Machine-checked proofs for electronic voting: privacy and verifiability for Belenios

Véronique Cortier
LORIA, CNRS & Inria & Université de Lorraine
veronique.cortier@loria.fr

Constantin Cătălin Drăgan
LORIA, CNRS & Inria (Now at University of Surrey)
c.dragan@surrey.ac.uk

François Dupressoir
University of Surrey
f.dupressoir@surrey.ac.uk

Bogdan Warinschi
University of Bristol
csxbw@bristol.ac.uk

***Abstract*—We present a machine-checked security analysis of Belenios – a deployed voting protocol used already in more than 200 elections. Belenios extends Helios with an explicit registration authority to obtain eligibility guarantees.**

We offer two main results. First, we build upon a recent framework for proving ballot privacy in EasyCrypt. Inspired by our application to Belenios, we adapt and extend the privacy security notions to account for protocols that include a registration phase. Our analysis identifies a trust assumption which is missing in the existing (pen and paper) analysis of Belenios: ballot privacy does not hold if the registrar misbehaves, even if the role of the registrar is seemingly to provide eligibility guarantees. Second, we develop a novel framework for proving strong verifiability in EasyCrypt and apply it to Belenios. In the process, we clarify several aspects of the pen-and-paper proof, such as how to deal with revote policies.

Together, our results yield the first machine-checked analysis of both ballot privacy and verifiability properties for a deployed electronic voting protocol. Perhaps more importantly, we identify several issues regarding the applicability of existing definitions of privacy and verifiability to systems other than Helios. While we show how to adapt the definitions to the particular case of Belenios, our findings indicate the need for more general security notions for electronic voting protocols with registration authorities.

1. Introduction

The need for secure electronic voting is indisputable. While some countries (like Germany, the UK or Norway) have decided to stop or even ban electronic voting due to the fear of security failures, induced by several attacks [32], [33], others are at least trialing, or even running legally binding elections using such electronic voting systems (used in Australia, Estonia, Switzerland or Uruguay, for example). Regardless of its use in major political elections, electronic voting is often employed in other types of elections, whose lower stakes does not make them less important, or less prone to subversion attempts, such as the election of union representatives or administration councils. To be useful in such settings, voting systems need to satisfy two key security properties: i. ballot privacy (no one knows how I voted), and ii. verifiability (I can check that the result includes my vote, and anyone can check that the result corresponds to the published evidence), both of which may sometimes come in stronger flavours (such as receipt-freeness, coercion-resistance, everlasting privacy, or accountability).

As is the case for general security protocols, the tasks of formalizing security notions for electronic voting, designing voting protocols that meet these notions, and formally proving that this is the case are all complex, error-prone tasks. In particular, formal definitions for privacy and verifiability are continously evolving [13], [21] and attacks against well-studied protocols still arise (for example, clash attacks in ThreeBallots [29] or replay attack on Helios [23]). The recent development of formal techniques and tools to reason about the correctness of cryptographic proofs, and their partial application to electronic voting, makes these protocols and their proofs interesting targets for formalization.

Our main contribution is a full, machine-checked security analysis of the Belenios voting protocol [1], covering both privacy and verifiability considerations. The choice of Belenios–a protocol that is deployed on a voting platform [1], and has already been used in more than 200 elections which include the election of representatives in research labs and universities–serves as evidence that machine-checked proofs can contribute to building high-assurance electronic voting systems that are applicable in practice, with increased understanding and adoption as a potential side-effect of reinforcing trust. Our analysis uses EasyCrypt[1] [6], an interactive theorem prover with support for writing cryptographic security proofs in the computational model of cryptography. It has been successfully applied to various protocols, including multi-party computation [3] and authenticated key-exchange protocols [5]. We build upon an existing formalisation of the Helios protocol and its privacy properties in EasyCrypt [18], and demonstrate some form of proof reuse, despite differences in the scheme and privacy definitions. We now detail our results.

PRIVACY. Our analysis of the privacy properties of Belenios builds on an existing proof methodology suggested by Bernhard et al [13]. This approach identifies three distinct security notions that have some bearing on the privacy of

1. https://www.easycrypt.info

individual votes. These properties, namely BPRIV (ballot privacy of the voting phase), strong correctness (a honestly generated ballot will always be accepted), and strong consistency (the tally behaves as counting the votes extracted from ballots) have been shown to entail simulation-based privacy [13].

These notions form the basis of a formal framework developed in EasyCrypt by Cortier et al [18] to analyze hundreds of variants of the Helios voting system. Although Belenios is an extension of Helios, the existing framework is not sufficient since it does not account for the type of registration authority which Belenios employs. Interestingly, straightforward extensions of the existing privacy notions result in definitions that are not satisfied by Belenios.

The first step in this formal analysis was therefore to extend the existing privacy definitions in a way that reflects the intuition behind the original notions and covers the setting specific to Belenios. In this, we follow the definitions proposed by Cortier et al. [20] in their pen-and-paper analysis of Belenios, but unveil a trust assumption that was missing. Interestingly, privacy holds only when the registrar is honest: a dishonest registrar may selectively prevent some honest ballots from being counted which results in a privacy breach, since only the votes from the targeted voters would remain. This trust assumption can be seen as the analogue of a trust assumption on the ballot box. Bernhard and Smyth [16] note that existing game-based definitions of privacy implicitly assume an honest ballot box: just like a corrupt registrar, a corrupt ballot box may selectively remove some honest ballots and break privacy. This assumption on the registrar is missing from the analysis of Cortier et al [20] only because they focus on ballot privacy–for which it is not necessary–whereas we also consider the related properties of strong correctness and strong consistency. As such, their analysis is not flawed, but simply incomplete.

On the matter of privacy, in addition to a machine-checked proof for Belenios, we therefore also contribute a formalization of extended definitions that include a registration phase and identify permissive trust assumptions under which Belenios can be proved secure.

VERIFIABILITY. As far as we are aware, our work is the first to produce a machine-checked proof of verifiability properties of a voting system in the computational model.

Specifically, from the various proposed definition of verifiability (see [21] for a survey), we select one due to Cortier et al [20]. The notion enjoys two nice features: it simultaneously captures individual, universal, and eligibility verifiability and additionally accounts for the fact that voters may not perform the verifiability checks. Intuitively, a voting system is strongly verifiable [20] if the result of the election reflects:

- all the votes of the honest voters who checked;
- *some* of the votes of the honest voters who did not check (an attacker may typically always drop such votes); and
- if k dishonest voters were involved in the election, then at most k additional votes.

We develop Easycrypt formalizations of strong verifiability and apply the resulting framework to Belenios. Our analysis yields several clarifications of the original pen-and-paper definitions of [20], for example including consideration of (simple) revoting policies and clearer definitions of circumstances in which a voter is deemed to have checked her vote. Moreover, we strengthen the definitions by letting the adversary have the election's secret key which models that the trusted parties may misbehave. We further extend our formal development to cover not only the deployed version of Belenios but also several of its election variants (changing, for example, the kind and amount of data is published by the tallying authorities).

Taken together, our results yield the first machine checked analysis of both privacy and verifiability properties for a deployed electronic voting protocol. Our formal definitions, statements and proof scripts are available publicly [19], along with instructions on how to verify them.

RELATED WORK. We already discussed in details the work of [18]. To our knowledge, it is the only machine-checked of a voting scheme in cryptographic models, under standard assumptions. It does not cover verifiability and cannot be applied directly to Belenios. Automatic proofs of privacy in the symbolic model have been proposed for several protocols of the literature such as FOO [25], Helios [23], or the Norwegian e-voting protocol [24]. In addition to analysing these protocols in models of cryptography that are considerably more abstract than the usual computational models, the use of automated protocol verification tools like ProVerif [17] requires further simplifications of both the cryptographic primitives and the protocol (e.g. a limited number of ballots can be received, in a given order).

A pen and paper proof of a preliminary version of Belenios appears in [20]. However, this work is not a direct implementation of this proof. First, our formal model of verifiability led to further clarifications of the original definitions. Second, and more importantly, the privacy proof was only sketched in [20] and covered only the BPRIV property. Our privacy analysis also covers the two complementary properties (strong correctness and strong consistency) and reveals a missing trust assumption (although not contracting the theorems of [20]): the registrar needs to be honest in order to aim for simulation-based security.

2. Preliminaries

In this section we recall the basic cryptographic primitives used by Belenios. We keep details minimal and provide appropriate references for further details.

