Provably Improving Election Verifiability in Belenios

Sevdenur Baloglu[1], Sergiu Bursuc[1], Sjouke Mauw[2], and Jun Pang[2]

[1] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
[2] DCS, University of Luxembourg, Esch-sur-Alzette, Luxembourg
{sevdenur.baloglu,sergiu.bursuc,sjouke.mauw,jun.pang}@uni.lu

Abstract. Belenios is an online voting system that provides a strong notion of election verifiability, where no single party has to be trusted, and security holds as soon as either the voting registrar or the voting server is honest. It was formally proved to be secure, making the assumption that no further ballots are cast on the bulletin board after voters verified their ballots. In practice, however, revoting is allowed and voters can verify their ballots anytime. This gap between formal proofs and use in practice leaves open space for attacks, as has been shown recently. In this paper we make two simple additions to Belenios and we formally prove that the new version satisfies the expected verifiability properties. Our proofs are automatically performed with the Tamarin prover, under the assumption that voters are allowed to vote at most four times.

Keywords: Electronic voting · Formal verification · Verifiability.

1 Introduction

Election verifiability aims to ensure that the outcome of an election, relying on a given electronic voting protocol, correctly reflects the votes of eligible voters. One of its important features is that it should be software independent and end-to-end: even if an adversary corrupts (the software on) voting platforms, election authorities, or voting servers, the public information published on the bulletin board should be sufficient to verify that the election outcome correctly reflects voter choices. This verification is performed by honest parties, which are typically a subset of voters and election auditors. Especially for voters, the verification procedure should also be easy to use, in order to achieve widespread adoption and security guarantees.

Helios is an internet voting system that targets this notion of end-to-end verifiability [1,6,7]. However, an important assumption is that the voting server is honest. Otherwise it could stuff ballots, allowing the adversary to add illegitimate votes, most easily for voters that have not voted. In general, for usability, *revoting is allowed* and *voters can verify their ballots anytime* after voting. In that case ballot stuffing is possible even for voters that have verified their ballots successfully. For example, the server can let some time elapse after a ballot was cast, and cast a new ballot in the name of the same voter. This looks like revoting to observers and will not be noticed by voters verifying their ballots right after voting. The so-called clash attacks allow ballot stuffing in a more surreptitious way [24,25,28]: the adversary gives the same credential to two voters, one single vote is cast for them, and the adversary can cast an additional ballot with no change in the total number of ballots. If revoting is disallowed or ballot verification is after the voting phase, this requires voting platforms to be corrupted, since the adversary needs to supply the same ballot for two voters. Otherwise, it was shown in [9] that corrupting the voting platform is not needed: one voter can verify one ballot and another voter can subsequently verify another ballot for the same credential.

Belenios extends Helios in order to get stronger election verifiability [2,16]. There is no single party that has to be trusted: verifiability holds as soon as either the voting server *or* the voting registrar is not corrupted. The registrar generates public credentials, publishes them on the bulletin board, and distributes the respective private credentials to each voter. The public credential is the verification key of a fresh signing key pair, while the private credential is the corresponding signing key. Ballots are signed and election authorities can verify on the bulletin board that all ballots have been cast by the expected legitimate party. A second advantage of Belenios is that it was proved to satisfy a formal notion of election verifiability, both in the symbolic model [15] (for a particular variant) and in the computational model [14]. This adds confidence that verifiability is satisfied by the protocol specification. Nonetheless, several problems of Belenios and of verifiability definitions in [14,15] were shown in [9], leading to weaker guarantees than expected. In the typical scenario when revoting is allowed and voters can verify their ballots anytime, attacks on verifiability are still possible, most damaging in the case when the registrar is corrupted. Even in the ideal case when both the server and the registrar are honest, ballot reordering attacks are possible, breaking individual verifiability. These attacks are outside the scope of proofs in [14,15], since they do not consider the typical scenario of revoting.

Usability, Everlasting Privacy and Verifiability. There are two main features that, put together, allow these attacks on verifiability in Belenios. The first feature is that, in practice [2], revoting is allowed and voters can verify their ballots anytime. This is important for usability and, eventually, also for coercion-resistance [11,23]. The second feature is that the voting server does not know the link between the public credentials and the corresponding voter identities. Only at ballot casting time does the voter reveal this link, and the server ensures its consistency, e.g. that the same public credential does not correspond to two different voters. Revealing minimal information about the association between voters and their public credentials is important in order to ensure everlasting privacy: even if an adversary may break the underlying encryption scheme and penetrate the private logs of the server, the connection between voters and the corresponding votes should remain private. A similar pattern underlies all attacks in [9]: a corrupted voter can be used by the adversary to cast a ballot for a public credential corresponding to an honest voter. Even if honest voters successfully verified their ballots, revoting allows the adversary to undetectably replace them with its own ballots (when the registrar is corrupted), or with earlier ballots submitted by the same voters (when the registrar is honest).

Our Contributions. We propose two simple additions to Belenios and we prove that election verifiability of the resulting system, that we call Belenios+, is strictly stronger in all three scenarios that are subject to attacks in [9]. Each scenario is defined by the corruption abilities of the adversary: $\mathcal{A}_1$ - both the server and the registrar are honest; $\mathcal{A}_2$ - the server is corrupt and the registrar honest; $\mathcal{A}_3$ - the server is honest and the registrar corrupt. In all cases, we assume the adversary may corrupt the secret key of the election, any number of voters and the communication network. For voters, the proposed additions do not require any change in the voting and verification procedures, maintaining the same usability as Belenios. We do not communicate any new information to the voting server regarding the link between voter identities and public credentials. We simply enforce the veracity of the information the voter already communicates. This means that our additions should not affect everlasting privacy in Belenios (everlasting privacy has not been formally proved for Belenios, but it is thought to hold when revoting is not allowed [16]). Our security proofs are in the symbolic model, automatically performed with the Tamarin prover [26], although we need to make some further abstractions, as explained below. We use the verifiability definition of [9], which is more general than [14,15], accounting for revoting and different corruption scenarios.

Belenios relies on a zero-knowledge proof in order to verifiably attach a label to each ballot cast. The label is the public credential of the voter who constructs the ballot and the ballot cannot be detached from the intended label. The goal of this construction is to ensure that each ballot is consistently cast for the intended public credential. Our techniques enrich the structure of the label in order to ensure stronger consistency properties. The first problem that we tackle is a ballot reordering attack, which is possible in all three corruption scenarios, i.e. even for the weakest adversary $\mathcal{A}_1$. Omitting some details (presented in Section 2.2), the attack is as follows: an honest voter with public credential cr, may submit two successive ballots b_1 and b_2; then, relying on a corrupt voter, the adversary can cast b_2 before b_1, for the same public credential cr. The honest voter may then verify b_2 and expect it to be tallied, whereas b_1 is tallied instead. The solution we propose for this problem is to augment the label in the zero-knowledge proof such that each new ballot can also be verifiably linked to the ballot that was cast just before for a given public credential. This proof is publicly verified on the bulletin board, thus it also helps in the scenario $\mathcal{A}_2$.

The second problem in Belenios relates to the scenario $\mathcal{A}_3$ and is at the root of several attacks in [9]: because the voting server does not know in advance the connection between voter identities and public credentials, an adversary corrupting the registrar and a voter may submit any ballot for any public credential cr, and claim it corresponds to that corrupt voter. In particular, this may be a ballot b constructed by an honest voter that received the public credential cr at registration. This leads to the fact that the honest voter may successfully verify b on the bulletin board, while afterwards the adversary is able to cast its own ballot $\mathsf{b}_{\mathcal{A}}$ for the credential cr. The solution we propose for this problem is to further augment the label in the zero-knowledge proof such that the voting server can ensure that the cast ballot is intended for the corresponding voter. However, we need to make sure that only the server can verify the link between a ballot and the voter identity. That is why the label does not directly contain the identity id of the voter, but a commitment to id, for which the server learns the randomness from the voting platform. The randomness can be discarded by the server after reconstructing the commitment and verifying the proof. To hide the identity from an all-powerful adversary against the bulletin board, we can use standard commitment schemes that are perfectly hiding, for example the Pedersen commitment [27].

Abstraction. In practice, the two additions we make do not significantly affect the complexity of running Belenios. However, the fact that we need to recursively link every new ballot with a previously cast ballot significantly affects the running time of Tamarin. To overcome this difficulty, we assume that each voter casts at most four ballots, in effect allowing revoting only thrice (all attacks of [9] occur in scenarios with at most two ballots per voter). We leave as open the problem of formally proving (or disproving) the validity of this assumption. We note that formal results that bound the number of agents or voters for verification have a similar flavour [8,12,13].

