. Common protocol structure

We consider the following parties common to all protocols:
- *Administrator* A determines the list of candidates $\mathsf{v}_1, ..., \mathsf{v}_k$ and voters $\mathsf{id}_1, ..., \mathsf{id}_n$.
- *Trustees* T generate the election keys $(\mathsf{pk}, \mathsf{sk})$ and publish pk on BB.
- *Registrar* VR generates a signing key pair $(\mathsf{spk}, \mathsf{ssk})$ for each

voter and publishes cr = spk on BBreg. While Selene and its variants do not explicitly mention a registrar, they assume that voters are registered with public keys before the start of the protocol. We consider that this registration is performed relying on a registrar. The case when voters register themselves is equivalent to the case of an honest registrar (if they use an honest device) or a corrupt registrar (if they use a corrupt one).
- *Voting server* VS is responsible for publishing authenticated valid ballots on BBcast in addition to generating a password pwd for each voter. The ballots to be tallied are put on BBtally.
- *Voters* V use their voting platform VP to construct and cast a ballot; they verify their ballot or vote on BB.
- *Voting platform* VP constructs a ballot; encrypting the desired vote, signing it and attaching a proof, and sends it to VS with authentication information.

Setup. Parties generate election parameters as described above, with public elements recorded in corresponding bulletin board tables: BBkey for the public key of the election and BBreg for public voter credentials. Each voter records the credential tuple $\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{ssk} \rangle$.

Voting. Ballots are constructed by VP and posted to VS. VP takes as input the vote v and voter credentials. The ballot b has three parts:

$$\begin{aligned} \mathsf{c} &= \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r}) && \textit{(encryption of } \mathsf{v}\textit{)}, \\ \mathsf{s} &= \mathsf{sign}(\mathsf{c}, \mathsf{ssk}) && \textit{(signature)}, \\ \pi &= \mathsf{pr}(\mathsf{c}, \mathsf{v}, \mathsf{r}, \ell) && \textit{(zk proof of knowledge of } \mathsf{v}, \mathsf{r}\textit{)}. \end{aligned}$$

We denote the operation of generating r and constructing b by $\mathsf{Gen}(\mathsf{v}, \mathsf{pk}, \mathsf{ssk})$. In BeleniosRF, the label ℓ in π is equal to cr and b is rerandomized before being published; in Selene and Hyperion, ℓ is empty. Authentication is modelled relying on a hash function and a term $\mathsf{h}(\mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b})$, which allows VS to verify that b was sent by a voter knowing the pwd for id. We note that this is an abstraction of any underlying authentication process, which allows the server to validate the identity of the voter sending the ballot.

Tally and verification. The zk proofs and signatures are verified before tally, which is done by verifiably decrypting the ballots from BB, with additional procedures for deriving trackers in Selene and Hyperion. Individual verification procedures are specific in each protocol, as shown later.

B. Corruption models

Other than the administrator, any party may be corrupted:

– Corrupt trustees: $\mathcal{A}$ can choose the secret key and perform any operations assigned to trustees in the corresponding protocol.
– Corrupt registrar: $\mathcal{A}$ is allowed to choose the public and private credentials for each voter.
– Corrupt server: $\mathcal{A}$ chooses voter passwords and accepts any ballot without authentication checks.
– Corrupt voter: all credentials are leaked to $\mathcal{A}$.
– Corrupt platform: reveals to $\mathcal{A}$ the voter credentials (except the trapdoor secret key in Selene and Hyperion), and $\mathcal{A}$ chooses the ballot that the voter will cast.

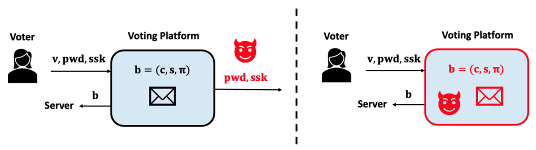

Fig. 5. Corrupt voter (left) vs corrupt platform (right).

```
let VotingPlatform(v, pk) =          let VotingServer(pk) =
    get Cred(id, cr, ssk) in             get BBreg(cr) in
    get Pwd(= id, pwd) in                get Pwd(id, pwd) in
    let b = Gen(v, pk, ssk) in           in(pub, (= id, = cr, b, a);
    let a = h(id, pwd, cr, b) in         if h(id, pwd, cr, b) = a then
    event Vote(id, cr, v);               if Verify(b, cr) = ok then
    insert Voted(id, cr, v);             let b′ = Rand(b, pk) in
    out(pub, (id, cr, b, a)).            insert BBcast(cr, b′).
 let VoterVer =
     get Voted(id, cr, v); BBtally(= cr, b′) in
     if Verify(b′, cr) = ok then event Verified(id, cr, v).
```

Fig. 6. Voting and verification processes for BeleniosRF.

Figure 5 illustrates the difference between a corrupt V and a corrupt VP. In the latter, we let the adversary construct the ballot as it wishes, providing $\mathcal{A}$ with credentials required to do so. Our notion of a corrupt VP is much stronger than the one considered in [6]: their model only allows $\mathcal{A}$ to choose the randomness used for the encryption; our model allows $\mathcal{A}$ to construct the whole ballot. Note that, in addition to voting credentials $\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{ssk} \rangle$, a voting protocol may provide voters with separate secret material required for verification. This is the case in Selene and Hyperion, where voters have an additional trapdoor secret key required to derive trackers. In the case of a corrupt VP, contrary to that of a corrupt V, we assume these credentials are kept secret. In practice, this means that the verification device should be different from the voting platform, or that trapdoor keys are generated and stored on special trusted hardware. A corruption model (or scenario) is defined by a subset of the corruption abilities described above. In Table I and Table II, we list all the considered corruption models and verification results found with ProVerif. In all cases, the communication network and voters can be corrupt. The ProVerif code is available online [1] and the running time for each scenario is given in its corresponding file. For BeleniosRF, we consider the trivial extraction function that associates cr to itself; for Selene and Hyperion, we consider extraction functions like in Example 6 and Example 7. For all cases, we consider the revote policy that counts the last vote cast by each voter.