Labelled Public-Key Encryption Scheme. Labelled encryption is an extension of the classical definition introduced by Shoup [31]. Encryption takes, in addition to keys and plaintexts, a public label. The resulting ciphertext protects the integrity of the label: decryption of a ciphertext will only succeed if the appropriate label is provided. The primitive does not aim to provide any privacy for the label which is transmited either out of band or together with the ciphertext.

Formally, the labelled public-key encryption schemes is defined as a triple of algorithms $\mathcal{E} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ where:

- KGen is a randomized algorithm which on input a security parameter λ, produces a pair of public and secret keys;
- Enc is a randomized algorithm which on input a public key, a label and a plaintext outputs a ciphertext; We write $C \leftarrow \mathsf{Enc}(\mathsf{pk}, \ell, m))$ for the process of producing ciphertext C of message m under public key pk and label ℓ.
- Dec is a deterministic algorithm which outputs either a plaintext or a special error symbol $\perp$, given as input the secret key, a label and a ciphertext.

Correctness requries that with overwhelming probability the relation

$$\mathsf{Dec}(\mathsf{sk}, \ell, \mathsf{Enc}(\mathsf{pk}, \ell, m)) = m,$$

holds, for any $(\mathsf{pk}, \mathsf{sk})$ output by KGen, any label ℓ and any message m.

We rely on two security properties for labeled encryption. First we ask for non-malleability under chosen-plaintext attack [26] and we use the equivalent formalization suggested by Bellare and Sahai [11] termed *indistinguishability under chosen-ciphertext attack with one parallel decryption query* (IND-1-CCA) in [12]. Second, we require *well-spreadness* for ciphertexts [28]. Intuitively, the latter models situation where it is highly unlikely (negligible) to find a collision on ciphertexts. More formally, $\mathcal{E}$ is well-spread if for any efficient adversary $\mathcal{B}$ the probability of the next experiment:

$$\Pr[c, m, \ell \leftarrow \mathcal{B}(\mathsf{sk});\ c' \leftarrow \mathsf{Enc}(\mathsf{pk}, m, \ell);\ \mathbf{return}\ (c = c')],$$

is negligible in λ, for any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$.

Most encryption schemes used in electronic voting protocols are homomorphic. This means, there exists a deterministic algorithm Add such that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Add}(c_1, c_2)) = \mathsf{Dec}(\mathsf{sk}, c_1) + \mathsf{Dec}(sk, c_2),$$

for any secret key sk, and any two ciphertexts c_1, c_2.

Hash functions: collision resistance and random oracles. In our analysis we model hash functions in two different ways. Whenever possible, we only resort to collision resistance of the hash function; in these cases we assume the hash function is keyed. Formally, any hash function H satisfies *collision resistance*, if for any efficient adversary $\mathcal{B}$ the following probability is negligible

$$\Pr[m_0, m_1 \leftarrow \mathcal{B}(K)\ :\ m_0 \neq m_1 \wedge H_K(m_0) = H_K(m_1)]$$

where the probability is over the choice of coins used by the key-generation algorithm of H, used to generate K.

For most uses however, we model them as random oracles [9] and rely on the same formalization as used in the ballot privacy proof for Helios [18]. That is, we replace the computation of a hash function for a value x with an oracle call $\mathsf{O}(x)$. This oracle maintains a table T that stores pairs of input and output values. When an algorithm calls $\mathsf{O}(x)$ for some x, O checks if there exists (x, y) in T and returns y if true; otherwise it generates a value y' stores (x, y') in T and returns y'.

To simplify notation, in many cases, we do not explicitly show the dependency/access to the random oracle.

Proof System. Belenios uses (zero-knowledge) proof systems in several places, to ensure that the various operations have been performed correctly. We briefly go over the formalization which we use in our anlaysis. Consider a binary (NP-relation) $\mathcal{R}$ over a pair of elements (x, w), where the element x is called a *statement* and w is called *witness*. A non-interactive proof system for $\mathcal{R}$ consists of a prover P and a verifier V algorithm which work on a common input x; the prover computes a proof $\pi \leftarrow \mathsf{P}(x, w)$ with the help of the witness and sends it to the verifier; the verifier then decides whether to accept or reject the proof.

A *proof system* is said to be *complete*, if for any $(x, w) \in \mathcal{R}$, we have that $\mathsf{V}(x, \mathsf{P}(x, w)) = \mathsf{true}$ with overwhelming probability.

The second property is soundness: a prover cannot convince a verifier that a false statement is true. In this paper we use the stronger property that asks that the proof system is a proof of knowledge: if a prover can convince a verifier that a statement is true, then is possible to extract a witness for that statement [8].

In this paper we consider an extension where the prover returns *two* valid statements (and corresponding proofs) and the extractor needs to provide witnesses for both. The formal experiment is in Figure 1. It considers an extractor K which works against an adversary $\mathcal{B}$: the adversary produces two statements and accompanying proofs. The adversary wins if the proofs are valid, yet the extractor cannot extract witnesses for both statements.

$\mathsf{Exp}^{pok}_{\mathcal{B},\mathsf{V},\mathcal{R}}()$

1 : $(x_1, \pi_1, x_2, \pi_2) \leftarrow \mathcal{B}$
2 : $w_1 \leftarrow K(x_1, \pi_1)$
3 : $w_2 \leftarrow K(x_2, \pi_2)$
4 : $e \leftarrow \mathsf{V}(x_1, \pi_1) \wedge \mathsf{V}(x_2, \pi_2)$
5 : $rel \leftarrow \mathcal{R}(x_1, w_1) \wedge \mathcal{R}(x_2, w_2)$
6 : $\mathbf{return}\ e \wedge \neg rel$

Figure 1. Proof of knowledge experiment.

The standard notion considers an adversary who outputs a single statement/proof pair. However, the non-adaptive case where the adversary outputs two such pairs is equivalent [14], [15]. Therefore, we abuse the notion and say that a proof system is a *a proof of knowledge system*, if there exists an efficient extractor K that produces witnesses for any valid-looking proofs and statement, such that for any efficient adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ the probability to win the experiment in Figure 1 is negligible.

$$\Pr\left[\mathsf{Exp}^{pok}_{\mathcal{B},\mathsf{V},\mathcal{R}}() = 1\right].$$

We emphasize that in the random oracle model the extractor K has access to the random oracle calls made by $\mathcal{B}$; for

simplicity we do not explicitly show this ability in Figure 1).

Another important security property of proof systems, mainly used in ensuring privacy of electronic voting, is *zero-knowledge*. It ensures that no information is leaked, beside the validity of the relation. Formally, we compare the execution of a zero-knowledge adversary $\mathcal{B}$ with that of a simulator S in the following two experiments.

$\mathsf{Exp}^{\mathsf{zk},0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda)$	$\mathsf{Exp}^{\mathsf{zk},1}_{\mathcal{B},\mathsf{S},\mathcal{R}}(\lambda)$
1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(n)$	1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(n)$
2 : $\pi \leftarrow \perp$	2 : $\pi \leftarrow \perp$
3 : **if** $(\mathcal{R}(x, w))$ **then**	3 : **if** $(\mathcal{R}(x, w))$ **then**
4 : $\quad \pi \leftarrow \mathsf{P}(x, w)$	4 : $\quad \pi \leftarrow \mathsf{S}(x)$
5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \pi)$	5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \pi)$
6 : **return** β'	6 : **return** β'

Informally, the goal of the simulator is to output valid proofs (for valid statements) but without having access to a corresponding witness: if such a simulator exists then the proof itself contains no information about the witness. We again emphasize that we are in the random oracle model: here the simulator is in complete control of the random oracle. As explained before, we do not show here the dependency on the oracle.

The advantage of a zero-knowledge adversary $\mathcal{B}$ over the proof system $\Sigma_{\mathcal{R}} = (\mathsf{P}, \mathsf{V})$, and simulator S is defined as:

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{zk},0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{zk},1}_{\mathcal{B},\mathsf{S},\mathcal{R}}(\lambda) = 1\right]\right|.$$

A proof system is zero-knowledge if for any adversary $\mathcal{B}$ there exists a simulator S which makes the advantage defined above, negligible.