Paper Structure. Section 2 contains preliminaries about election verifiability and attacks on Belenios. In Section 3 we describe our improvements and in Section 4 we describe the protocol specification and automated verification with the Tamarin prover.

2 Preliminaries

We describe Belenios in more detail in Section 2.1. The formal notion of election verifiability and the attacks on Belenios are described in Section 2.2.

2.1 Introduction to Belenios

Apart from voters (V), the parties in the Belenios protocol [16,2] are:

- *Administrator* (A): determines the list of eligible candidates and the list of eligible voters.
- *Bulletin Board* (BB): public ledger containing election information: the public key, the list of candidates, the list of public credentials for eligible voters, the list of cast ballots, the final outcome and proofs of correctness. We denote specific portions of BB with suffixes. In particular, BBkey contains the public key of the election, BBcast contains the list of ballots cast for each public credential, and $\mathsf{BBtally}$ contains the list of ballots chosen for tally. BB can only be changed by writing new information on it; previously written information cannot be changed.

- *Trustees* (T): generate the secret key of the election, publish the corresponding public key on BB, compute the final outcome.
- *Registrar* (VR): for each eligible voter, it creates a fresh signing key pair $(\mathsf{vk}, \mathsf{skey})$; vk is the public credential, which is also denoted by cr in the following; it publishes the list of all public credentials on BB.
- *Voting Server* (VS): receives ballots cast by authenticated voters and publishes them on BB; voter authentication is done via passwords.
- *Voting Platform* (VP): constructs ballots for voter choices; authenticates voters with respect to VS and transmits ballots to VS; each ballot contains a ciphertext encrypting the vote, a signature of the ciphertext with respect to skey of the corresponding voter, and zero-knowledge proofs.
- *Election Auditors* (EA): perform audit and verification of proofs on BB. The validity of the ballot is verified by VS at ballot-casting time, and can also be verified by EA at any time afterwards on BBcast.

Setup Phase. A determines the list of eligible voters $\mathsf{id}_1, \ldots, \mathsf{id}_n$, and sends the list to VR and VS. VR generates the public and private credentials for each voter, while VS generates login passwords. Each voter id receives the tuple $\langle \mathsf{cr}, \mathsf{skey}, \mathsf{pwd} \rangle$ during setup phase and BB is updated by the following:

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

Voting Phase. In this phase, voters interact with their voting platform VP to construct a ballot b, which is sent together with their public credential cr to VS. Upon authentication of the voter and validity checks with respect to cr, the ballot is published on BBcast.

$$\begin{aligned}
\mathsf{VP}\colon\ & \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pk}, \mathsf{r}); \ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{skey}); \ \ \mathsf{pr}_\mathsf{R} = \mathsf{proof}_\mathsf{R}(\mathsf{c}, \mathsf{r}, \langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle); \\
& \mathsf{pr}_\mathsf{L} = \mathsf{proof}_\mathsf{L}(\mathsf{c}, \mathsf{r}, \mathsf{cr}); \ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr}_\mathsf{R}, \mathsf{pr}_\mathsf{L} \rangle; \\
\mathsf{VS}\colon\ & \text{authenticates } \mathsf{id} \text{ with } \mathsf{pwd}\text{; receives } \mathsf{b} \text{ and the public credential } \mathsf{cr}; \\
& \text{verifies } \mathsf{s}, \mathsf{pr}_\mathsf{R} \text{ and } \mathsf{pr}_\mathsf{L}\text{; and stores } (\mathsf{id}, \mathsf{cr}) \text{ in } \mathsf{Log}; \\
\mathsf{BBcast}\colon\ & (\mathsf{cr}, \mathsf{b}).
\end{aligned}$$

The signature ensures the voter holds the private part of the public credential cr. The zero-knowledge proof pr_R ensures that the ciphertext contains a vote in a valid range $\langle \mathsf{v}_1, \ldots, \mathsf{v}_k \rangle$. The proof pr_L ensures that the ballot (and the ciphertext) is verifiably linked to the label cr, and cannot be cast for any other credential cr'. In the cryptographic construction, the underlying zero-knowledge proof system takes the arguments of $\mathsf{proof}_\mathsf{R}$ and $\mathsf{proof}_\mathsf{L}$ and returns pr_R and pr_L [14,16]. Moreover, the following consistency property is ensured by VS for the Log storing the association between voter identities and public credentials:

$$\begin{aligned}
&(\mathsf{id}, \mathsf{cr}) \in \mathsf{Log} \ \wedge \ (\mathsf{id}, \mathsf{cr}') \in \mathsf{Log} \ \Rightarrow \ \mathsf{cr} = \mathsf{cr}' \quad \text{and} \\
&(\mathsf{id}, \mathsf{cr}) \in \mathsf{Log} \ \wedge \ (\mathsf{id}', \mathsf{cr}) \in \mathsf{Log} \ \Rightarrow \ \mathsf{id} = \mathsf{id}'.
\end{aligned}$$

This prevents a corrupt voter to use a public credential already used by an honest voter, and also to cast ballots for more than one public credential. In addition to ensuring basic integrity properties, consistency of the log also prevents ballot copy attacks like in [17]. The individual verification procedure enables voters to check their ballots on BB anytime during the election. Specifically, they should check that the expected ballot b is published next to their public credential cr on BBcast.

Tally Phase. The ballots which will be tallied are selected and marked as input for the tally procedure. Selection typically chooses the last ballot cast by each $\mathsf{cr_i}$ and we have $\mathsf{BBtally}\colon (\mathsf{cr_1}, \mathsf{b_1}), \ldots, (\mathsf{cr_n}, \mathsf{b_n})$. $\mathsf{b_i} = \bot$ if no ballot was cast for $\mathsf{cr_i}$. Based on the homomorphic properties of ElGamal encryption [19,22], ciphertexts corresponding to non-empty ballots on $\mathsf{BBtally}$ are combined into a ciphertext c encoding the total number of votes for each candidate. Then, c is decrypted by trustees to obtain the result of the election.

2.2 Election Verifiability and Attacks on Belenios

We consider the symbolic definition of election verifiability from [9], which is an extension of the symbolic definition introduced in [15]. Election verifiability is modelled as a conjunction of properties $\Phi^{\mathsf{h}}_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\circ}_{\mathsf{res}}$, where:

Individual verifiability: $\Phi^{\mathsf{h}}_{\mathsf{iv}}$ ensures that if an honest voter successfully verified the last ballot they cast, then the corresponding vote should be part of the final tally.

Eligibility: Φ_{eli} ensures that if a voter successfully verified a ballot, then the corresponding public credential should be recorded at registration on BB. Moreover, any tallied ballot should correspond to a public credential recorded at registration.

No clash: Φ_{cl} ensures that no two voters can successfully verify their ballot for the same public credential.

Result integrity: $\Phi^{\circ}_{\mathsf{res}}$ ensures that the adversary can cast a ballot for a given public credential only if the corresponding voter is corrupted or has not performed the individual verification procedure for any of the ballots cast. A stronger notion of result integrity, denoted by $\Phi^{\bullet}_{\mathsf{res}}$, prohibits the adversary to cast a ballot even if the voter has not verified any of the ballots cast.

A violation of Φ_{res} is called ballot stuffing; a violation of Φ_{cl} is a clash attack. Belenios is expected to satisfy election verifiability in the following adversarial scenarios: $\mathcal{A}_1$ - both the server and the registrar are honest; $\mathcal{A}_2$ - the server is corrupt and the registrar honest; $\mathcal{A}_3$ - the server is honest and the registrar corrupt. Security should be ensured by private signing keys - when the registrar is honest, and by private passwords and server logs - when the server is honest. However, [9] shows several attacks resulting from the fact that the server does not know the association between a public credential and the identity of the corresponding voter. A corrupt voter can then cast a ballot for any public credential, as soon as the adversary manages to obtain ballots signed with the corresponding private credential.

Ballot Reordering Attack by $\mathcal{A}_1$, $\mathcal{A}_2$ or $\mathcal{A}_3$. Assume an honest voter id with public credential cr casts ballots b_1 and b_2, in this order, and only verifies b_2. Then b_2 should be counted for the respective public credential. However, the adversary can cause b_1 to be counted instead. The attack scenario is as follows:

$\mathsf{V}(\mathsf{id},\mathsf{cr})$: casts b_1 and b_2, which are blocked by $\mathcal{A}$;
$\mathcal{A}$: casts b_2 for cr (relying on a corrupted voter or voting server);
BBcast: $(\mathsf{cr},\mathsf{b}_2)$ is verified by the voter $\mathsf{V}(\mathsf{id},\mathsf{cr})$;
$\mathcal{A}$: casts b_1 for cr;
$\mathsf{BBtally}$: $(\mathsf{cr},\mathsf{b}_1)$.