C. BeleniosRF specification and verification

To model the cryptographic primitives of BeleniosRF, in addition to equations from Example 2, we consider the equations (4)-(6) below to model the properties that allow ciphertext rerandomization (4) and the adaptation of the corresponding signature (5) and zk proof (6). Equation (7) models the verification of a proof of correct decryption, provided by

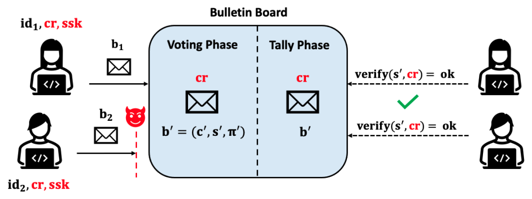

Fig. 7. Clash attack by corrupt registrar in BeleniosRF.

trustees in all protocols. Part of the specification for BeleniosRF is presented in Figure 6. It is similar to Example 1, the main difference being the randomization of the ballot b into $\mathsf{b}' = \langle \mathsf{c}', \mathsf{s}', \pi' \rangle$ before casting on BB. We denote this operation, together with the generation of r', by $\mathsf{Rand}(\mathsf{b}, \mathsf{pk})$. The voter verifies b' on BB by verifying the signature s' and the zero-knowledge proof π'; we denote the verification result by $\mathsf{Verify}(\mathsf{b}, \mathsf{cr})$.

$$\begin{aligned} \mathsf{c}' &= \mathsf{renc}(\mathsf{c}, \mathsf{pk}, \mathsf{r}') && \textit{(re-encryption)},\\ \mathsf{s}' &= \mathsf{resign}(\mathsf{s}, \mathsf{pk}, \mathsf{r}') && \textit{(adaptation of signature)},\\ \pi' &= \mathsf{repr}(\mathsf{pr}, \mathsf{pk}, \mathsf{r}', \mathsf{cr}) && \textit{(adaptation of proof)}. \end{aligned}$$

$$\begin{aligned} &(4) && \mathsf{renc}(\mathsf{enc}(x, y, z_1), y, z_2) = \mathsf{enc}(x, y, z_2)\\ &(5) && \mathsf{resign}(\mathsf{sign}(\mathsf{enc}(x, y, z_1), w), y, z_2) = \mathsf{sign}(\mathsf{enc}(x, y, z_2), w)\\ &(6) && \mathsf{repr}(\mathsf{pr}(\mathsf{enc}(x, y, z_1), x, z_1, w), y, z_2, w))\\ & && = \mathsf{pr}(\mathsf{enc}(x, y, z_2), x, z_2, w)\\ &(7) && \mathsf{ver}'(\mathsf{pr}'(\mathsf{enc}(x, \mathsf{pk}(y), z), x, y), \mathsf{enc}(x, \mathsf{pk}(y), z), x, \mathsf{pk}(y)) = \mathsf{ok} \end{aligned}$$

Attacks. We find an attack against individual verifiability in the case of a corrupt registrar, i.e. the scenario $\mathcal{A}_3$. The weakness comes from the fact that individual verification does not directly check the ballot posted by voter, but its randomization. The properties $\Phi^{\triangledown}_{\mathsf{iv1}}$ and $\Phi^{\triangledown}_{\mathsf{iv2}}$ in $\mathsf{E2E}[\mathsf{iv}_{\triangledown}, \mathsf{res}_{\diamond}]$ are violated for $(\triangledown, \diamond) \in \{\circ, \bullet\} \times \{\circ, \bullet\}$, because of a clash attack as illustrated in Figure 7. It shows a corrupt registrar communicating the same signature key pair $\langle \mathsf{cr}, \mathsf{ssk} \rangle$ to two honest voters id_1 and id_2. Assume the two voters cast their ballots separately relying on their credential cr. A ballot b from one of the voters is accepted, randomized, and published on BB. For the second voter, we assume the ballot is dropped by the network, before the voter noticing it. When verifying their ballots (possibly later when the voting phase is finished), both voters can verify that, for the tuple $(\mathsf{cr}, \mathsf{b}')$ on BB, b' is the rerandomization of a ballot submitted with their credential. They both verify the attached signature and zk proof since they share the same credentials. However, only one vote will be counted for these two voters. A similar attack is possible in the case of a corrupt voter, even when all other parties are honest (these are the two lines filled with ✗ for the case of $\mathsf{iv}_{\bullet}$ in Table I). It is sufficient for the communication network to block the ballot of that voter, and for the adversary to cast a different ballot with the credential of that voter. In Belenios this does not lead to an attack, but in BeleniosRF it does.