Signature schemes. Formally, a *signature scheme* is a triple of algorithms $\mathcal{E} = (\mathsf{SKGen}, \mathsf{Sig}, \mathsf{Vf})$ where:

SKGen is a randomized algorithm which on input a security parameter λ, produces a pair of verification and signing credential keys;

Sig is a randomized algorithm which on input a signing key and a message outputs a string;

Vf is a deterministic algorithm which outputs a boolean value, given as input the verification key, the message and string.

For any $(\mathsf{upk}, \mathsf{usk})$ output by SKGen, any message m, we have with overwhelming probability that:

$$\mathsf{Vf}(\mathsf{upk}, m, \mathsf{Sig}(\mathsf{usk}, m)) = \mathsf{true}.$$

The security property for signature schemes used by the voting protocols considered in this paper rely on *strong existential unforgeability* [4]. Intuitively, it states that for an adversary it should be difficult to create a new valid signature for some message m even after seeing signatures on arbitrary messages of his chosing, including on the message for which it attempts the forgery. Formally, a signing system $\Pi = (\mathsf{SKGen}, \mathsf{Sig}, \mathsf{Vf})$ is *strongly existentially unforgeable*, if for any efficient adversary $\mathcal{B}$ the probability to win the experiment in Figure 2 is negligible.

$$\Pr\left[\mathsf{Exp}^{seuf}_{\mathcal{B},\Pi}() = 1\right].$$

$\mathsf{Exp}^{seuf}_{\mathcal{B},\Pi}()$	$Osign(m)$
1 : $sL \leftarrow [\,]$	1 : $s \leftarrow \mathsf{Sig}(\mathsf{usk}, m)$
2 : $(\mathsf{upk}, \mathsf{usk}) \leftarrow \mathsf{SKGen}()$	2 : $sL \leftarrow sL \cup (m, s)$
3 : $(m, s) \leftarrow \mathcal{B}^{Osign}(\mathsf{upk})$	3 : **return** s
4 : $e \leftarrow \mathsf{Vf}(\mathsf{upk}, m, s)$	
5 : **return** $e \wedge (m, s) \notin sL$	

Figure 2. Strong existential unforgeability experiment.

3. Voting Definitions and Properties

In this section we recall *single-pass voting schemes*, the class of schemes that we use as basis for our analysis of Belenios. We start with their syntax and then detail the desired privacy and verifiability properties for such schemes. A *single-pass voting system* [13] is a tuple of algorithms

$$\big(\mathsf{Setup},\ \mathsf{Register},\ \mathsf{Vote},\ \mathsf{Valid},\ \mathsf{VerifyVote},\ \mathsf{Publish},\ \mathsf{Tally},\ \mathsf{Verify},\ \mathsf{Box},\ \mathsf{Count},\ \mathsf{Policy}\big).$$

$\mathsf{Setup}(1^\lambda)$**:** Returns a pair of keys $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Register}(id)$**:** Creates a pair of signing keys $(\mathsf{upk}, \mathsf{usk})$ for the voter id.

$\mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$**:** Constructs a ballot b for voter id that contains the encryption of their cast vote v with label upk, and a signature over this encryption with the signing key usk. The verification key upk can trivially be computed from the signing key usk.

$\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$**:** Checks the validity of ballot b with respect to the ballot box BB.

$\mathsf{Publish}(\mathsf{BB})$**:** Returns the public view of the ballot box BB called the public bulletin board.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$**:** Computes the result r of the election and a proof π of correct computation from BB.

$\mathsf{Box}(\mathsf{BB}, b)$**:** Returns a ballot box BB'. If $\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$ holds then $\mathsf{BB}' = \mathsf{BB} \cup \{b\}$; otherwise $\mathsf{BB}' = \mathsf{BB}$.

$\mathsf{Verify}((\mathsf{pk}, pbb, r), \pi)$**:** Checks that π is a valid proof of correct computation for result r and public bulletin board pbb.

$\mathsf{VerifyVote}(b, pbb)$**:** Evaluates if the ballot b was properly registered with respect to the public bulletin board pbb.

$\mathsf{Count}(L)$**:** Given a list of votes as input, a method of computing the result of the election is applied.

$\mathsf{Policy}(L)$**:** Applies a filtering policy that decides which vote is kept for each voter, given a list of voters and votes as input.

3.1. Privacy

Vote privacy, is the idea that no one can obtain information about how individual voters have voted. For our

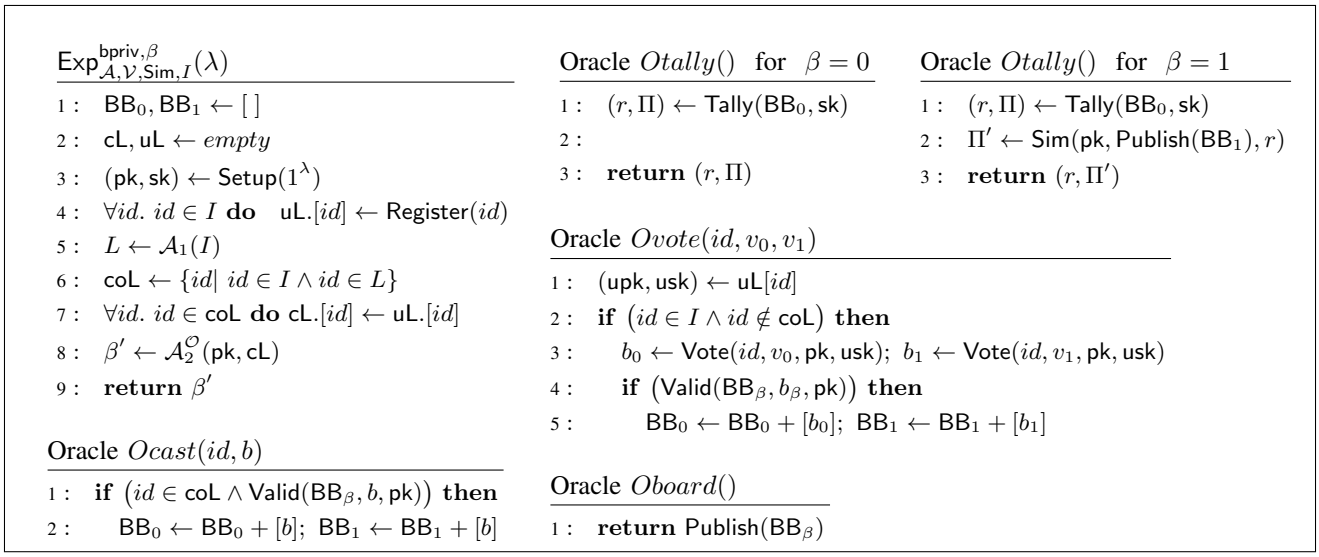

Figure 3. In the experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V},I}$, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has access to the set of oracles $\mathcal{O} = \{Ocast, Ovote, Otally, Oboard\}$. The adversary is allowed to call the $Otally$ oracle at most once.

analysis, we use the approach of Bernhard et al [13] who distinguish between three rather distinct aspects which impact the privacy of individual votes: ballot privacy, strong consistency and strong correctness. Together, the three notions imply security in a simulation based sense. We will say a voting scheme is private, if it meets all three properties. Below, we recall the intuition behind these definitions and explain how we adapt and extend them to the case of Belenios.

BALLOT PRIVACY PROPERTY. We start with ballot privacy which guarantees that the ballots themselves (as opposed to ballots together with other information that is published, e.g. the result of the election) do not reveal any information about the votes cast. For our analysis, we extend the notion to account for registration authority and for the use of credentials to cast votes. For reasons which we explain a bit later in the paper, we constrain our analysis only to the case of static corruption.

The formal definition is in Figure 3. It considers an adversary that attempts to distinguish between two worlds characterised by a hidden bit β. In the real world ($\beta = 0$) honest votes are cast as chosen by the users, and the final tally is performed honestly by the authorities; in particular the tally may come with additional information (e.g. a ZK proof that the tally was performed honestly). In the ideal world ($\beta = 1$) ballots submitted by honest parties contain fake votes. We emphasize that the tally is always performed on the real board (i.e. the BB_0): this captures the idea that the tally itself may reveal some information and that ballot privacy does not account for that information. Nonetheless, since the adversary always sees the "real" result, in the fake world (i.e. for $\beta = 1$), security demands the existence of an efficient simulator (who only has access to the visible part of the real board) and needs to produce the additional information in a way that is not distinguishable by the adversary. Within this setup, if for some scheme the ballots leak information about the underlying votes then clearly no simulator would exist, and the scheme would be deemed insecure, as desired. For more extensive intuition and discussion of ballot privacy we refer to the paper where it has been introduced and discussed and compared with other definitional attempts [13].