In a normal execution, the reception of b_1 or b_2 from id would link cr to id, thus the adversary cannot cast b_1 after b_2 when the server is honest - unless it corrupts the password of id. The crucial point of the attack by $\mathcal{A}_1$ is that b_2 is cast for the same public credential cr by a distinct corrupted voter.

Ballot Stuffing and Individual Verifiability Attacks by $\mathcal{A}_3$. When an honest voter id_1 with cr_1 casts a ballot b, the adversary can block and cast it in the name of a corrupt voter id_2, for the same public credential cr_1. The voter id_1 successfully verifies b. Subsequently, relying on a corrupt registrar, the adversary can cast another ballot $\mathsf{b}_{\mathcal{A}}$ for cr_1. This violates result integrity $\Phi^{\circ}_{\mathsf{res}}$ and individual verifiability Φ_{iv}, since an adversarial ballot $\mathsf{b}_{\mathcal{A}}$ is cast for cr_1, even though the corresponding voter is honest and has successfully verified the ballot b.

$\mathcal{A}$: corrupts VR and $\mathsf{V}(\mathsf{id}_2)$ to obtain $\langle \mathsf{cr}_1, \mathsf{skey}_1, \mathsf{pwd}_2\rangle$;
$\mathsf{V}(\mathsf{id}_1)$: casts b, which is blocked by $\mathcal{A}$;
$\mathcal{A}$: casts b with $\langle \mathsf{cr}_1, \mathsf{pwd}_2\rangle$, and VS stores $(\mathsf{id}_2, \mathsf{cr}_1)$ in Log;
BBcast: $(\mathsf{cr}_1, \mathsf{b})$ is verified by $\mathsf{V}(\mathsf{id}_1)$;
$\mathcal{A}$: casts $\mathsf{b}_{\mathcal{A}}$ with $\langle \mathsf{cr}_1, \mathsf{pwd}_2\rangle$, which is accepted and published;
$\mathsf{BBtally}$: $(\mathsf{cr}_1, \mathsf{b}_{\mathcal{A}})$.

If the voter id_2 verified the cast ballot b, this also counts as a clash attack in the definition from [9], as it requires resistance to clash attacks even for corrupted voters. A variation of this attack can also lead to a weaker form of ballot stuffing: the adversary can submit $\mathsf{b}_{\mathcal{A}}$ before id_1 has a chance to cast a ballot. In that case, the voting server will not accept any further ballot from id_1, since this would break the consistency of the log for cr_1. Formally, this is a violation of $\Phi^{\bullet}_{\mathsf{res}}$. Our techniques in the following protect against (strong) ballot stuffing, ballot reordering, individual verifiability attack and the clash attack. They do not protect against the weaker form of ballot stuffing, i.e. the violation of $\Phi^{\bullet}_{\mathsf{res}}$.

3 Towards Improved Election Verifiability

In Belenios, the aim of the zero-knowledge proof $\mathsf{pr_L} = \mathsf{proof_L}(\mathsf{c},\mathsf{r},\mathsf{cr})$ in a ballot $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr_R}, \mathsf{pr_L}\rangle$ is to verifiably link the ciphertext $\mathsf{c} = \mathsf{enc}(\mathsf{v},\mathsf{pk},\mathsf{r})$, and therefore the ballot b, to the public credential cr for which b is cast. We denote the corresponding verification procedure by $\mathsf{ver_L}(\mathsf{pr_L}, \mathsf{c}, \mathsf{cr})$. A valid proof can only be constructed by the party who constructs the ciphertext, by proving knowledge of the corresponding randomness r and associating it with the label cr. This is called labeled encryption in [14]. The idea is that the ciphertext cannot be detached from the label: the adversary cannot copy c, or create a ciphertext related to the encoded vote, and cast it for a different credential cr'. This is required in order to protect from attacks against privacy like in [17]. Concretely, the labeled encryption in Belenios is based on ElGamal encryption with a Chaum-Pedersen proof of knowledge, where the label cr is part of the input to a hash function (SHA-256) that computes the challenge for a non-interactive zero-knowledge proof.

We enrich the structure of the label in order to also protect against the attacks presented in Section 2.2. The elements of the new label structure can be given as inputs to the hash function along with cr in the Chaum-Pedersen proof, thus we can rely on the same labeled encryption construction as Belenios. Moreover, we prove in Section 4 that no further attacks are possible on election verifiability in the resulting system. We present the new structure of the label stepwise: first a label structure that protects against ballot reordering attacks by $\mathcal{A}_1, \mathcal{A}_2$ or $\mathcal{A}_3$; then a label structure that protects against other attacks by $\mathcal{A}_3$ (in particular ballot stuffing); finally, combining the two labels protects against all attacks by $\mathcal{A}_1, \mathcal{A}_2$ or $\mathcal{A}_3$.

3.1 Protection Against Ballot Reordering

We assume initially there are empty ballots next to eligible public credentials on BB. Moreover, a specific portion of BB is reserved for displaying the last ballot cast for each credential:

$$\text{(Before voting)} \quad \mathsf{BBlast} : (\mathsf{cr_1}, \perp), \ldots, (\mathsf{cr_n}, \perp)$$

$$\text{(During voting)} \quad \mathsf{BBlast} : (\mathsf{cr_1}, \mathsf{b_1}), \ldots, (\mathsf{cr_n}, \mathsf{b_n})$$

When the voting platform VP constructs a new ballot for a voter with public credential cr, it fetches from BBlast the last ballot b' associated to cr. Then, in the construction of the proof $\mathsf{pr_L}$, instead of cr, VP uses the label $\mathsf{h}(\mathsf{cr}, \mathsf{b}')$, where h is a collision-resistant hash function mapping the pair $(\mathsf{cr}, \mathsf{b}')$ into the appropriate domain for labels:

$$\ell = \mathsf{h}(\mathsf{cr}, \mathsf{b}'); \quad \mathsf{pr_L} = \mathsf{proof_L}(\mathsf{c}, \mathsf{r}, \ell); \quad \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr_R}, \mathsf{pr_L}, \ell \rangle.$$

BBcast records all ballots cast for cr, and their order cannot be changed on BB. Election auditors can look at any two consecutive ballots b' and b cast for a credential cr and verify that

$$\mathsf{ver_L}(\mathsf{pr_L}, \mathsf{c}, \mathsf{h}(\mathsf{cr}, \mathsf{b}')) = \mathsf{ok},$$

thereby ensuring that the party constructing b indeed expects it to follow b'. In particular, if an honest voter casts $\mathsf{b_2}$ after $\mathsf{b_1}$, the adversary cannot cast $\mathsf{b_2}$

first, since it would have to generate a proof linking b_2 to an earlier ballot b_0, which is impossible since the adversary does not know the randomness in the ciphertext corresponding to b_2. This label structure ensures election verifiability in corruption scenarios when the registrar is honest, i.e. $\mathcal{A}_1$ and $\mathcal{A}_2$.

3.2 Protection Against a Corrupted Registrar

The main cause of the attacks, in the scenario with a corrupted registrar, is that the adversary can block a ballot b of an honest voter and cast it under the identity of a corrupt voter, while maintaining the same public credential associated to b. Subsequently, after the honest voter verified b, the adversary can override it with an own ballot $\mathsf{b}_{\mathcal{A}}$. In order to prevent this, we enrich the label attached to b so that it includes a commitment to the identity of the voter. More precisely, during ballot casting for a voter id, VP generates a fresh randomness t, constructs the label $\langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle$ and sends t together with the ballot to the voting server VS. Since the label cannot be reconstructed publicly by election auditors, we explicitly include it in the ballot. We have:

$$\begin{array}{ll} \mathsf{VP} & : \ell = \langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle;\ \ \mathsf{pr_L} = \mathsf{proof_L}(\mathsf{c}, \mathsf{r}, \ell);\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr_R}, \mathsf{pr_L}, \ell\rangle, \\ \hdashline \mathsf{VS} & : \text{receives } (\mathsf{cr}, \mathsf{b}, \mathsf{t}) \text{ from } \mathsf{VP} \text{ for a given } \mathsf{id};\ \ell' = \langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle, \\ & \ \ \text{casts } \mathsf{b} \text{ if and only if } \ell' = \ell \text{ and } \mathsf{ver_L}(\mathsf{pr_L}, \mathsf{c}, \ell) = \mathsf{ok}. \end{array}$$

In the attack scenario described above, the adversary cannot construct a proof pr'_L so that b is cast by VS under the identity of a corrupt voter. Indeed, the ciphertext in b cannot be detached from the identity of the honest voter. More generally, we prove that this structure of the label is sufficient to ensure election verifiability in the corruption scenarios when the server is honest, i.e. $\mathcal{A}_1$ and $\mathcal{A}_3$. Election auditors can still check the proof $\mathsf{pr_L}$ on BB, but they are only be able to ensure the ballot is cast for the expected public credential cr and will not have knowledge of the underlying id. Note that we cannot use the id directly in the label, as this would reveal the link between id and cr. Moreover, the commitment scheme should be perfectly hiding, in order to resist an all-powerful, e.g. quantum, adversary.