Related attacks. BeleniosVS [18] is a variant of Belenios that uses the same ballot rerandomization feature as BeleniosRF, in order to achieve better privacy in the case of a dishonest voting platform. In consequence, a similar attack as ours applies to

TABLE I
VERIFICATION RESULTS FOR BELENIOS AND BELENIOSRF

Corruption Models		$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_4$
Trustees		H	C	C	C
Registrar		H	H	C	H
Server		H	H	H	C
Voting Platform		H	H	H	H
Belenios*	$\mathsf{E2E}[\mathsf{iv}_{\bullet}, \mathsf{res}_{\bullet}]$	✓	✓	✓	✓
	$\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\bullet}]$	✓	✓	✓	✓
	$\mathsf{E2E}[\mathsf{iv}_{\bullet}, \mathsf{res}_{\circ}]$	✓	✓	✓	✓
	$\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$	✓	✓	✓	✓
BeleniosRF†	$\mathsf{E2E}[\mathsf{iv}_{\bullet}, \mathsf{res}_{\bullet}]$	✗	✗	✗	✗
	$\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\bullet}]$	✓	✓	✗	✓
	$\mathsf{E2E}[\mathsf{iv}_{\bullet}, \mathsf{res}_{\circ}]$	✗	✗	✗	✗
	$\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\circ}]$	✓	✓	✗	✓

$*$: no revoting or voter verification in the tally phase
$\dagger$: no revoting

BeleniosVS in the case of a corrupted registrar, and the authors of [18] have already observed this. As a fix, they propose that the voting server sends an acknowledgement to the voter when the ballot is successfully cast. In an attack scenario similar to the one above (from Figure 7), this should counter the dropping of the second ballot (b_2) by the network, since the voter id_2 would notice that no acknowledgement was sent. We plan to investigate in future work if a similar fix could help in BeleniosRF, and if it still protects against a stronger adversarial model in BeleniosVS. Indeed, in our analysis, we consider a strong attacker model that controls the secure channel that is established between the voting platform and the voting server. A simple acknowledgement would not help against such an attacker, since the attacker could impersonate the voting server and spoof the acknowledgement.

Proofs. BeleniosRF satisfies election verifiability, i.e. $\mathsf{E2E}[\mathsf{iv}_{\circ}, \mathsf{res}_{\diamond}]$ for $\diamond \in \{\circ, \bullet\}$, when the registrar, the voter and the voting platform are honest. In this case, the trustees and the voting server can be fully corrupt. All attacks and proofs for Belenios and BeleniosRF are found automatically with ProVerif within several seconds.

D. Selene and Hyperion specification and verification

To model the cryptographic primitives of Selene, in addition to equations from Example 2, and equations (4) and (7) from above, we consider equations for additional zk proofs, for trapdoor commitments and for deriving a commitment from a ciphertext. We omit from the description the equations for zk proofs, as they are similar to (3) and (7), where terms are modified accordingly to account for the desired relation. To model trapdoor commitments, we use a theory previously considered in [11], [25] for verifying privacy-type properties. The main property is that a commitment can be opened with the corresponding trapdoor secret key, as in equation (8) below. Furthermore, equation (9) models the fact that a trapdoor secret key along with the randomness from the commitment can be used to generate randomness that opens that commitment to

Selene

```
Voter(id, cr) generates :
  Td(id, tpk, tsk)
Trustees generate :
  BBptr(cr, tpk, enc(tr, pk, r),
     com(tr, tpk, r_T), pr_ptr, pk)
  BBres(tr, v), Com(cr, r_T)
let VoterVerification =
  get Voted(id, cr, v)
  BBptr(= cr, x_1, x_2, y, x_3, x_4)
  BBres(tr, = v), Com(= cr, r_T)
  Td(id, tpk, tsk) in
  if open(y, tsk, r_T) = tr then
  event Verified(id, cr, v).
```

Hyperion

```
Voter(id, cr) generates :
  Td(id, tpk, tsk)
  where tpk = g^tsk
Trustees generate :
  BBptr(cr, tpk^r, pr_exp)
  BBres(tpk^{s_i}, v), Exp(cr, g^{s_i})
let VoterVerification =
  get Voted(id, cr, v)
  BBres(tr, = v)
  Exp(= cr, g^{s_i})
  Td(id, tpk, tsk) in
  if (g^{s_i})^tsk = tr then
  event Verified(id, cr, v).
```

Fig. 8. Tracker generation and voter verification in Selene and Hyperion.

any desired value. As shown in [11], two more equations need to be added in order to make this theory convergent for use in ProVerif, which we will omit here. Equation (10) models a relation between ElGamal and Pedersen commitments, which allows trustees to compute a commitment to a tracker from an encryption of that tracker, without decrypting it. This operation relies on homomorphically combining the encryption of a tracker x with the encryption of a term w^{z}, where w will play the role of the public trapdoor key, and z will play the role of the randomness in the commitment. The corresponding secret decryption key is needed to obtain the commitment.

(8) $\mathsf{open}(\mathsf{com}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{z}), \mathsf{y}, \mathsf{z}) = \mathsf{x}$
(9) $\mathsf{com}(\mathsf{x}_2, \mathsf{pk}(\mathsf{y}), \mathsf{fake}(\mathsf{x}_1, \mathsf{x}_2, \mathsf{y}, \mathsf{z})) = \mathsf{com}(\mathsf{x}_1, \mathsf{pk}(\mathsf{y}), \mathsf{z})$
(10) $\mathsf{ecom}(\mathsf{enc}(\mathsf{x}, \mathsf{pk}(\mathsf{y}), \mathsf{r}), \mathsf{enc}(\mathsf{w}^{\mathsf{z}}, \mathsf{pk}(\mathsf{y}), \mathsf{r}'), \mathsf{y}) = \mathsf{com}(\mathsf{x}, \mathsf{w}, \mathsf{z})$

TABLE II
VERIFICATION RESULTS FOR SELENE, SELENERF AND HYPERION

Corruption Models	$\mathcal{A}'_1$	$\mathcal{A}'_2$	$\mathcal{A}'_3$	$\mathcal{A}'_4$	$\mathcal{A}'_5$	$\mathcal{A}'_6$	$\mathcal{A}'_7$
Trustees	H	C	C	C	C	C	C
Registrar	H	H	C	H	H	C	H
Server	H	H	H	C	H	H	C
Voting Platform	H	H	H	H	C	C	C
$\mathsf{E2E}[\mathsf{iv}_\bullet, \mathsf{res}_\bullet]$	✓	✗	✗	✗	✗	✗	✗
$\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$	✓	✓	✓	✓	✗	✗	✗
$\mathsf{E2E}[\mathsf{iv}_\bullet, \mathsf{res}_\circ]$	✓	✗	✗	✗	✗	✗	✗
$\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$	✓	✓	✓	✓	✓	✓	✓