The security experiment starts by initializing various variables used throughout: the two boards BB_0 and BB_1, credentials for the users in the involved (those in I), and various bookkeeping lists. Next, the adversary decides on a list of users to corrupt (those in L) and obtains their credentials (maintained in list cL). The rest of the experiment models the execution of the protocol in the presence of the adversary. The capabilities of the adversary are modeled using the oracles in Figure 3, which we informally describe below:

$Ovote(id, v_0, v_1)$: models voting by honest parties. The adversary decides on two votes for user id; the oracle uses the secret credential usk of id to create ballots b_0 and b_1 for v_0 and v_1. The oracle then checks if ballot b_β is valid with respect to board BB_β; if this is the case then b_0 is added to BB_0 and b_1 is added to BB_1.

$Ocast(id, b)$: models ballot submission by corrupt users. If ballot b submitted on behalf of user id is valid with respect to board BB_β (which is the board which the adversary can see) then b is added to both BB_0 and BB_1.

$Oboard(\mathsf{BB})$: allows the adversary to access the public part of the bulletin board.

$Otally()$: returns to the adversary the result of the tally. The oracle computes the result on board BB_0 and additional information that depends on β: if $\beta = 0$ this is the information that the tally authorities would normally return. If $\beta = 1$ it this information is provided

by the simulator Sim.

Our formalism constrains the ballot privacy adversary to corrupt users only before the voting process starts. This constraint is introduced since natural extensions of the existing ballot privacy definition to incorporate registration authorities are too strong. Indeed, a simple replay of a honest ballot as a cast ballot for the same voter (after corruption) shows that most voting protocols would be insecure under such extensions, although this scenario does not indicate a real weakness in the system.

In brief, this scenario is as follows. The adversary asks that some honest, registered user id with public credential upk and secret credential usk submits a challenge (v_0, v_1) to its $Ovote$ oracle. BB_β now contains a single encrypted ballot (id, b_β). The adversary now corrupts id and recasts b_β on behalf of the voter, leading to the final bulletin board shown as lists of ballots at the beginning of Fig. 4. For schemes that allow revote and where the revote policy is to only consider the last ballot cast (like Belenios [20] and Helios [2]), the tally will trivially reveal β. Notice that this sequence of calls, in practice, does not reveal more information about the ballot than the information revealed by casting the vote only once. Somehow by chance, Helios, shown private in [13], [18], does not face this issue thanks to the weeding policy (duplicate ballots have to be removed to avoid known - real - attacks [22], [23]). Since there is no clear way to modify the ballot privacy definition in a way that does not make it unreasonably weak, in this paper we settle on proving security of Belenios under static corruptions and leave an investigation of alternative definitions as future work.

$\beta = 0$	$\beta = 1$
$\mathsf{BB}_0 = [(id, b_0), (id, b_0)]$	$\mathsf{BB}_1 = [(id, b_1), (id, b_1)]$
	$\mathsf{BB}_0 = [(id, b_0), (id, b_1)]$
$(v_0, \pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0)$	$(v_1, \pi) \leftarrow \mathsf{Tally}(\mathsf{BB}_0)$
Adversary sees board BB_0, result v_0, and real proof π	Adversary sees board BB_1, result v_1, and sim. proof π'

Figure 4. Replay with dynamic corruption.

Definition 1 (Ballot Privacy)**.** A voting scheme $\mathcal{V}$ has ballot privacy if there exists a simulator Sim such that no efficient adversary $\mathcal{A}$ can distinguish between the games $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V},\mathsf{Sim},I}(\lambda)$ and $\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V},\mathsf{Sim},I}(\lambda)$ defined in Figure 3. That is, the expression

$$\mathsf{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim},I}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V},\mathsf{Sim},I}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V},\mathsf{Sim},I}(\lambda) = 1\right]\right|$$

is negligible in λ, for any set of voters I.

STRONG CONSISTENCY. The second notion proposed by Bernhard et al [13] is *strong consistency*. Informally, this notion demands that for any (adversarially produced) bulletin board for which each individual ballot is valid, the result provided by the Tally algorithm is the "correct" one. That is, the result is the same as the one obtained by decrypting each individual vote, filtering them using the a filtering policy Policy that is in place and applying the desired result function Count.

The original definition associated voter identities to ballots; in this paper we refine the notion and use their public credentials instead.

Definition 2 (Strong Consistency [13])**.** Informally, a voting scheme is *strongly consistent* if an adversary cannot construct a bulletin board containing only valid ballots in such a way that applying the tally algorithm to the board yields a different result than induced by the votes underlying the ballots. To formalize this notion we demand that there exist:

- An extraction algorithm $\mathsf{Extract}(b, \mathsf{sk})$ that provides the upk with the value computed by decrypting of the ciphertext contained in the ballot b with the secret sk; and
- A ballot validation algorithm $\mathsf{ValidInd}(b, \mathsf{pk})$ that returns true iff the ballot b is "well-formed" with respect to some notion of well-formedness determined in advance by the election.

These algorithms must satisfy the following conditions:

1) For any $(\mathsf{pk}, \mathsf{sk})$ obtained from $\mathsf{Setup}(\lambda)$, and any (id, v, usk) with upk trivially computed from usk, if $b \leftarrow \mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$ then $\mathsf{Extract}(b, \mathsf{sk})$ returns (upk, v) with overwhelming probability.

2) For any adversarially produced (BB, b), if $\mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$ returns true, then $\mathsf{ValidInd}(b, \mathsf{pk})$ returns true as well.

3) For any adversary $\mathcal{B}$ that returns a ballot box with ballots that satisfy ValidInd, the experiment $\mathbf{Exp}^{\mathsf{consis}}_{\mathcal{B},\mathcal{V},I}(\lambda)$ specified in Figure 5 returns true with a probability negligible in the security parameter. In the game, the adversary is allowed to register a set of identities of its chosing and then, for honestly generated keys for the election (pk, sk) come up with a board BB such that the result of the tally r is different from the extracting the votes underlying the ballots, applying the weeding policy of the election and then applying the result function to the votes left.

$\mathsf{Exp}^{\mathsf{consis}}_{\mathcal{B},\mathcal{V},I}(\lambda)$

1 : $\mathsf{uL} \leftarrow \mathsf{empty}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $\forall id \in I$ **do** $\mathsf{uL}.[id] \leftarrow \mathsf{Register}(id)$
4 : $\mathsf{BB} \leftarrow \mathcal{B}(\mathsf{pk}, \mathsf{uL})$
5 : $e \leftarrow \forall b \in \mathsf{BB}.\ \mathsf{ValidInd}(b, \mathsf{pk})$
6 : $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$
7 : **for** i **in** $1..|\mathsf{BB}|$ **do**
8 : $\quad dbb[i] \leftarrow \mathsf{Extract}(\mathsf{BB}[i], \mathsf{sk})$
9 : $r' \leftarrow \mathsf{Count} \circ \mathsf{Policy}(dbb)$
10 : **return** $(r \neq r' \wedge e)$

Figure 5. The Strong Consistency experiment

STRONG CORRECTNESS. Finally, strong correctness is the guarantee that an honestly created ballot will not be rejected

by a ballot validation algorithm. The existing definition [13] demands that this property holds *even* with respect to boards that are created maliciously. While natural, this requirement is too strong for systems where there is a registration authority, and where validity of ballots depends on wether their associated credentials having been used already. For example, in the case of Belenios, an adversary could easily win the game by returning a bulletin board which contains some identity id with a credential that does not belong to id and a ballot: any further vote by id with his own credential is rejected. In this paper we propose an extension of strong correctness which incorporates in the security experiment the condition that the board that the adversary returns satisfies some minimal well-formedness condition.

The formal definition uses the experiment in Figure 6. It considers an adversary Belenios who after seeing the list of credentials (honestly generated by the registration authority), it choses a voter id, a vote v and outputs a bulletin board BB (lines 1-4 from the experiment). The experiment will check if an honestly created ballot by id for vote v will be accepted as valid with respect to BB. The minimal condition that BB needs to satisfy is that any ballot present on BB which involves either id or its credentials must involve both. The adversary wins if for such a board, the honest vote for id is invalid.