3.3 Putting the Labels Together

We combine the labels from Section 3.1 and Section 3.2 as follows:

$$\ell_1 = \mathsf{h}(\mathsf{cr}, \mathsf{b}'); \quad \ell_2 = \mathsf{com}(\mathsf{id}, \mathsf{t}); \quad \ell = \langle \ell_1, \ell_2\rangle.$$

We call Belenios$_{\text{tr}}$ (from tracking) the variant of Belenios where we augment the label as described in Section 3.1, Belenios$_{\text{id}}$ the variant where the label is as in Section 3.2 and Belenios+ the variant where the label ℓ is as described in this section. For a protocol P, a corruption scenario $\mathcal{A}$ and a property Φ, we denote by $(P, \mathcal{A}) \models \Phi$ the fact that P satisfies Φ in the corruption scenario $\mathcal{A}$. Let $\Phi^{\circ}_{\mathsf{E2E}}$ be the election verifiability property $\Phi^{\mathsf{h}}_{\mathsf{iv}} \wedge \Phi_{\mathsf{eli}} \wedge \Phi_{\mathsf{cl}} \wedge \Phi^{\circ}_{\mathsf{res}}$ as described in Section 2.2

and in [9]. In the next section, we describe the specification and automated verification with Tamarin. They allow us to derive the following results:

$$\begin{array}{rrl} & (\text{ Belenios}_{\text{tr}},\ \mathcal{A}\ ) \models \Phi^{\circ}_{\mathsf{E2E}} & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2\}, \\ & (\text{ Belenios}_{\text{id}},\ \mathcal{A}\ ) \models \Phi^{\circ}_{\mathsf{E2E}} & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_3\}, \\ & (\text{ Belenios+},\ \mathcal{A}\ ) \models \Phi^{\circ}_{\mathsf{E2E}} & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}, \\ \text{while we have} & (\text{ Belenios},\ \mathcal{A}\ ) \not\models \Phi^{\circ}_{\mathsf{E2E}} & \text{for } \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}. \end{array}$$

The property $\Phi^{\circ}_{\mathsf{E2E}}$ corresponds to the standard verifiability notion used in [14,15]. In particular, this notion ensures that, if an honest voter successfully verified a ballot b for a public credential cr, then b is counted in the final tally as the contribution of cr. A stronger notion of verifiability, denoted by $\Phi^{\bullet}_{\mathsf{E2E}}$, was also proposed in [9]: if a ballot is counted for a public credential corresponding to an honest voter, then it must necessarily have been cast by that voter - independently of the individual verification procedure. In the scenario $\mathcal{A}_3$, an adversary corrupting the registrar and a voter can cast a ballot $\mathsf{b}_{\mathcal{A}}$ for any public credential, violating the strong verifiability notion $\Phi^{\bullet}_{\mathsf{E2E}}$, even in Belenios+. The label $\langle \mathsf{h}(\mathsf{cr}, \mathsf{b}'), \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle$ does not help here, since the adversary can freely combine the identity of a corrupted voter with any credential, sign the ballot and construct valid zero-knowledge proofs. If the honest voter already submitted and successfully verified a ballot b, then the adversary cannot make VS accept $\mathsf{b}_{\mathcal{A}}$ for the same public credential under the identity of a corrupt voter. This is due to the fact that the association between the honest voter and the public credential is recorded by the server in the log upon accepting b. That is why $\Phi^{\circ}_{\mathsf{E2E}}$ holds for Belenios+.

4 Specification and Verification

4.1 Specifying Protocols in Tamarin

We perform our analysis of Belenios+ using the Tamarin prover, which is based on a multiset rewriting framework. We only illustrate the most relevant features of Tamarin here. For a detailed understanding of Tamarin we refer the reader to [3,26,29]. In Tamarin, messages (or terms) are built from a set of function symbols and properties of cryptographic primitives are modelled by a set of equations. Protocol state information and adversarial knowledge are represented by facts, modelled relying on special fact symbols. Protocol actions are specified by multiset rewriting rules, denoted by $[L]{-}\!\![\ M\ ]\!\!\rightarrow[N]$, in which a set of *premise facts* L allows to derive a set of *conclusion facts* N, while recording certain events in *action facts* M.

Example 1. In a voting protocol, the generation of a secret/public key pair can be modelled by the following multiset rewriting rule, that we denote by $\mathsf{R_{key}}$:

$$[\ \mathsf{Fr(k)}\ ]{-}\!\![\ \mathsf{!BBkey(pk(k))}, \mathsf{Phase('setup')}\ ]\!\!\rightarrow[\ \mathsf{!Sk(k)}, \mathsf{!BBkey(pk(k))}, \mathsf{Out(pk(k))}\ ]$$

where $\mathsf{Fr(k)}$ denotes the randomly generated fresh key k as a premise. The conclusion facts $\mathsf{!Sk(k)}$ and $\mathsf{!BBkey(pk(k))}$ record the secret and the public key of the election, respectively; the term $\mathsf{pk}(\mathsf{k})$ represents the public key itself, while $!\mathsf{BBkey}(\mathsf{pk}(\mathsf{k}))$ represents the fact that $\mathsf{pk}(\mathsf{k})$ is a public key published on BBkey. If a fact is preceded by !, it means that it can be consumed (i.e. used as premise) any number of times by other protocol rules. Otherwise it can be consumed only once, and it is called a *linear* fact. The fact symbols In and Out are used for communication over the network, controlled by the attacker. The action fact $\mathsf{BBkey}(\mathsf{pk}(\mathsf{k}))$ records the event that the public key is published on the bulletin board. The action fact $\mathsf{Phase}('\mathsf{setup}')$ records that the rule should be executed in the setup phase. The following rules set up candidates v_1 and v_2 and voter identities id:

$$\begin{aligned} \mathsf{R}_{\mathsf{cand}} &: [\ \mathsf{In}(\langle \mathsf{v}_1, \mathsf{v}_2\rangle)\ ]\!-\!\![\ \mathsf{Phase}('\mathsf{setup}')\ ]\!\!\rightarrow [\ !\mathsf{BBcand}(\mathsf{v}_1), !\mathsf{BBcand}(\mathsf{v}_2)\ ] \\ \mathsf{R}_{\mathsf{id}} &: \qquad [\ \mathsf{In}(\mathsf{id})\ ]\!-\!\![\ \mathsf{Phase}('\mathsf{setup}')\ ]\!\!\rightarrow [\ !\mathsf{Id}(\mathsf{id})\ ] \end{aligned}$$

To cast a ballot, the voter with identity id makes a choice between the candidates recorded on BBcand and encrypts the vote v using the public key from BBkey together with fresh randomness r. The output including the voter identity id can be sent to the server over the network. To model this action, we define the following rule, where the event $\mathsf{Vote}(\mathsf{id}, \mathsf{v})$ is recorded as an action fact:

$$\begin{aligned} \mathsf{R}_{\mathsf{vote}} : &[\ !\mathsf{Id}(\mathsf{id}), !\mathsf{BBcand}(\mathsf{v}), !\mathsf{BBkey}(\mathsf{pk}(\mathsf{k})), \mathsf{Fr}(\mathsf{r})\ ] \\ &-\!\![\ \mathsf{Vote}(\mathsf{id}, \mathsf{v}), \mathsf{Phase}('\mathsf{voting}')\ ]\!\!\rightarrow [\ \mathsf{Out}(\langle \mathsf{id}, \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r})\rangle)\ ] \end{aligned}$$

Cryptographic operations are specified by equations. For example, decryption using the private key k is specified by:

$$\mathsf{dec}(\mathsf{enc}(\mathsf{v}, \mathsf{pk}(\mathsf{k}), \mathsf{r}), \mathsf{k}) = \mathsf{v}$$

where the term $\mathsf{enc}(\mathsf{v}, \mathsf{pk}(\mathsf{k}), \mathsf{r})$ represents the encryption of v with public key $\mathsf{pk}(\mathsf{k})$ and randomness r. It can be decrypted only if the secret key k is provided.