Part of the formal specification of parties in Selene is given in Figure 8. We use tables for storing voter credentials and keys, and also for the private channel Com used by the trustees to communicate commitment randomness to voters. Trackers are generated by trustees, encrypted with the public key of the election and transformed to derive the corresponding commitments for the public trapdoor key of voters, relying on equation (10). All this pre-tracking information is then stored in BBptr. Along with the ballot, it is used by trustees for tally and by voters to derive their trackers, after receiving the corresponding randomness from trustees. We assume trustees publish zk proofs of correctness for their computation and that these are verified for all credentials and ballots tallied.

Attacks. A trust assumption in Selene is that the voters performing verification are honest. On the other hand, the trustees can be fully corrupt. If both the verifying voter and the trustees are corrupt, there is a trivial attack against election verifiability. The main idea is that trustees can use the trapdoor secret key of the voter to compute fake randomness that opens their commitment to a different tracker. If that tracker points to the same vote as chosen by the voter, this results in a clash attack. This scenario is formally represented by the properties $\mathsf{iv}_\bullet$ (corrupt voters) and attackers $\mathcal{A}'_2 - \mathcal{A}'_7$ (corrupt trustees) in Table II. Another notable attack is ballot stuffing (i.e. attack against $\mathsf{iv}_\bullet$) when the voting platform is corrupt. Indeed, in that case, the adversary can replace the vote of the voter with any other vote, as soon as the voter does not verify their vote. This shows that the tracking mechanism in Selene protects voters agains corrupt platforms only if they verify their votes, as expected.

Proofs. Notable cases are the positive results in corruption scenarios $\mathcal{A}'_3, \mathcal{A}'_4$ and $\mathcal{A}'_6, \mathcal{A}'_7$. Assuming that the voting platform is honest, the former show that the (standard) verifiability property $\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\bullet]$ is true if either the registrar or the voting server is honest, like in Belenios. Furthermore, if the voting platform is corrupt, the results for $\mathcal{A}'_6, \mathcal{A}'_7$ show that the weaker notion $\mathsf{E2E}[\mathsf{iv}_\circ, \mathsf{res}_\circ]$ still holds, meaning that ballot stuffing is possible only if voters have not verified their votes.

E. Hyperion and SeleneRF

To model the cryptographic primitives of Hyperion, one set of new equations that needs to be added are for handling the specific zk proofs, related to exponentiation instead of commitment and encryption. We also need to consider a model of exponentiation that is more general than the usual model used in ProVerif, which only allows to capture Diffie-Hellman-like interactions with two exponentiations. We need three exponentiations, since trackers for voters are generated through: $\mathsf{tr}_i = ((\mathsf{g}^{\mathsf{tsk}_i})^{\mathsf{r}_i})^{\mathsf{s}}$. For this, we consider the equations (11) and (12) from below. We note that this theory is still incomplete, as in general one needs to cover any number of exponentiations in order to reason about security, and not only to be able to execute the protocol.

(11) $(\mathsf{g}^{\mathsf{x}})^{\mathsf{y}} = (\mathsf{g}^{\mathsf{y}})^{\mathsf{x}}$ (13) $\mathsf{verify}(\mathsf{prexp}(\mathsf{x}, \mathsf{y}, \mathsf{x}^{\mathsf{y}}), \mathsf{x}, \mathsf{x}^{\mathsf{y}}) = \mathsf{ok}$
(12) $((\mathsf{g}^{\mathsf{x}})^{\mathsf{y}})^{\mathsf{z}} = ((\mathsf{g}^{\mathsf{y}})^{\mathsf{z}})^{\mathsf{x}}$ (14) $\mathsf{extract}(\mathsf{x}, \mathsf{prexp}(\mathsf{x}, \mathsf{y}, \mathsf{z})) = \mathsf{x}^{\mathsf{y}}$

In order to add the event $\mathsf{Link}(\mathsf{cr}, \mathsf{tr})$, according to Definition 6, Definition 5 and Example 7, we need to verify an exponentiation mixnet proof. Since we consider an unbounded number of voters, the verification of such proofs cannot be directly modelled in ProVerif. Instead, we consider an abstraction of mixnets that takes each tracker individually, exponentiates it to a secret power s, and outputs the resulting tracker together with a proof of exponentiation, whose verification is modelled by equation (13). Then, we can rely on the equation (14) for extraction. This abstraction of mixnets is sufficient in our case since we are not considering privacy properties. In fact, this model represents a weaker guarantee than a mixnet, which is still sufficient for election verifiability of Selene and Hyperion. This (malicious) mixnet is allowed to drop some of its inputs (these inputs will be considered as dropped ballots by the definition). What is important for the verifiability proof, is that the elements that do get output by the mixnet are in the prescribed relation with respect to the corresponding input elements. This is ensured by equation (13) in case of an exponentiation mixnet, and a similar equation for the case of a re-encryption mixnet. We also specify SeleneRF, that combines the models for ballot randomization from BeleniosRF and the tracking mechanism in Selene. For these models of Hyperion and SeleneRF, we obtain the same analysis results as for Selene.