$\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda)$

1 : $\mathsf{uL} \leftarrow \mathsf{empty}$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $\forall id \in I$ **do** $\mathsf{uL}.[id] \leftarrow \mathsf{Register}(id)$
4 : $(id, v, \mathsf{BB}) \leftarrow \mathcal{B}(\mathsf{pk}, \mathsf{uL})$
5 : $e \leftarrow \mathsf{false}$
6 : **if** $(id \in I)$ **then**
7 : $\quad(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{uL}.[id]$
8 : $\quad e_1 \leftarrow \forall x.\forall y.\ (x, y, \star, \star) \in \mathsf{BB} \implies$
9 : $\quad\quad (x, y) = (id, \mathsf{upk}) \vee (x \neq id \wedge y \neq \mathsf{upk})$
10 : $\quad b \leftarrow \mathsf{Vote}(id, v, \mathsf{pk}, \mathsf{usk})$
11 : $\quad e_2 \leftarrow \mathsf{Valid}(\mathsf{BB}, b, \mathsf{pk})$
12 : $\quad e \leftarrow e_1 \wedge \neg e_2$
13 : **return** e

Figure 6. The Strong Correctness experiment

Definition 3 (Strong Correctness)**.** A voting scheme $\mathcal{V}$ is *strongly correct* if the advantage of any efficient adversary $\mathcal{B}$, defined by $\mathcal{A}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda) = \Pr[\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda) = 1]$ (where $\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V},I}(\lambda)$ is defined in Figure 6) is negligible as a function of λ.

3.2. Verifiability

At the very least, a verifiable voting schemes should allow voters to check that their vote contributed to the outcome of the election. This is usually done in a individual way: each user checks the presence of their ballot in the ballot box. In addition, it should also be possible for voters (or, better, for any third party) to check whether the published result of the election corresponds to the votes cast by voters (and recorded in the ballot box). Various definitions of verifiability have been proposed [21]. We consider here the notion of *strong verifiability* introduced in [20] for Helios-C, a preliminary version of Belenios. It captures individual, universal, and eligibility verifiability and accounts for the fact that voters may not perform the verifiability checks. As in [20], we consider two corruption scenarios:

1) a dishonest ballot box with an honest registrar;
2) a honest ballot box with a dishonest registrar.

In both cases we assume the tallying authorities may misbehave.

VERIFIABILITY AGAINST DISHONEST BALLOT BOX. The security game corresponding to a dishonest ballot box and an honest registrar is depicted in Figure 7. After an honest registration phase, the adversary entirely controls the ballot box and has access to two oracles:

- $\mathsf{O}_{vote}(id, v)$ to require that some honest voter id votes for v. The fact that voter id voted is stored in Hvote.
- $\mathsf{O}_{corrupt}(id)$ to corrupt voter id and retrieve her secret credential. Previous votes from id are removed from Hvote.

Then the adversary publishes the final ballot box and voters may check whether their ballot has been included in the ballot box (through oracle $\mathsf{O}_{check}(id)$). If the test is successful, the corresponding voter (and vote) is stored in Checked.

The game $\mathsf{Exp}^{dbb}_{\mathcal{A},\mathcal{V},I}(\lambda)$ guarantees that the election result corresponds to:

- all the votes from the honest voters who checked (set C);
- some of the votes from the honest voters who did not check (the adversary may always remove such votes). This corresponds to the set $L \subseteq H$;
- at most k other (valid) votes, where k is the number of corrupted voters (set D). This last condition controls ballot stuffing: any adversary may be able to choose as many votes as the number of corrupted voters but not more. In particular, he cannot add arbitrary votes to the results.

Formally, a voting scheme $\mathcal{V}$ is *verifiable against dishonest ballot box*, if for any set I of voters and for any efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the probability

$$\Pr\left[\mathsf{Exp}^{dbb}_{\mathcal{A},\mathcal{V},I}() = 1\right]$$

to win the experiment depicted in Figure 7 is negligible.

VERIFIABILITY AGAINST DISHONEST REGISTRAR. The security game corresponding to a honest ballot box and a dishonest registrar is depicted in Figure 8. It is very similar to the previous game except that the ballot box now behaves honestly and the adversary choses the credentials of the voters. Again, the result has to contain all the votes from honest voters who checked, a subset of the votes of voters who did not check and at most k arbitrary valid votes from the k dishonest voters. A voting scheme $\mathcal{V}$ is *verifiable*

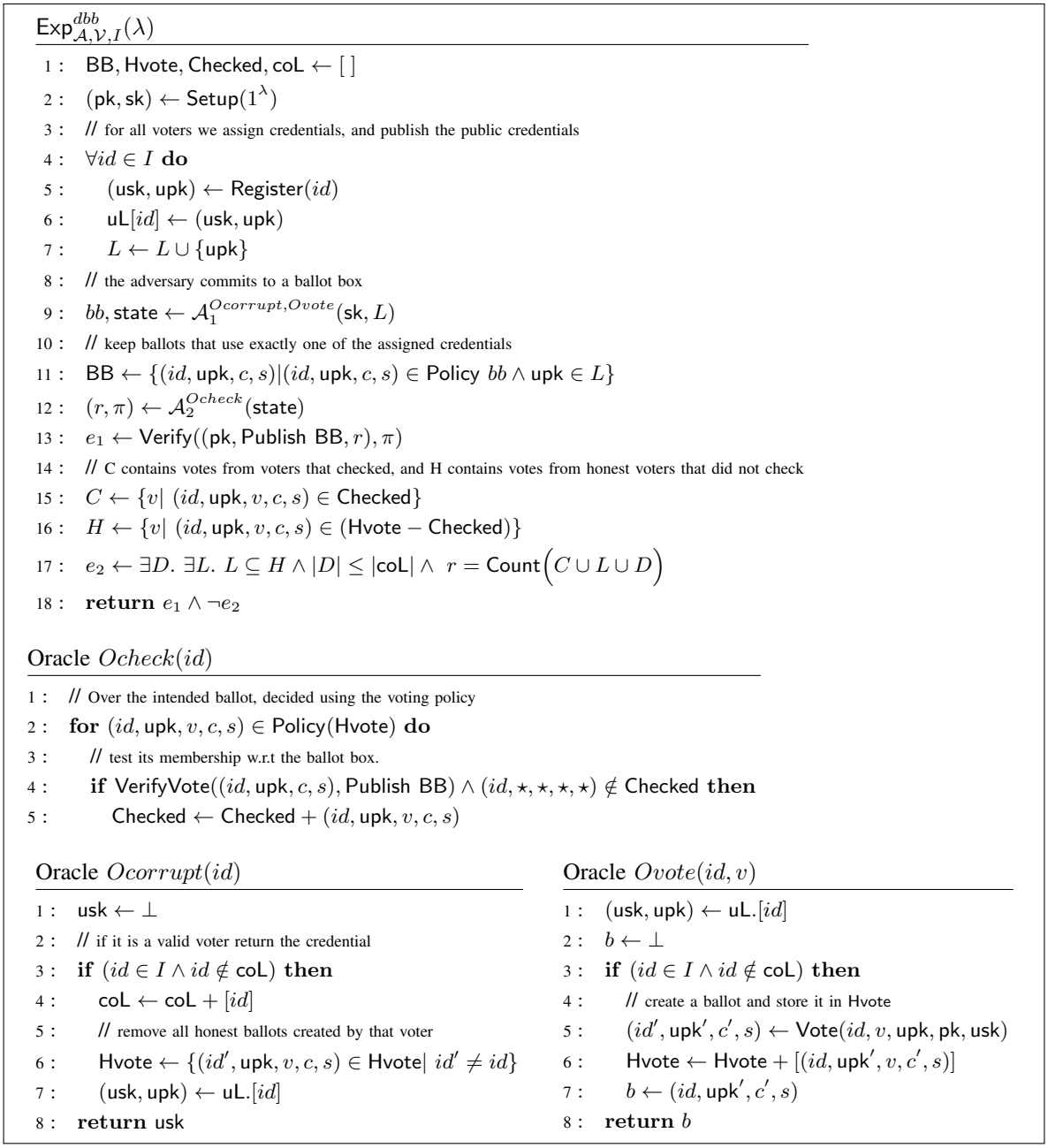

Figure 7. The verifiability experiment against a dishonest ballot box. In experiment $\mathsf{Exp}^{dbb}_{\mathcal{A},\mathcal{V},I}$, the adversary $\mathcal{A}_1$ has access to the set of oracles $\mathcal{O} = \{Ocorrupt, Ovote\}$, and $\mathcal{A}_2$ has access to $Ocheck$.

against dishonest registration, if for any efficient adversary $\mathcal{A}$ the probability

$$\Pr\Big[\mathsf{Exp}^{dreg}_{\mathcal{A},\mathcal{V},I}() = 1\Big]$$

to win the experiment depicted in Figure 8 is negligible.