A *restriction* in Tamarin is a logical formula that constrains the application of protocol rules. For example, the restriction $\forall x, y, i, j.\ \mathsf{BBkey}(x)\ @i\ \wedge\ \mathsf{BBkey}(y)\ @j \Rightarrow x = y$ applied to the rule $\mathsf{R}_{\mathsf{key}}$ in Example 1 means that it is not possible to have two different election keys. The symbol @ refers to the timepoints i and j in the execution trace when the rule $\mathsf{R}_{\mathsf{key}}$ is applied. We can also express a timepoint ordering or equality. For example, the restriction $\forall i, j.\ \mathsf{Phase}('\mathsf{setup}')\ @i\ \wedge\ \mathsf{Phase}('\mathsf{voting}')\ @j \Rightarrow i \prec j$ means that all setup actions should occur before voting actions. A restriction can also encode the equality predicate, enforcing that u and v are equal in any occurrence of the action fact $\mathsf{Eq}(u, v) : \forall u, v, i.\ \mathsf{Eq}(u, v)\ @i \Rightarrow u = v$.

We note that formal verification with Tamarin does not guarantee full-proof security, as Tamarin itself may have bugs. Recently, there is research aiming to underpin fully automated provers like Tamarin with foundations from interactive theorem provers like Coq [4,10,21].

4.2 Specification and Verification of Belenios+

We define a set of equations used for specifying decryption (1), signature verification (2), verification of a range proof (3), and verification of a proof attaching a label to a ciphertext (4):

(1) $\mathsf{dec}(\mathsf{enc}(x, \mathsf{pk}(y), z), y) = x$,

(2) $\mathsf{ver}(\mathsf{sign}(x, y), x, \mathsf{pk}(y)) = \mathsf{ok}$,

(3) $(\forall i)\ \mathsf{ver_R}(\mathsf{proof_R}(\mathsf{enc}(x_i, y, z), z, \langle x_1, \ldots, x_k\rangle), \mathsf{enc}(x_i, y, z), y, \langle x_1, \ldots, x_k\rangle) = \mathsf{ok}$,

(4) $\mathsf{ver_L}(\mathsf{proof_L}(\mathsf{enc}(x, y, z), z, \ell), \mathsf{enc}(x, y, z), \ell) = \mathsf{ok}$.

To specify the set of equations (3) in Tamarin, the number of candidates k has to be fixed in advance. We use $\mathsf{k} = 2$, but any constant would work. For modelling the actions of participants in the protocol, we define a set of rules and restrictions. For the complete specification, we refer to the Tamarin code online [5]. It is an extension of the code corresponding to Belenios in [9]. In the following, we discuss two of the most important rules in the specification: ballot casting as it happens on the voting platform VP and on the voting server VS. We highlight the difference between Belenios+ and Belenios in red. We use special linear facts in order to track the last ballot cast for each credential: $\mathsf{VPlast}(\mathsf{cr}, \mathsf{b_0})$ - to be used by the voting platform, and $\mathsf{BBlast}(\mathsf{cr}, \mathsf{b_0})$ - to be used by the voting server. The rule for ballot casting on the voting server makes sure these two facts are in sync. For voter credentials, we use special facts $!\mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{skey})$ and $!\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd})$ to store credentials received from the registrar and from the server, respectively. Ballot casting by VP is represented by the following rule:

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

$$\begin{aligned}&\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{skey});\ \ \ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b_0}), \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle;\\ &\qquad \mathsf{pr_R} = \mathsf{proof_R}(\mathsf{c}, \mathsf{r}, \mathsf{vlist});\ \ \mathsf{pr_L} = \mathsf{proof_L}(\mathsf{c}, \mathsf{r}, \ell);\\ &\qquad \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr_R}, \mathsf{pr_L}, \ell\rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{t}\rangle)\ \ \mathsf{in}\\ &[\ !\mathsf{BBcand}(\mathsf{v}), !\mathsf{BBkey}(\mathsf{pkey}), \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t}), !\mathsf{Vlist}(\mathsf{vlist}), !\mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{skey}),\\ &\ \ !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b_0})\ ]\!-\!\![\ \mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow\\ &[\ !\mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{t}\rangle)\ ]\end{aligned}$$

where we use the Tamarin construction let. . . in for assigning terms to variables. The rule abstracts password-based authentication with the help of a hash function, essentially ensuring that only a party knowing the password can cast a ballot for a given id. In reality, the randomness t used for the commitment should be sent on the same secure channel as the password. However, the secrecy of t is not important for verifiability properties, thus we can send it on the public channel. The rule $\mathsf{R^{VP}_{vote}}$ consumes the linear fact $\mathsf{VPlast}(\mathsf{cr}, \mathsf{b_0})$, thus it can be executed only once for any ballot posted on BB. This mechanism is complemented by the ballot casting rule on the server side:

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

$$\begin{aligned}&\mathsf{let}\ \ \ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b_0}), \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle;\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr_R}, \mathsf{pr_L}, \ell\rangle;\\ &\qquad \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{t}\rangle)\ \ \mathsf{in}\\ &[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{t}\rangle), !\mathsf{BBkey}(\mathsf{pkey}), !\mathsf{Vlist}(\mathsf{vlist}), !\mathsf{BBreg}(\mathsf{cr}), !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}),\\ &\ \ \mathsf{BBlast}(\mathsf{cr}, \mathsf{b_0})\ ]\!-\!\![\ \mathsf{a}' = \mathsf{a}, \mathsf{ver}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) = \mathsf{ok}, \mathsf{ver_R}(\mathsf{pr_R}, \mathsf{c}, \mathsf{pkey}, \mathsf{vlist}) = \mathsf{ok},\\ &\ \ \mathsf{ver_L}(\mathsf{pr_L}, \mathsf{c}, \ell) = \mathsf{ok}, \mathsf{Log}(\mathsf{id}, \mathsf{cr}), !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow\\ &[\ !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}), \mathsf{BBlast}(\mathsf{cr}, \mathsf{b}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b})\ ]\end{aligned}$$

where we receive a ballot from the voter and perform the corresponding validation steps: verifying the password, the signature and the zero-knowledge proofs. The fact containing the last ballot cast is consumed, and new facts are produced for the new ballot: one to be consumed by the voting platform, and one to be consumed by the server when the next ballot is cast. In order to obtain termination, we have a restriction limiting the number of applications of this rule to at most four for each voter. The following rule and restriction model the individual verification procedure, where the restriction ensures that the voter verifies the last ballot cast:

$$\mathsf{R}^{\mathsf{V}}_{\mathsf{ver}} : [\ !\mathsf{Voted}(\mathsf{id},\mathsf{cr},\mathsf{v},\mathsf{b}), !\mathsf{BBcast}(\mathsf{cr},\mathsf{b})\ ]\!\!-\!\![\ \mathsf{Verified}(\mathsf{id},\mathsf{cr},\mathsf{v}), \mathsf{VerB}(\mathsf{id},\mathsf{cr},\mathsf{b})\ ]\!\!\rightarrow\![\ ]$$

$$\Psi^{\mathsf{V}}_{\mathsf{last}} : !\mathsf{BBcast}(\mathsf{cr},\mathsf{b})\ @i \wedge !\mathsf{BBcast}(\mathsf{cr},\mathsf{b}')\ @j\ \wedge\ \mathsf{VerB}(\mathsf{id},\mathsf{cr},\mathsf{b})\ @l\ \wedge\ i \prec l\ \wedge\ j \prec l \Rightarrow j \prec i\ \vee\ \mathsf{b} = \mathsf{b}'$$

Corruption Scenarios. We have three adversary models $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, as described in Section 2.2. Trustees are corrupted by default: we have a rule that takes the secret key as input from the attacker. For other corruption abilities, we have the following rules:

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

$\mathcal{C}^{\mathsf{VS}}_{\mathsf{pwd}}$: **corrupt server to determine password**
$[\ !\mathsf{Id}(\mathsf{id}), \mathsf{In}(\mathsf{pwd})\ ]\!-\!\![\ ]\!\!\rightarrow\![\ !\mathsf{Pwd}(\mathsf{id},\mathsf{pwd})\ ]$

$\mathcal{C}^{\mathsf{VS}}_{\mathsf{cast}}$: **corrupt server to stuff ballots**
$[\ \mathsf{In}(\langle \mathsf{cr},\mathsf{b}\rangle), \mathsf{BBlast}(\mathsf{cr},\mathsf{b}_0)\ ]\!-\!\![\ !\mathsf{BBcast}(\mathsf{cr},\mathsf{b})\ ]\!\!\rightarrow$
$[\ !\mathsf{BBcast}(\mathsf{cr},\mathsf{b}), \mathsf{BBlast}(\mathsf{cr},\mathsf{b}), \mathsf{VPlast}(\mathsf{cr},\mathsf{b})\ ]$

$\mathcal{C}^{\mathsf{VR}}_{\mathsf{reg}}$: **corrupt registration of public / secret credentials**
let $\mathsf{cr} = \mathsf{pk}(\mathsf{skey})$ in
$[\ !\mathsf{Id}(\mathsf{id}), \mathsf{In}(\langle \mathsf{skey},\mathsf{cr}'\rangle)\ ]\!-\!\![\ !\mathsf{BBreg}(\mathsf{cr}')\ ]\!\!\rightarrow\![\ !\mathsf{Reg}(\mathsf{id},\mathsf{cr},\mathsf{skey}), !\mathsf{BBreg}(\mathsf{cr}')\ ]$

Moreover, when the server is corrupted, in the rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{vote}}$ we only keep the verification actions that can be publicly performed by election auditors. Table 1 contains verification results for the corresponding specifications with Tamarin, obtained with the specifications posted online [5]. We can see that the positive results for Belenios+ are the union of the positive results for Belenios$_{\mathrm{tr}}$ and Belenios$_{\mathrm{id}}$, in each of the corruption cases $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$. In Table 2, we give execution times for the verification of Belenios+ when we bound the number of ballots per voter accordingly. Tamarin does not terminate without such a bound (it takes more than one hour for five ballots per voter).