VI. Conclusion and future work

We have made several contributions related to automated verification of election verifiability in electronic voting protocols: we have proposed a new symbolic definition improving on the state-of-the-art, we have applied it to verify several protocols that were not verified before, and we have found new attacks and security proofs as a result. Our symbolic definition allows to cover many corruption scenarios for various protocols in a generic way, can be applied to receipt-free protocols and we have proved that it implies a general, realistic notion of election verifiability. We have found new attacks on BeleniosRF in scenarios that were previously thought to be secure and performed automated security proofs for Selene and its variants.

In future work, we plan to complement our analysis of receipt-free systems by verifying the property of receipt-freeness, aiming for a global security analysis including both verifiability and receipt-freeness. We have seen that one of our attacks on BeleniosRF is very similar to an attack against BeleniosVS that was discussed [18], which also propose a simple countermeasure based on an acknowledgement message from the voting server. We plan to apply our framework for verifying if a refinement of this countermeasure would be sufficient against the stronger network attacker that we consider in this paper.

Acknowledgment

This work was supported by the Luxembourg National Research Fund (FNR) and the Research Council of Norway for the joint INTER project SURCVS, ref. 11747298.

References

[1] Additional material: specifications in ProVerif. https://github.com/sbaloglu/proverif-codes.

[2] Belenios - Verifiable online voting system. https://belenios.org/.

[3] Helios - Verifiable online elections. https://heliosvoting.org/.

[4] 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.

[5] Michael Backes, Cătălin Hriţcu, 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.

[6] Sevdenur Baloglu, Sergiu Bursuc, Sjouke Mauw, and Jun Pang. Election verifiability revisited: Automated security proofs and attacks on Helios and Belenios. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–15. IEEE, 2021.

[7] 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.

[8] 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.

[9] 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.

[10] Olivier Blazy, Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Signatures on randomizable ciphertexts. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, volume 6571 of *Lecture Notes in Computer Science*, pages 403–422. Springer, 2011.

[11] Alessandro Bruni, Eva Drewsen, and Carsten Schürmann. Towards a mechanized proof of Selene receipt-freeness and vote-privacy. 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 110–126. Springer, 2017.

[12] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In *23rd ACM Conference on Computer and Communications Security (CCS'16)*, pages 1614–1625, Vienna, Austria, October 2016. ACM.

[13] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84âĂŞ90, feb 1981.

[14] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On some incompatible properties of voting schemes. In 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*, pages 191–199. Springer, 2010.

[15] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 354–368, 2008.

[16] Véronique Cortier, Constantin Cătălin Drăgan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *IEEE Symposium on Security and Privacy*, pages 993–1008, 2017.

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

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

[19] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. Election verifiability for Helios under weaker trust assumptions. In *ESORICS - European Symposium on Research in Computer Security*, pages 327–344. Springer International Publishing, 2014.

[20] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. SoK: Verifiability notions for e-voting protocols. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 779–798. IEEE Computer Society, 2016.

[21] 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.

[22] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 409–423. ACM, 2017.

[23] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, pages 17–32. IEEE, 2020.

[24] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 297–311. IEEE Computer Society, 2011.

[25] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.*, 17(4):435–487, 2009.

[26] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2008.

[27] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005*, pages 61–70, 2005.

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

[29] 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.

[30] 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.

[31] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. *What did I really vote for? On the usability of verifiable e-voting schemes*, page 1âĂŞ13. Association for Computing Machinery, New York, NY, USA, 2018.

[32] 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.

[33] 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.

[34] Peter Y. A. Ryan, Peter B. Roenne, and Simon Rastikian. Hyperion: An enhanced version of the Selene end-to-end verifiable voting scheme. In *Sixth International Joint Conference on Electronic Voting E-Vote-ID*. University of Tartu Press, 2021.

[35] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 2016.

Appendix A
Proof of Theorem 1

Theorem 1. *For every voting specification $\mathcal{S}$, symbolic election verifiability implies general election verifiability for end-to-end voting traces. That is, for any trace τ of $\mathcal{S}$, we have*

$$\tau \vDash \mathsf{SE2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond] \ \wedge\ \tau \vDash \Psi_{\mathsf{E2E}} \Longrightarrow \tau \Vdash \mathsf{E2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$$

Proof. Assume $\tau \vDash \mathsf{SE2E}[\mathsf{iv}_\triangledown, \mathsf{res}_\diamond]$ and $\tau \vDash \Psi_{\mathsf{E2E}}$ for an end-to-end voting trace τ of a voting specification.

(1) Let $\{\mathsf{cr} \mid \tau \vDash \mathsf{BBreg(cr)}\}$ be the set of all registered public credentials. Assume there are n such credentials in total. By $\tau \vDash \Psi_{\mathsf{E2E}}$, we have $\mathsf{n} = |\{\mathsf{tr} \mid \exists \mathsf{v}.\, \tau \vDash \mathsf{BBres(tr, v)}\}|$, implying that there are precisely n trackers $\{\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{n}\}$ for which $\tau \vDash \mathsf{BBres(tr, v)}$ is true for some v. Then, by $\tau \vDash \Phi_{\mathsf{eli}}$, each tracker $\mathsf{tr_i} \in \{\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{n}\}$ is linked to a credential $\mathsf{cr_i} \in \mathsf{BBreg}$, i.e. we have $\tau \vDash \mathsf{Link(cr_i, tr_i)}$. From the consistency properties $\tau \vDash \Phi_{\mathsf{link1}} \wedge \Phi_{\mathsf{link2}}$, for any $\tau \vDash \mathsf{Link(cr_i, tr_i)} \wedge \mathsf{Link(cr_j, tr_j)}$, we have $\mathsf{tr_i} \neq \mathsf{tr_j}$ and $\mathsf{cr_i} \neq \mathsf{cr_j}$ for any $\mathsf{i} \neq \mathsf{j}$. Furthermore, from Φ_{one}, we know that there is at most one occurrence of the event $\tau \vDash \mathsf{BBres(tr, v)}$ for each tracker. Therefore, we can deduce that there is one-to-one correspondence between the list of trackers in the outcome, i.e. $\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{n}$, and the credentials that are linked to those. Let us call these credentials $\mathsf{cr}_1, ..., \mathsf{cr}_\mathsf{n}$.