DISCUSSION. Compared to the original definition of [20], we made three main changes. First, we now provide the secret key of the election to the adversary. Indeed, for Belenios as well as many other voting schemes, verifiability does not rely on the security of the key used to encrypt the votes. Hence we propose a stronger definition that states that verifiability is guaranteed even when the election key is corrupted.

Second, we clarify the definition of the set Checked that records which voters have checked, making sure that we only consider the final ballot box. Indeed, in case a voter re-votes (e.g. first for v_1 and then v_2), she should check that her last cast ballot belongs to the ballot box (and not the first one). What to do in case of revote was not specified in [20]. In practice, it may be cumbersome for voters to

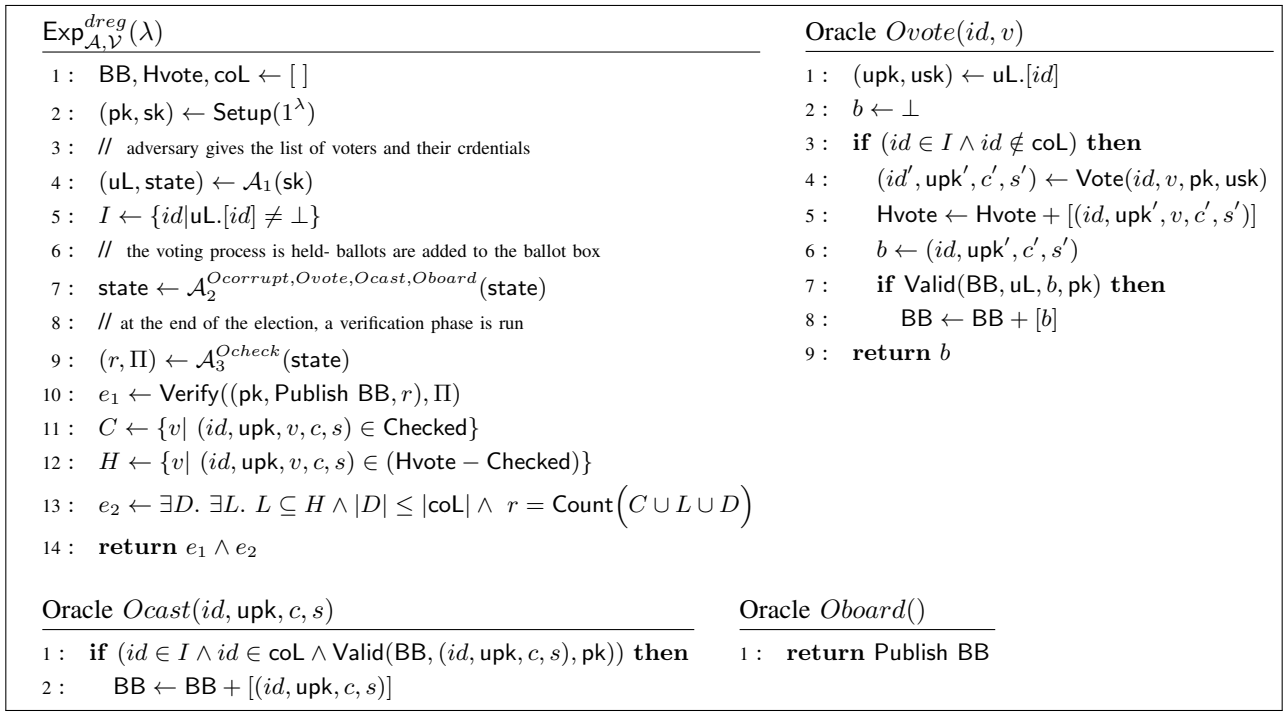

Figure 8. The verifiability experiment against a dishonest registration. In experiment $\mathsf{Exp}^{dreg}_{\mathcal{A},\mathcal{V}}$, the adversary $\mathcal{A}_2$ has access to the set of oracles $\mathcal{O} = \{Ocorrupt, Ovote, Ocast, Oboard\}$, and adversary $\mathcal{A}_3$ has access to $Ocheck$. $Ocheck$ and $Ocorrupt$ are those shown in Figure 7

wait for the final box to make their check. This condition may often be relaxed, assuming the presence of auditors that ensure that the ballot box grows consistently (i.e. no ballots are removed).

Finally, when the registrar is dishonest, even honestly generated ballots may not be valid. For example, several voters may receive the same credentials, or some voters may receive invalid credentials. Therefore, we now reflect that, although a voter may believe she has voted properly (recorded in the set Hvote), her ballot should be included in the ballot box only if it is valid.

4. Belenios

In this section, we present the Belenios voting system and state our main results, i.e. machine-checked proofs of privacy and verifiability for this protocol. We keep the description as close as possible to the current implementation of the protocol while keeping it compatible with the syntax of a single-pass voting system.

The core cryptographic primitive which enables privacy preserving tallying in Belenios is a labelled public-key encryption scheme constructed by composing El Gamal encryption with a Chaum-Pedersen proof of knowledge. We model this composition by considering an abstract homomorphic encryption scheme $\mathcal{E}' = (\mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{Add}')$ and an abstract proof system $\Sigma'_{\mathcal{R}'} = (\mathsf{P}', \mathsf{V}')$ and building a labelled encryption scheme $\mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'}) = (\mathsf{KGen}', \mathsf{Dec}, \mathsf{Enc})$ as shown in Figure 10. Typically, the proof system is used to prove that the vote v is valid, also including the voter's public credentials inside the statement to link the creation of the non-malleable ciphertext with the signature.

Belenios is also parameterized by a hash function Hash used to display a fingerprint of the ballots (easier to check for a voter).

The Belenios Voting Scheme. Belenios is constructed around a labelled public-key encryption scheme. The formal description of the algorithms that define the protocol is in Figure 9. Here we provide an overview of how these algorithms work.

Definition 4. Let $\mathcal{E} = \mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$ be a labelled public-key encryption scheme constructed as shown in Figure 10 from a homomorphic encryption scheme $\mathcal{E}'$, and a proof system $\Sigma'_{\mathcal{R}'}$. Consider a signature scheme $\Pi = (\mathsf{SKGen}, \mathsf{Sig}, \mathsf{Vf})$, a second proof system $\Sigma_{\mathcal{R}} = (\mathsf{P}, \mathsf{V})$ whose relation $\mathcal{R}$ holds if the result corresponds to the ballot box, and a hash function Hash over the public credential, labelled ciphertext and signature. Given an abstract revote policy Policy, we define the Belenios voting scheme

$$\mathsf{Belenios}[\mathsf{Policy}]$$

as the single-pass voting scheme defined by the algorithms shown in Figure 9.

Less formally, the Belenios algorithms operate as follows:

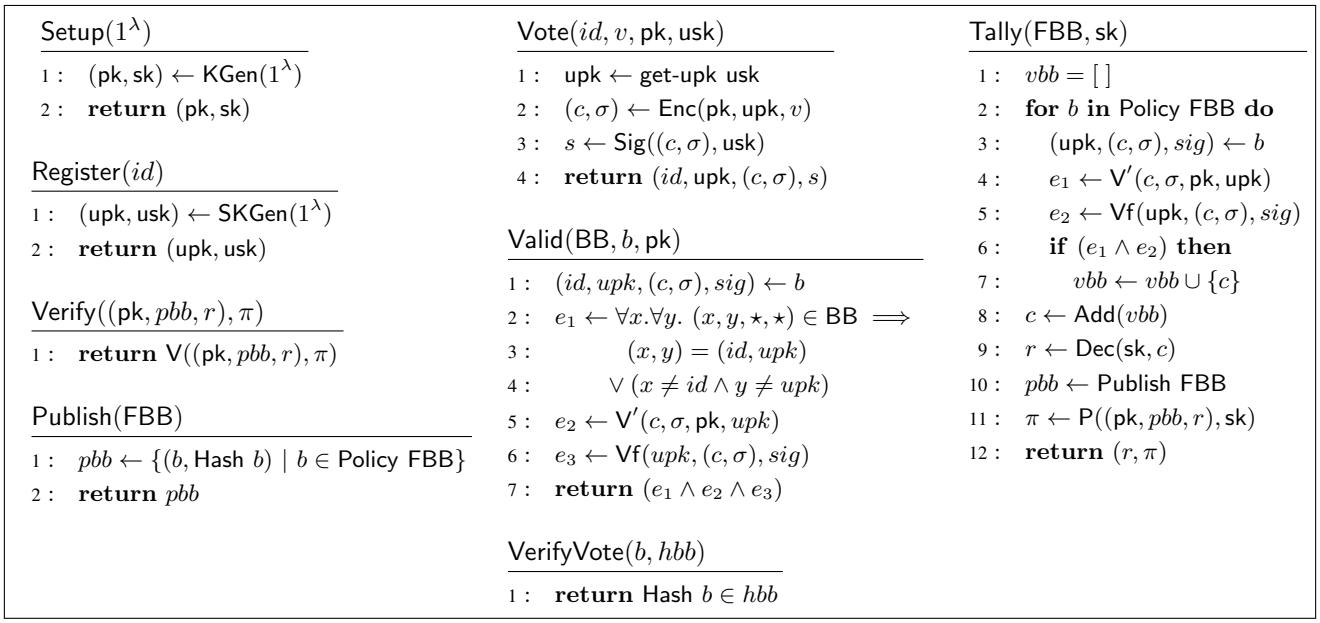

Figure 9. Algorithms defining the Belenios Belenios[Policy] scheme. BB is stored internally, while FBB = $\{(upk, cip, sig) | (id, upk, cip, sig) \in \mathsf{BB}\}$ is given to the talliers to compute the result of the election.

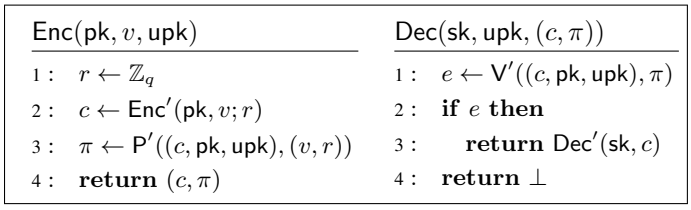

Figure 10. Constructing labelled public-key encryption scheme from homomorphic encryption and PoK: $\mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$.

Setup(1^λ)**:** Generates a pair of keys (pk, sk) by calling the encryption scheme $\mathcal{E}$'s key generation algorithm KGen(1^λ).

Register(id)**:** Creates a pair of signing keys (upk, usk) for the voter id, using the signature scheme Π's key generation algorithm SKGen. We assume that the public credential upk can be efficiently recovered given the secret credential usk.

Vote(id, v, pk, usk)**:** Returns, for a voter id with credentials (upk, usk), a ballot of the form (id, upk, (c, σ), s). The ciphertext (c, σ) is the output of Enc(pk, v, upk), where Enc is the encryption algorithm of the labelled public-key encryption scheme $\mathcal{E}$. The signature s is computed over (c, σ) using the signature scheme Π, as the output of Sig(usk, (c, σ)).

Valid(BB, (id, upk, (c, σ), s), pk)**:** Evaluates the *validity* of the ballot (id, upk, (c, σ), s). First, it checks that the proof and the signature by calling V′(c, σ, pk) and Vf(upk, (c, σ)). Then, it further checks that there does not exist a ballot in BB that uses either: i) the same identity id with a different credential upk′ ≠ upk, or ii) a different identity $id' \neq id$ with the same credential upk.

Box(BB, b)**:** Returns the ballot box BB ∪ $\{b\}$ if Valid(BB, b, pk) holds; and BB otherwise.

Publish(FBB)**:** Publishes information about the content of a stripped ballot box that does not contain voter identities. First, a core list of ballots are obtained in C by applying a filtering policy based on the public credential.

$$C = \mathsf{Policy}(\mathsf{FBB})$$

Then, every ballot is extended by adding a hash over it, and publishing the result

$$PBB = \{(b, \mathsf{Hash}(b)) \mid b \in C\}.$$

Tally(FBB, sk)**:** Returns the result and a proof that the tally was computed correctly by calling the prover P. At tallying time, the ballot box BB′ contains only stripped ballots that do not contain identities, and have the form (upk, (c, σ), s). First, all ballots that contain invalid proofs or invalid signatures are removed. The revote policy Policy is then used to filter the ballots. Finally, the Add algorithm is applied to the ciphertexts in the remaining ballots to homomorphically compute a single ciphertext that gets decrypted to obtain the election's result.

Verify((pk, pbb, r), Π)**:** Calls the verify algorithm of the verifier V, to check that the tally corresponds to the public bulletin board.

VerifyVote(b, pbb)**:** Checks that the ballot b belongs to the public bulletin board. This check is made lighter by only verifying that the ballot's hash appears on the bulletin board, rather than the entire ballot.

Security assumptions. In the two following sections we present our main results concerning the privacy and verifiability of the Belenios voting scheme. For all these theorems, we provide machine-checked proofs in EasyCrypt [6].

Throughout, and to avoid repetition, we assume the following of the primitives used in the construction of Belenios:

- the encryption scheme $\mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$ is IND-1-CCA and well-spread;
- the proof system $\Sigma_{\mathcal{R}}$ (used to prove correct tallying) is zero-knowledge, a proof of knowledge and complete;
- the signature scheme Π is strongly existentially unforgeable, and
- Hash is a collision-resistant hash function.

In addition, we note that our proof is in the random oracle model. In particular, the hash functions used in the zero-knowledge proof systems are modelled as random oracles. As discussed by Cortier et al. [18], the use of random oracles which often simplifies pen-and-paper proofs considerably, it leads to significant complexity for machine-checked proofs. We do not insist on this aspect in this paper.

Security of Belenios. We can now formally state the security properties of Belenios. We consider two common cases for the revote policy Policy:

1) first. Only the first valid vallot cast by each voter is kept and tallied. This corresponds to the case where revotes are forbidden.
2) last. Only the last valid ballot cast by each voter is kept and tallied. This option is the policy implemented by Belenios.

Belenios is ballot private and verifiable for both policies.

Theorem 1. *Under the assumptions stated above the voting scheme* Belenios[Policy] *(for* Policy $\in \{\mathsf{first}, \mathsf{last}\}$*) satisfies*

- *privacy – i.e. ballot privacy, strong correctness and strong consistency;*
- *strong verifiability against a dishonest ballot box (with honest registration);*
- *strong verifiability against a dishonest registration (with honest ballot box).*

Here we give an overview of the main ideas behind the proof, with the EasyCrypt proof closely following standard cryptographic practice.

PRIVACY. Strong correctness and strong consistency immediately follow from the definition of Belenios.

For ballot privacy, we follow the same two-step proof strategy as for the existing analysis of Helios [18]. First, we replace the real zero-knowledge proof with a simulated one. Then, we use the security of the encryption scheme to change the view of the adversary on the ballot box with one which maintains no information on the honest votes. It is worth noting that since we consider a static corruption model, some of the proof aspects are simpler than the corresponding ones in [18].

STRONG VERIFIABILITY AGAINST DISHONEST BALLOT BOX. To show that Belenios satisfies this property we need to show that the adversary has to provide the correct result and proof for the ballot box he is committed too. We split this property in two distinct ones which we call tally uniqueness and accuracy (See Appendix A for details). In turn, we show that these two properties hold under the completeness and the proof of knowledge property of the proof system.

Part of the proof deals with ensuring the correctness of the result. Specifically, we need to show that the votes of honest voters are correctly tallied and that the total number of dishonest votes in the ballot box does not exceed that of the dishonest voters. Clearly, the first part of the statement holds only for those voters who check that their vote is present; under the collision resistance property of the hash function Hash if the check succeeds, then the ballot which they have cast is in the ballot box. To argue the upper bound on the number of dishonest votes cast, we use the unforgeability of the signature scheme: the only votes that are in the ballot box need to correspond to a valid credential.

STRONG VERIFIABILITY AGAINST DISHONEST REGISTRAR. To prove strong verifiability when the ballot box is honest but the registrar may misbehave, we proceed similarly in the case of honest voters (e.g. to show that their votes are correctly recorded, whenever they check that this is the csae). To bound the number of dishonest votes, we rely on the honesty of the ballot box: the adversary may only add valid votes by calling the $Ovote$ oracle (which only adds honest ballots) or by calling $Ocast$, which adds a ballot in the name of a dishonest voter. Then the policy (first or last) guarantees that at most one ballot per dishonest voter is counted.