5 Conclusion and Future Work

We have introduced a simple extension of Belenios and we have proved with the Tamarin prover that the resulting system improves election verifiability in various corruption scenarios. These additions do not affect usability and efficiency of

Table 1. Verifiability analysis of the variants of Belenios.

$\boldsymbol{\Phi}/\mathcal{A}_j$	Belenios*			Belenios$_{\text{tr}}$		Belenios$_{\text{id}}$		Belenios+		
	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_1$	$\mathcal{A}_3$	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
$\Phi^{\mathsf{h}}_{\mathsf{iv}}$	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
Φ_{eli}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Φ_{cl}	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
$\Phi^{\bullet}_{\mathsf{res}}$	✓	✓	✗	✓	✓	✓	✗	✓	✓	✗
$\Phi^{\circ}_{\mathsf{res}}$	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓

$*$: Verification results for Belenios as in [9].

Table 2. Execution times for the verification of verifiability of Belenios+.

$\#\mathsf{b}/\mathcal{A}_j$	Belenios+		
	$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
2 ballots per voter	17 sec	8 sec	57 sec
3 ballots per voter	1 min	33 sec	2 min 47 sec
4 ballots per voter	12 min 6 sec	15 min	15 min 53 sec

Belenios. We also claim that (everlasting) privacy is not affected, but this has to be formally proved. The bulletin board has the same structure, but the order in which all ballots are cast for a given credential should be clear. Our open problems are related to the formal verification and to the design of electronic voting protocols. Our specification makes certain abstractions that should be lifted or formally justified, for greater confidence in results. The most important abstraction is the one limiting the number of ballots to four for each voter. Concerning the design, our techniques still do not achieve the stronger notion of election verifiability, that prevents the adversary from casting ballots even for honest voters that have not verified their ballots. We also think election verifiability could be achieved in stronger corruption scenarios, e.g. when both the registrar and the server are (partially) corrupted. For example, it could be interesting to achieve public verifiability for the fact that each ballot is associated to an eligible voter, while perfectly hiding the actual identity of the voter. This would limit the corruption abilities of the registrar who generates the public credentials, without relying on the server to perform the verification.

Acknowledgement

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

References

1. Helios - Verifiable online elections, `https://heliosvoting.org/`
2. Belenios - Verifiable online voting system, `https://belenios.org/`
3. Tamarin prover, `https://tamarin-prover.github.io`
4. The Coq proof assistant, `https://coq.inria.fr/`
5. Tamarin specifications for the variants of Belenios, `https://github.com/sbaloglu/tamarin-codes/tree/main/belenios-zkp`
6. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA. pp. 335–348. USENIX Association (2008), `http://www.usenix.org/events/sec08/tech/full_papers/adida/adida.pdf`
7. Adida, B., De Marneffe, O., Pereira, O., Quisquater, J.J.: Electing a university president using open-audit voting: Analysis of real-world use of helios. In: 2009 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections. Usenix (2009)
8. Arapinis, M., Cortier, V., Kremer, S.: When are three voters enough for privacy properties? In: Computer Security - ESORICS - 21st European Symposium on Research in Computer Security. Lecture Notes in Computer Science, vol. 9879, pp. 241–260. Springer (2016). https://doi.org/10.1007/978-3-319-45741-3_13
9. Baloglu, S., Bursuc, S., Mauw, S., Pang, J.: Election verifiability revisited: Automated security proofs and attacks on Helios and Belenios. In: 34th IEEE Computer Security Foundations Symposium (2021), `https://eprint.iacr.org/2020/982`
10. Castéran, P., Bertot, Y.: Interactive theorem proving and program development. Coq'Art: The Calculus of inductive constructions. Texts in Theoretical Computer Science, Springer Verlag (2004), `https://hal.archives-ouvertes.fr/hal-00344237`
11. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: 2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA. pp. 354–368 (2008). https://doi.org/10.1109/SP.2008.32
12. Comon-Lundh, H., Cortier, V.: Security properties: Two agents are sufficient. In: Degano, P. (ed.) Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003. Lecture Notes in Computer Science, vol. 2618, pp. 99–113. Springer (2003). https://doi.org/10.1007/3-540-36575-3_8
13. Cortier, V., Dallon, A., Delaune, S.: Bounding the number of agents, for equivalence too. In: Piessens, F., Viganò, L. (eds.) Principles of Security and Trust (POST). Lecture Notes in Computer Science, vol. 9635, pp. 211–232. Springer (2016). https://doi.org/10.1007/978-3-662-49635-0_11
14. Cortier, V., Drăgan, C.C., Dupressoir, F., Warinschi, B.: Machine-checked proofs for electronic voting: Privacy and verifiability for Belenios. In: Proceedings of the 31st IEEE Computer Security Foundations Symposium. pp. 298–312. IEEE Computer Society (2018). https://doi.org/10.1109/CSF.2018.00029
15. Cortier, V., Filipiak, A., Lallemand, J.: BeleniosVS: Secrecy and verifiability against a corrupted voting device. In: 32nd IEEE Computer Security Foundations Symposium. pp. 367–381 (2019). https://doi.org/10.1109/CSF.2019.00032
16. Cortier, V., Gaudry, P., Glondu, S.: Belenios: A simple private and verifiable electronic voting system. In: Guttman, J.D., Landwehr, C.E., Meseguer, J., Pavlovic, D. (eds.) Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows. Lecture Notes in Computer Science, vol. 11565, pp. 214–238. Springer (2019). https://doi.org/10.1007/978-3-030-19052-1_14

17. Cortier, V., Smyth, B.: Attacking and fixing Helios: An analysis of ballot secrecy. Journal of Computer Security **21**(1), 89–148 (2013). https://doi.org/10.3233/JCS-2012-0458
18. Dang, Q.: Secure hash standard (shs) (2012-03-06 2012). https://doi.org/https://doi.org/10.6028/NIST.FIPS.180-4
19. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara. pp. 10–18 (1984). https://doi.org/10.1007/3-540-39568-7_2
20. Glondu, S.: Belenios specification (2018 [Online]), `https://www.belenios.org/specification.pdf`
21. Hess, A.V., Mödersheim, S., Brucker, A.D., Schlichtkrull, A.: Performing security proofs of stateful protocols. In: 34th IEEE Computer Security Foundations Symposium (CSF). vol. 1, pp. 143–158. IEEE (2021). https://doi.org/10.1109/CSF51468.2021.00006, `https://www.brucker.ch/bibliography/abstract/hess.ea-performing-2021`
22. Hirt, M., Sako, K.: Efficient receipt-free voting based on homomorphic encryption. In: Advances in Cryptology - EUROCRYPT 2000. pp. 539–556. Springer (2000)
23. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society, WPES. pp. 61–70 (2005). https://doi.org/10.1145/1102199.1102213
24. Küsters, R., Truderung, T., Vogt, A.: Verifiability, privacy, and coercion-resistance: New insights from a case study. In: 32nd IEEE Symposium on Security and Privacy. pp. 538–553. IEEE Computer Society (2011). https://doi.org/10.1109/SP.2011.21
25. Küsters, R., Truderung, T., Vogt, A.: Clash attacks on the verifiability of e-voting systems. In: 33rd IEEE Symposium on Security and Privacy. pp. 395–409. IEEE Computer Society (2012). https://doi.org/10.1109/SP.2012.32
26. Meier, S., Schmidt, B., Cremers, C., Basin, D.A.: The TAMARIN prover for the symbolic analysis of security protocols. In: 25th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, vol. 8044. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_48
27. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology. p. 129–140. CRYPTO '91, Springer-Verlag (1991)
28. Pereira, O., Wallach, D.S.: Clash attacks and the star-vote system. In: Krimmer, R., Volkamer, M., Binder, N.B., Kersting, N., Pereira, O., Schürmann, C. (eds.) Electronic Voting - Second International Joint Conference, E-Vote-ID. Lecture Notes in Computer Science, vol. 10615, pp. 228–247. Springer (2017). https://doi.org/10.1007/978-3-319-68687-5_14
29. Schmidt, B., Meier, S., Cremers, C.J.F., Basin, D.A.: Automated analysis of diffie-hellman protocols and advanced security properties. In: 25th IEEE Computer Security Foundations Symposium, (CSF'12). pp. 78–94. IEEE Computer Society (2012). https://doi.org/10.1109/CSF.2012.25