(2) Without loss of generality, let $\{\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{k}\}$ be the subset of trackers for which $\mathsf{v_i} \neq \bot$ for all $1 \leq \mathsf{i} \leq \mathsf{k}$. By $\tau \vDash \Phi_{\mathsf{eli}}$, for each tracker $\mathsf{tr_i}$ in this subset, we have $\tau \vDash \mathsf{Link(cr_i, tr_i)} \ \wedge\ \mathsf{Reg(id_i, cr_i)}$. From the consistency properties $\tau \vDash \Phi_{\mathsf{reg1}} \ \wedge\ \Phi_{\mathsf{reg2}}$, for any $\tau \vDash \mathsf{Reg(id_i, cr_i)} \wedge \mathsf{Reg(id_j, cr_j)}$, we have $\mathsf{id_i} \neq \mathsf{id_j}$ and $\mathsf{cr_i} \neq \mathsf{cr_j}$, for any $\mathsf{i} \neq \mathsf{j}$. This implies that there are exactly $\mathsf{k}-$distinct voter identities $\mathsf{id}_1, ..., \mathsf{id}_\mathsf{k}$ registered with the credentials $\mathsf{cr}_1, .., \mathsf{cr}_\mathsf{k}$ for which we have $\tau \vDash \mathsf{Reg(id_i, cr_i)}$ for $1 \leq \mathsf{i} \leq \mathsf{k}$. By ***(1)***, we can conclude that there is one-to-one correspondence between the set of distinct voter identities $\mathsf{id}_1, ..., \mathsf{id}_\mathsf{k}$, and the trackers $\mathsf{tr}_1, ..., \mathsf{tr}_\mathsf{k}$ for which $\mathsf{v_i} \neq \bot$ on BBres.

(3) According to the one-to-one correspondence between trackers and voter identities established at point (2), we can define the following set:

$$\begin{aligned} \mathsf{Res}_{\mathsf{ID,V}}(\tau) \quad = \quad \{(\mathsf{id, v}) \mid \tau \vDash \mathsf{BBres(tr, v)} \ \wedge\ \mathsf{Link(cr, tr)} \\ \wedge\ \mathsf{Reg(id, cr)} \ \wedge\ \mathsf{v} \neq \bot\} \end{aligned}$$

which associates to each voter identity the corresponding vote in the final result. Note that $\mathsf{Res}(\tau) = \{\!\!\{\mathsf{v} \mid (\mathsf{id, v}) \in \mathsf{Res}_{\mathsf{ID,V}}(\tau)\}\!\!\}$. By $\tau \vDash \Phi^\diamond_{\mathsf{res}}$, for each $(\mathsf{id, v}) \in \mathsf{Res}_{\mathsf{ID,V}}(\tau)$, either $\tau \vDash \mathsf{Vote(id, v)}$ or $\tau \vDash \mathsf{Adv}^\diamond\mathsf{(id)}$, and these events are not mutually exclusive. If we define the following multisets,

$$\begin{aligned} \mathsf{Vote}_{\mathsf{ID,V}}(\tau) &= \{\!\!\{(\mathsf{id, v}) \mid \tau \vDash \mathsf{Vote(id, v)}\}\!\!\} \\ \mathsf{Adv}_{\mathsf{ID,V}}(\tau) &= \{\!\!\{(\mathsf{id, v}) \mid \mathsf{id} \in \mathsf{Adv}^\diamond(\tau)\}\!\!\} \end{aligned}$$

and sets:

$$\begin{aligned} R'_2 &= (\,\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Vote}_{\mathsf{ID,V}}(\tau)\,) \setminus \mathsf{Adv}_{\mathsf{ID,V}}(\tau) \\ R'_3 &= \mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID,V}}(\tau) \end{aligned}$$

then, each $(\mathsf{id, v}) \in \mathsf{Res}_{\mathsf{ID,V}}(\tau)$ is either in R'_2 or in R'_3, where $R'_2 \cap R'_3 = \emptyset$. Intuitively, R'_2 and R'_3 represent the partition of the final result according to honest or adversarial votes. Now, let us define:

$$\mathsf{Ver}_{\mathsf{ID,V}}(\tau) = \{\!\!\{(\mathsf{id, v}) \mid \exists \mathsf{id}', \mathsf{cr}.\ \tau \vDash \mathsf{Ver}^\triangledown(\mathsf{id}', \mathsf{cr, v}) \ \wedge\ \mathsf{Reg(id, cr)}\}\!\!\}$$

and
$$\begin{aligned}R_1 &= \mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID,V}}(\tau),\\ R_2 &= R'_2 \setminus \mathsf{Ver}_{\mathsf{ID,V}}(\tau),\\ R_3 &= R'_3 \setminus \mathsf{Ver}_{\mathsf{ID,V}}(\tau).\end{aligned}$$

By definition, $R_1 \cap R_2 = R_1 \cap R_3 = R_2 \cap R_3 = \emptyset$. Thus, we can conclude $\mathsf{Res}_{\mathsf{ID,V}}(\tau) = R_1 \uplus R_2 \uplus R_3$. It follows that $\mathsf{Res}(\tau) = V_1 \uplus V_2 \uplus V_3$, where $V_i = \{\!\!\{\mathsf{v} \mid \exists \mathsf{id}.\ (\mathsf{id},\mathsf{v}) \in R_i\}\!\!\}$ for $i = 1,2,3$. We show that the multisets V_1, V_2, V_3 satisfy the requirements of Definition 3, respectively. That is, we show:

$$(1)\ V_1 = \mathsf{Ver}^{\triangledown}(\tau) \quad (2)\ V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau),$$
$$(3)\ |V_3| \le |\mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)|.$$