Appendix A Details for the ZK Proofs in Belenios+

The zero-knowledge proofs are used in Belenios: (1) for trustees (1a) to prove their knowledge of secret keys in setup phase, and (1b) to prove the correct decryption in tally phase; (2) for voters (2a) to prove their vote v being within valid range $\langle \mathsf{v}_1, ..., \mathsf{v}_k \rangle$, and (2b) to prove the knowledge of randomness r and the credential cr for the association of the ballot to the owner in voting phase. In this paper, we focus on the zero-knowledge proof of (2b) which corresponds to $\mathsf{pr_L}$ in the ballot structure $\mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr_R}, \mathsf{pr_L} \rangle$ throughout the paper.

The cryptography underlying Belenios [16,20] makes use of a cyclic group $\mathbb{G} = \langle \mathsf{g} \rangle$ with order q, which is a multiplicative subgroup of $\mathbb{F}_\mathsf{p}^*$. Proofs are generated using the Chaum-Pedersen algorithm and made non-interactive by the Fiat-Shamir technique. The algorithm generates a challenge ch and a response re to prove the knowledge of a secret sec corresponding to a public value $\mathsf{g}^{\mathsf{sec}}$, and sends $(\mathsf{ch}, \mathsf{re})$ as a proof to the verifier. The verifier computes ch using the messages $\mathsf{g}^{\mathsf{sec}}$ and re, and accepts if the computed value matches the one previously received. To generate the proof $(\mathsf{ch}, \mathsf{re})$, $\mathsf{w} \in \mathbb{Z}_\mathsf{q}$ is randomly chosen. Then,

$$\mathsf{ch} = \mathsf{h}(\mathsf{g}^{\mathsf{sec}},\ \mathsf{g}^{\mathsf{w}}) \mod \mathsf{q} \quad \text{and} \quad \mathsf{re} = \mathsf{w} - \mathsf{sec} \times \mathsf{ch} \mod \mathsf{q}$$

are computed. To verify the proof, given $(\mathsf{ch}, \mathsf{re})$ and $\mathsf{g}^{\mathsf{sec}}$, $\mathsf{A} = \mathsf{g}^{\mathsf{re}}(\mathsf{g}^{\mathsf{sec}})^{\mathsf{ch}}$ is computed and checked that ch is equal to $\mathsf{h}(\mathsf{g}^{\mathsf{sec}}, \mathsf{A})$.

In the implementation of zero-knowledge proofs in Belenios [20], h is specifically the SHA-256 hash function [18], which can take an input up to 2^{64} bits and generates a fixed size output of 256 bits, and the proof $\mathsf{pr_L} = \mathsf{proof_L}(\mathsf{c}, \mathsf{r}, \mathsf{cr})$ is computed as follows:

$$\mathsf{ch} = \mathsf{SHA256}(\mathsf{cr} \mid \langle \mathsf{pk}, \mathsf{c} \rangle \mid \mathsf{g}^{\mathsf{w}}) \mod \mathsf{q} \quad \text{and} \quad \mathsf{re} = \mathsf{w} - \mathsf{r} \times \mathsf{ch} \mod \mathsf{q},$$

where $\mathsf{cr} \in \mathbb{G}$ is the verification key of the corresponding voter, $\mathsf{w} \in \mathbb{Z}_\mathsf{q}$, pk is the election public key, $\mathsf{c} = (\mathsf{g}^{\mathsf{r}}, \mathsf{pk}^{\mathsf{r}}\mathsf{g}^{\mathsf{v}})$ is the ciphertext of the vote v. Here, the randomness r is the secret to be proved as a knowledge for a valid encryption. Thus, $\mathsf{A} = \mathsf{g}^{\mathsf{re}}(\mathsf{g}^{\mathsf{r}})^{\mathsf{ch}}$ is computed for the verification of $\mathsf{pr_L}$.

In this paper, we propose to enrich the structure of $\mathsf{pr_L}$ with the following replacements of cr:

- $\mathsf{h}(\mathsf{cr}, \mathsf{b}')$ for Belenios$_{\mathrm{tr}}$,
- $\langle \mathsf{cr}, \mathsf{com}(\mathsf{id}, \mathsf{t}) \rangle$ for Belenios$_{\mathrm{id}}$, and
- $\langle \mathsf{h}(\mathsf{cr}, \mathsf{b}'), \mathsf{com}(\mathsf{id}, \mathsf{t}) \rangle$ for Belenios+.

This means that cr in the generation of challenge ch will be replaced accordingly. For Belenios$_{\mathrm{tr}}$, we propose to use a collision-resistant hash function h that takes as input cr and the former cast ballot b' on the BB. The hash function h can be SHA-256 for the compatibility within the system, i.e. $\mathsf{h}(\mathsf{cr}, \mathsf{b}') \equiv \mathsf{SHA256}(\mathsf{cr} \mid \mathsf{b}')$. Assume that the former ballot on BB is not empty, i.e. $\mathsf{b}' \neq \perp$. Then, b' will be in the following form:

$$\begin{aligned} \mathsf{b}' &= (\mathsf{c}, \mathsf{s}, \mathsf{pr_R}, \mathsf{pr_L}, \ell') \\ &= ((\mathsf{g}^{\mathsf{r}}, \mathsf{pk}^{\mathsf{r}}\mathsf{g}^{\mathsf{v}}), (\mathsf{ch}_1, \mathsf{re}_1), (\mathsf{ch}_2, \mathsf{re}_2), (\mathsf{ch}_3, \mathsf{re}_3), \mathsf{h}(\mathsf{cr}, \mathsf{b}'')). \end{aligned}$$

Here, s corresponds to a Schnorr-like digital signature which is also a pair of challenge and response in $\mathbb{Z}_\mathsf{q}$. Therefore, we have $\mathsf{c} \in \mathbb{G} \times \mathbb{G}$ and $(\mathsf{ch_i}, \mathsf{re_i}) \in \mathbb{Z}_\mathsf{q} \times \mathbb{Z}_\mathsf{q}$. Note that every element in $\mathbb{G}$ has the same size as p since $\mathbb{G}$ is a subgroup of $\mathbb{F}_\mathsf{p}^*$.

In the specification [20], the lengths of p and q are taken as 2048 bits and 256 bits, respectively. Together with $\mathsf{cr} \in \mathbb{G}$ and $\ell' \in \mathbb{Z}_{\mathsf{q}}$, the input size for h makes $31 \times 256 \approx 2^{13}$ bits, which is definitely suitable for SHA-256.

For Belenios$_{\text{id}}$, cr in the challenge ch is replaced with $\langle \mathsf{cr}, \mathsf{com}(\mathsf{id},\mathsf{t})\rangle$. This new structure requires a commitment to the voter's identity id with a randomness t. Regarding the cryptography used for Belenios, the commitment can be a Pedersen commitment. In this case, another generator $\bar{\mathsf{g}}$ of $\mathbb{G}$ will be used for the commitment $\mathsf{com}(\mathsf{id},\mathsf{t}) = \mathsf{g}^{\mathsf{id}}\bar{\mathsf{g}}^{\mathsf{t}} \in \mathbb{G}$ for $\mathsf{t} \in \mathbb{Z}_{\mathsf{q}}$. Thus, the input size of SHA-256 in ch will be increased by 2048 bits since we add a commitment in addition to cr. In a similar fashion, when we enrich pr_{L} by applying $\langle \mathsf{h}(\mathsf{cr},\mathsf{b}'), \mathsf{com}(\mathsf{id},\mathsf{t})\rangle$ as a first argument in the challenge for Belenios+, the input size of SHA-256 will be increased by the output size of $\mathsf{h}(\mathsf{cr},\mathsf{b}')$, which is 256 bits. Recall that $\mathsf{com}(\mathsf{id},\mathsf{t})$ has the same length as cr and $\mathsf{h}(\mathsf{cr},\mathsf{b}')$ is $\mathsf{SHA256}(\mathsf{cr} \mid \mathsf{b}')$ as shown above. Hence, our propositions to enrich the structure of the proof pr_{L} fit well with the cryptographic primitives used in Belenios.