(4.1) By definition, we have $V_1 \subseteq \mathsf{Ver}^{\triangledown}(\tau)$. We have to show $\mathsf{Ver}^{\triangledown}(\tau) \subseteq V_1$. Let $\mathsf{id}_1, \dots, \mathsf{id}_\mathsf{q}$ be the set of (mutually distinct) identities of the voters who verified votes for some credential, and for which the votes are different from $\perp$. This means that there are non-empty sets of votes $A_1, \dots, A_\mathsf{q}$ such that

$$\forall \mathsf{i} \in \{1,\dots,\mathsf{q}\}, \forall \mathsf{v} \in A_\mathsf{i}, \exists \mathsf{cr}.\ \tau \vDash \mathsf{Ver}^{\triangledown}(\mathsf{id}_\mathsf{i}, \mathsf{cr}, \mathsf{v}).$$

Intuitively, for each id_i, A_i represents the set of votes that are verified by that voter and which satisfy the revote policy. By definition, we have $\mathsf{Ver}^{\triangledown}(\tau) = A_1 \uplus \dots \uplus A_\mathsf{q}$.

For each $\mathsf{i} \in \{1,\dots,\mathsf{q}\}$, let C_i be the set of credentials for which $\tau \vDash \mathsf{Ver}^{\triangledown}(\mathsf{id}_\mathsf{i}, \mathsf{cr}, \mathsf{v})$. From $\tau \vDash \Phi^{\triangledown}_{\mathsf{iv2}}$, we deduce that $C_\mathsf{i} \cap C_\mathsf{j} = \emptyset$ for any $\mathsf{i} \neq \mathsf{j}$. Furthermore, assume $C_\mathsf{i} = \{\mathsf{cr}_1, \dots, \mathsf{cr}_\mathsf{s}\}$ for some $\mathsf{i} \in \{1,\dots,\mathsf{q}\}$. From $\tau \vDash \Phi^{\triangledown}_{\mathsf{iv3}} \wedge \Phi_{\mathsf{link2}}$, we deduce

$$\begin{aligned}\tau &\vDash \mathsf{Link}(\mathsf{cr}_1, \mathsf{tr}_1) \wedge \dots \wedge \mathsf{Link}(\mathsf{cr}_\mathsf{s}, \mathsf{tr}_\mathsf{s})\\ \tau &\vDash \mathsf{BBreg}(\mathsf{cr}_1) \wedge \dots \wedge \mathsf{BBreg}(\mathsf{cr}_\mathsf{s})\end{aligned}$$

for mutually distinct $\mathsf{tr}_1, \dots, \mathsf{tr}_\mathsf{s}$. Let T_i be the set of trackers linked to the credentials in C_i, i.e. $T_\mathsf{i} = \{\mathsf{tr}_1, \dots, \mathsf{tr}_\mathsf{s}\}$. Then, from $\tau \vDash \Phi_{\mathsf{link1}}$ and $C_\mathsf{i} \cap C_\mathsf{j} = \emptyset$, we conclude $T_\mathsf{i} \cap T_\mathsf{j} = \emptyset$ for any $\mathsf{i} \neq \mathsf{j}$.

On the other hand, as shown at point ***(1)***, each cr on BBreg is associated to a unique tr on BBres with $\tau \vDash \mathsf{Link}(\mathsf{cr}, \mathsf{tr})$. From the consistency properties Φ_{link1} and Φ_{link2}, the same credential cannot be associated to two different trackers. Then, we can deduce that each set T_i of trackers, that are linked to the credentials in C_i, are included in the final result. Moreover, by $\tau \vDash \Phi_{\mathsf{one}}$, there is at most one vote per tracker in the result. Therefore, we have $\tau \vDash \mathsf{BBres}(\mathsf{tr}_1, \mathsf{v}'_1) \wedge \dots \wedge \mathsf{BBres}(\mathsf{tr}_\mathsf{s}, \mathsf{v}'_\mathsf{s})$, for some votes $\mathsf{v}'_1, \dots, \mathsf{v}'_\mathsf{s}$.

For each $\mathsf{i} \in \{1,\dots,\mathsf{q}\}$, let A'_i be the set of votes in the result that correspond to the trackers in T_i. Since we have established a one-to-one correspondence between C_i (associated to the set A_i) and T_i (associated to the set A'_i), we know that there is a one-to-one correspondence between the sets A_i and A'_i. We will show that these sets are actually equal. Indeed, for any $\mathsf{cr} \in C_\mathsf{i}, \mathsf{v} \in A_\mathsf{i}$ and associated $\mathsf{tr} \in T_\mathsf{i}, \mathsf{v}' \in A'_\mathsf{i}$, we have $\tau \vDash \mathsf{Ver}^{\triangledown}(\mathsf{id}_\mathsf{i}, \mathsf{cr}, \mathsf{v}) \wedge \mathsf{Link}(\mathsf{cr}, \mathsf{tr}) \wedge \mathsf{BBres}(\mathsf{tr}, \mathsf{v}')$. From $\tau \vDash \Phi^{\triangledown}_{\mathsf{iv1}}$, we deduce $\mathsf{v} = \mathsf{v}'$, and we can conclude that $A_\mathsf{i} = A'_\mathsf{i}$. Therefore, we have obtained:

$$\mathsf{Ver}^{\triangledown}(\tau) = A'_1 \uplus \dots \uplus A'_\mathsf{q} = \{\mathsf{v}'_1, \dots, \mathsf{v}'_\mathsf{p}\} \qquad (\star)$$

for some votes $\mathsf{v}'_1, \dots, \mathsf{v}'_\mathsf{p}$ and some $\mathsf{p} \ge \mathsf{q}$ (every voter verified at least one vote).

On the other hand, by definition of the set R_1, it consists of all the pairs $(\mathsf{id}, \mathsf{v}) \in \mathsf{Res}_{\mathsf{ID,V}}(\tau)$ such that $\mathsf{v} \neq \perp$ and some id' verified v. Thus, we deduce

$$\begin{aligned}&\{(\mathsf{id}_1, \mathsf{v}'_1), \dots, (\mathsf{id}_\mathsf{p}, \mathsf{v}'_\mathsf{p})\} \subseteq R_1, \text{ and therefore}\\ &\{\mathsf{v}'_1, \dots, \mathsf{v}'_\mathsf{p}\} \subseteq V_1 \qquad (\star\star)\end{aligned}$$

From $(\star)$ and $(\star\star)$, we deduce $\mathsf{Ver}^{\triangledown}(\tau) \subseteq V_1$ and we can conclude $V_1 = \mathsf{Ver}^{\triangledown}(\tau)$ as required. Furthermore, we have also established

$$|\ \mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID,V}}(\tau)\ | \ge |\ \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)\ | \qquad (*)$$

i.e. the number of verified votes in the final result is greater than the number of voters for whom their votes are verified.

(4.2) We show that $V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau)$. By definition, we have

$$R_2 = (\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Vote}_{\mathsf{ID,V}}(\tau)) \setminus (\mathsf{Ver}_{\mathsf{ID,V}}(\tau) \cup \mathsf{Adv}_{\mathsf{ID,V}}(\tau))$$

and $V_2 = \{\!\!\{\mathsf{v} \mid \exists \mathsf{id}.\ (\mathsf{id},\mathsf{v}) \in R_2\}\!\!\}$. Furthermore, from Ψ_{res2} and the one-to-one correspondence between voter identities and trackers established at point (2), we know that

$$R_2 = \{(\mathsf{id}_1, \mathsf{v}_1), \dots, (\mathsf{id}_\mathsf{s}, \mathsf{v}_\mathsf{s})\} \text{ and } V_2 = \{\mathsf{v}_1, \dots, \mathsf{v}_\mathsf{s}\}$$

for some mutually distinct $\mathsf{id}_1, \dots, \mathsf{id}_\mathsf{s}$. Thus, for all $\mathsf{i} \in \{1,\dots,\mathsf{s}\}$ we have:

$$\tau \vDash \mathsf{Vote}(\mathsf{id}_\mathsf{i}, \mathsf{v}_\mathsf{i}) \ \wedge\ \mathsf{id}_\mathsf{i} \notin (\,\mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau) \cup \mathsf{Adv}^{\diamond}(\tau)\,)$$

Therefore, using the definition of $\mathsf{Vote}^{\triangledown,\diamond}(\tau)$, we can conclude that $V_2 \in \mathsf{Vote}^{\triangledown,\diamond}(\tau)$.

(4.3) We show that $|V_3| \le |\mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)|$. By definition, we have

$$R_3 = (\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID,V}}(\tau)) \setminus \mathsf{Ver}_{\mathsf{ID,V}}(\tau)$$

and $V_3 = \{\!\!\{\mathsf{v} \mid \exists \mathsf{id}.\ (\mathsf{id},\mathsf{v}) \in R_3\}\!\!\}$. Relying on Φ_{one} and the one-to-one correspondence between voter identities and trackers, we have that

$$\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID,V}}(\tau) = \{(\mathsf{id}_1, \mathsf{v}_1), \dots, (\mathsf{id}_\mathsf{t}, \mathsf{v}_\mathsf{t})\}$$

for some mutually distinct $\mathsf{id}_1, \dots, \mathsf{id}_\mathsf{t}$. We can also deduce that

$$|\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID,V}}(\tau)| \le |\mathsf{Adv}^{\diamond}(\tau)| \qquad (\dagger)$$

From $(*)$, since by Φ_{one} there is one and only one vote in the result for each tracker (and therefore for each voter id according to our established correspondence), we deduce

$$|\ \mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID,V}} \cap \mathsf{Adv}_{\mathsf{ID,V}}(\tau)\ | \ge |\ \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau) \cap \mathsf{Adv}^{\diamond}(\tau)\ | \quad (\dagger\dagger)$$

Thus, we can conclude that

$$\begin{aligned}|R_3| &= |\ (\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID,V}}(\tau)) \setminus \mathsf{Ver}_{\mathsf{ID,V}}(\tau)\ |\\ &\le |\ (\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Adv}_{\mathsf{ID,V}}(\tau)) \setminus (\mathsf{Res}_{\mathsf{ID,V}}(\tau) \cap \mathsf{Ver}_{\mathsf{ID,V}}(\tau))\ |\\ &\le |\ \mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)\ |\end{aligned}$$

where, to deduce the second inequality, we rely on $(\dagger), (\dagger\dagger)$ and the general set theory fact that for all sets A_1, A_2, B_1, B_2:

$$\begin{aligned}&|A_1| \le |A_2| \ \wedge\ |A_1 \cap B_1| \ge |A_2 \cap B_2|\\ &\implies |A_1 \setminus B_1| \le |A_2 \setminus B_2|\end{aligned}$$

Hence, $|V_3| = |R_3| \le |\mathsf{Adv}^{\diamond}(\tau) \setminus \mathsf{Ver}^{\triangledown}_{\mathsf{id}}(\tau)|$, as required. $\square$