Appendix B Belenios+ Specification Details

The details for the Belenios+ specification is in Figure 1 on the following page.

Appendix C Limiting the Number of Ballots in Tamarin

To obtain verification results in Tamarin regarding Belenios+ specification, we need to restrict the number of ballots which can be cast by each voter, i.e. we need to limit the number of revotes. Our current specification in Tamarin allows up to four cast ballots, i.e. three revotes. The code does not terminate for a higher bound. We specify the number of ballots allowed to be cast by restrictions in Tamarin. If the allowance is for two ballots, then we use a restriction Ψ_{two} as follows:

$$\begin{aligned}\Psi_{\mathsf{two}} : {} & \mathsf{TwoTimes(x)}\ @i \ \wedge\ \mathsf{TwoTimes(x)}\ @j \ \wedge\ \mathsf{TwoTimes(x)}\ @k \\ & \Rightarrow i = j \ \vee\ i = k \ \vee\ j = k\end{aligned}$$

This restriction refers to the action fact $\mathsf{TwoTimes}(\langle \mathsf{cr}, '\mathsf{cast}'\rangle)$ used in the rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$, which leads to a limitation on the number of ballots on BBcast. Similarly, to specify an allowance for three and four ballots, we use the following restrictions:

$$\begin{aligned}\Psi_{\mathsf{three}} : {} & \mathsf{ThreeTimes(x)}\ @i \ \wedge\ \mathsf{ThreeTimes(x)}\ @j \ \wedge\ \mathsf{ThreeTimes(x)}\ @k \ \wedge \\ & \mathsf{ThreeTimes(x)}\ @l \Rightarrow i = j \ \vee\ i = k \ \vee\ i = l \ \vee\ j = k \ \vee\ j = l \ \vee\ k = l\end{aligned}$$

$$\begin{aligned}\Psi_{\mathsf{four}} : {} & \mathsf{FourTimes(x)}\ @i \ \wedge\ \mathsf{FourTimes(x)}\ @j \ \wedge\ \mathsf{FourTimes(x)}\ @k \ \wedge \\ & \mathsf{FourTimes(x)}\ @l \ \wedge\ \mathsf{FourTimes(x)}\ @m \Rightarrow i = j \ \vee\ i = k \ \vee\ i = l \ \vee \\ & i = m \ \vee\ j = k \ \vee\ j = l \ \vee\ j = m \ \vee\ k = l \ \vee\ k = m \ \vee\ l = m\end{aligned}$$

For these restrictions, we call the respective action facts $\mathsf{ThreeTimes}(\langle \mathsf{cr}, '\mathsf{cast}'\rangle)$ and $\mathsf{FourTimes}(\langle \mathsf{cr}, '\mathsf{cast}'\rangle)$ in the same server casting rule $\mathsf{R}^{\mathsf{VS}}_{\mathsf{cast}}$.

SETUP PHASE

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

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

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

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

$[\ \mathsf{In}(\mathsf{vlist})\ ]\!-\!\![\ !\mathsf{Vlist}(\mathsf{vlist})\ ]\!\!\rightarrow\![\ !\mathsf{BBcand}(\mathsf{v}_1), \ldots, !\mathsf{BBcand}(\mathsf{v}_\mathsf{k}), !\mathsf{Vlist}(\mathsf{vlist})\ ]$

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

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

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

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

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

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

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

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

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

VOTING PHASE

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

$\mathsf{let}\ \ \mathsf{c} = \mathsf{enc}(\mathsf{v}, \mathsf{pkey}, \mathsf{r});\ \ \mathsf{s} = \mathsf{sign}(\mathsf{c}, \mathsf{skey});\ \ \ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle;$

$\quad \mathsf{pr}_\mathsf{R} = \mathsf{proof}_\mathsf{R}(\mathsf{c}, \mathsf{r}, \mathsf{vlist});\ \ \mathsf{pr}_\mathsf{L} = \mathsf{proof}_\mathsf{L}(\mathsf{c}, \mathsf{r}, \ell);$

$\quad \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr}_\mathsf{R}, \mathsf{pr}_\mathsf{L}, \ell\rangle;\ \ \mathsf{a} = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{t}\rangle)\ \ \mathsf{in}$

$[\ !\mathsf{BBcand}(\mathsf{v}), !\mathsf{BBkey}(\mathsf{pkey}), \mathsf{Fr}(\mathsf{r}), \mathsf{Fr}(\mathsf{t}), !\mathsf{Vlist}(\mathsf{vlist}), !\mathsf{Reg}(\mathsf{id}, \mathsf{cr}, \mathsf{skey}), !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b}_0)\ ]$

$-\!\![\ \mathsf{Vote}(\mathsf{id}, \mathsf{cr}, \mathsf{v}), \mathsf{VoteB}(\mathsf{id}, \mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow\![\ !\mathsf{Voted}(\mathsf{id}, \mathsf{cr}, \mathsf{v}, \mathsf{b}), \mathsf{Out}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{t}\rangle)\ ]$

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

$\mathsf{let}\ \ \ell = \langle \mathsf{h}(\mathsf{cr}, \mathsf{b}_0), \mathsf{com}(\mathsf{id}, \mathsf{t})\rangle;\ \ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr}_\mathsf{R}, \mathsf{pr}_\mathsf{L}, \ell\rangle;\ \ \mathsf{a}' = \mathsf{h}(\langle \mathsf{id}, \mathsf{pwd}, \mathsf{cr}, \mathsf{b}, \mathsf{t}\rangle)\ \ \mathsf{in}$

$[\ \mathsf{In}(\langle \mathsf{id}, \mathsf{cr}, \mathsf{b}, \mathsf{a}, \mathsf{t}\rangle), !\mathsf{BBkey}(\mathsf{pkey}), !\mathsf{Vlist}(\mathsf{vlist}), !\mathsf{BBreg}(\mathsf{cr}), !\mathsf{Pwd}(\mathsf{id}, \mathsf{pwd}), \mathsf{BBlast}(\mathsf{cr}, \mathsf{b}_0)\ ]$

$-\!\![\ \mathsf{a}' = \mathsf{a}, \mathsf{ver}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) = \mathsf{ok}, \mathsf{ver}_\mathsf{R}(\mathsf{pr}_\mathsf{R}, \mathsf{c}, \mathsf{pkey}, \mathsf{vlist}) = \mathsf{ok}, \mathsf{ver}_\mathsf{L}(\mathsf{pr}_\mathsf{L}, \mathsf{c}, \ell) = \mathsf{ok}, \mathsf{Log}(\mathsf{id}, \mathsf{cr}),$

$\quad !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b})\ ]\!\!\rightarrow\ [\ !\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}), \mathsf{BBlast}(\mathsf{cr}, \mathsf{b}), \mathsf{VPlast}(\mathsf{cr}, \mathsf{b})\ ]$

TALLY PHASE

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

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

INDIVIDUAL VERIFICATION

$\mathsf{R}^{\mathsf{V}}_{\mathsf{ver}}$: **voter verifies the ballot anytime on** BBcast

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

- -

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

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

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

$!\mathsf{BBcast}(\mathsf{cr}, \mathsf{b}) \Rightarrow !\mathsf{BBreg}(\mathsf{cr})\ \wedge\ (\ \mathsf{b} = \langle \mathsf{c}, \mathsf{s}, \mathsf{pr}_\mathsf{R}, \mathsf{pr}_\mathsf{L}, \ell\rangle \Rightarrow !\mathsf{BBkey}(\mathsf{pkey}) \wedge !\mathsf{Vlist}(\mathsf{vlist})$

$\wedge\ \mathsf{ver}(\mathsf{s}, \mathsf{c}, \mathsf{cr}) = \mathsf{ok}\ \wedge\ \mathsf{ver}_\mathsf{R}(\mathsf{pr}_\mathsf{R}, \mathsf{c}, \mathsf{pkey}, \mathsf{vlist}) = \mathsf{ok}\ \wedge\ \mathsf{ver}_\mathsf{L}(\mathsf{pr}_\mathsf{L}, \mathsf{c}, \ell) = \mathsf{ok}\ )$

$\Psi^{\mathsf{VS/EA}}_{\mathsf{tally}}$: **the last ballot added to BB is selected for tally; can be audited by** EA

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

$\Psi^{\mathsf{V}}_{\mathsf{last}}$: **the verified ballot is currently the last on BB**

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

Fig. 1. The specification of Belenios+.