Extension to scheme variants. Our approach also covers several interesting election-variants of Belenios, where the talliers may have access to the entire ballot box (including the link between voters and their cast ballot). Specifically, we provide a proof for a generalized version of Belenios by abstracting two of the algorithms which specify the details of how the scheme operates. Specifically:

Publish-policy (PubPol, for short) The information made available on the public bulletin board may vary.

1) identity. A liberal choice is to publish all the available information.

$$\mathsf{PubPol}(L) = L.$$

2) policy-view. Another possibility is to reveal only the relevant information, by publishing the exact ballots used to compute the result.

$$\mathsf{PubPol}(L) = \mathsf{Policy}(L).$$

Anonymize (Anon, for short) Whether or not the identity of the voter should be part of the public ballot depends on the election rules. Anonymize selects if the identity of the voter is made public or kept hidden.

1) false. Returns the entire ballot. The talliers (and the voters by means of PubPol) have access to the link between a voter and his cast ballot.

$$\mathsf{Anon}((id, upk, cip, sig)) = (id, upk, cip, sig)$$

2) true. Returns the ballot without the voter identity. The identities of the voters are kept secret, and the talliers do not have access to information like who voted.

$$\mathsf{Anon}((id, upk, cip, sig)) = (upk, cip, sig)$$

Additionally, we now consider the hash function Hash, used in defining Belenios (Figure 9), to take an additional input: the voter identity or an empty element depending on the instantiation of Anon. This is just a method of ensuring the same input type for the hash function is provided throughout our definitions and proofs.

Using the same syntax than for the definition of Belenios, we introduce the voting scheme

$$\mathsf{Belenios}'[\mathsf{PubPol}, \mathsf{Anon}, \mathsf{Policy}]$$

- where Setup, Register, Vote, Verify, Valid have been defined in Figure 9;
- the definition of Tally remains unchanged from Figure 9, with the exception that it takes input BB (instead of FBB) and line 3 where the ballot is of form $(id, upk, (c, \sigma), sig)$;
- and Publish, VerifyVote are modified as described in Figure 11. The algorithm Publish publishes the ballots according to PubPol and Anon, the algorithm VerifyVote is updated accordingly.

In particular,

$$\mathsf{Belenios}[\mathsf{Policy}] = \mathsf{Belenios}'[\text{policy-view}, \mathsf{true}, \mathsf{Policy}].$$

Publish(BB)

1 : $pbb \leftarrow \{(\mathsf{Anon}(b), \mathsf{Hash}(\mathsf{Anon}(b)))|b \in \mathsf{PubPol}(\mathsf{BB})\}$
2 : **return** pbb

VerifyVote(b, pbb) for PubPol = policy-view

1 : **return** $\mathsf{Hash}(\mathsf{Anon}(b)) \in \{h|(b, h) \in pbb\}$

VerifyVote(b, pbb) for PubPol = identity

1 : **return** $\mathsf{Hash}(\mathsf{Anon}(b)) \in \{h|(b, h) \in \mathsf{Policy}(pbb)\}$

Figure 11. The algorithms Publish, VerifyVote of the voting scheme Belenios′[PubPol, Anon, Policy].

Our proof extends to the generalised voting scheme Belenios′[PubPol, Anon, Policy], under the same cryptographic assumptions.

Theorem 2. *Let* $\mathcal{V} = \mathsf{Belenios}'[\mathsf{PubPol}, \mathsf{Anon}, \mathsf{Policy}]$. *Under the assumptions stated above (earlier in this section)* $\mathcal{V}$ *is private (ballot private, strongly correct and strongly consist), strongly verifiabley against a dishonest ballot box (with honest registration), and strongly verifiable against a dishonest registration (with honest ballot box) if :*

- $\mathsf{PubPol} \in \{\mathsf{identity}, \text{policy-view}\}$;
- $\mathsf{Anon} \in \{\mathsf{true}, \mathsf{false}\}$;
- $\mathsf{Policy} \in \{\mathsf{last}, \ \mathsf{first}\}$,

5. Formal Proofs, our Results and their Impact

We now discuss our formally verified results touching upon the effort involved and lessons learned. All the results presented in this paper are backed by machine-checked proofs in EasyCrypt. In Table 1, we detail the size, similarity and development effort for each of the proof components; we also give a rough indication of their complexity. We break down the information in 4 categories: **General Concepts** cover definitions for the underlying primitives and assumptions; **Privacy** covers the definitions and proofs for the three privacy-related properties for Belenios; **Verifiability** covers the definitions and proofs for the verifiability properties; and **Variants** covers the adaptation of the proofs to election-variants of Belenios.

One first observation is how much of the existing Helios privacy proof [18] we could reuse. It turns out that even with the reasonably big changes to the security definitions which we have adapted to the case of Belenios, we could reuse much of the proof of privacy (about 75%). [2] The main difficulties faced in extending the proof of privacy to Helios are in fact related to the definitional challenges including the registrar involved. Perhaps more interestingly, the proof of verifiability against a dishonest ballot box and the proof of verifiability against a dishonest registrar are very similar despite the differences in the definitions of these two properties and in the assumptions they rely on. Overall, the Trusted Computing Base (TCB) of the formal development, consisting of definitions and axioms, saw the addition of 1188 new (or modified) lines of code compared to the previous TCB for Helios. This includes definitions related to signature schemes (217 lines) and the security definitions for verifiability (865 lines), but also the extensions to privacy notions that let them capture the registration phase (106 lines).

Finally, we note that we intentionally made little effort to make our approach more modular: the proofs for election-variants of Belenios are obtained as modifications of the Belenios code-base rather than instances of a more general approach. Effort towards a more modular approach would certainly be useful in identifying interesting properties that underlie verifiability, in general. However, such an effort would be premature. As we highlight in this paper, there is still work to be done to stabilise definitions of verifiability, as illustrated throughout the paper. For example, extending definitions to voter registration and key distribution had unexpected consequences even on the definition of ballot privacy.

Table 1 also presents the estimated development effort (in person-weeks) involved in each of the proof components, with the Helios development [18] as a starting point.

BENEFITS OF MACHINE-CHECKED PROOFS. EasyCrypt [6] is an interactive proof assistant with built-in support for

2. The low degree of similarity in General Concepts accounts mainly for the addition of security properties for the signature, and a generic reduction to multi-challenge unforgeability with corruption. Many of the base definitions are in fact common.

TABLE 1. DEVELOPMENT STATISTICS FOR THE BELENIOS PROOF

Belenios	LoC	Ver. Time (s)	Code Sim. (%)	Dev. Effort (PW)
General Concepts	5936	348	55% Helios	4
Privacy	2700	238	75% Helios	2
Verifiability	14590	1523	-	20
Variants	47030	3965	95% Belenios	1

expressing cryptographic security definitions (game-based and simulation-based) and formalising cryptographic proofs in a language that is familiar to cryptographers. Security definitions are modelled as probabilistic programs, parameterised by other probabilistic programs representing the adversary and the oracles it is given access to. Security proofs are then modelled as sequences of conditional equivalences between these programs, in a way similar to that introduced by Bellare and Rogaway [10].

Given an EasyCrypt proof (or other machine-checked proof), a reader who trusts the automated proof checker only has to read and understand the definitions and theorem statements to be convinced of the results. The asymmetry this introduces between the proof writer and the proof reader is particularly appealing in a domain such as cryptography, where reading a proof often requires a level of expertise almost equal to that of the proof writer. Although machine-checked proofs do not alleviate the need to understand the formalised definitions, being able to only focus on understanding the definition and completely bypass the need to understand the proofs may be a significant step towards better acceptance and adoption. Other tools such as F* [7] or FCF [30] have been used to formalise cryptographic security proofs. Among them, EasyCrypt has demonstrated its ability to both handle large and complex protocols, and at the same time support cryptographic definitions that remain close in style and spirit to their pen-and-paper specifications. If the former is necessary for protocols and properties as complex as those involved in electronic voting, the latter is particularly valuable when trying to understand the effect of small adjustments to security definitions on security notions and trust assumptions.

LESSONS LEARNED AND RECOMMENDATIONS. From the formalization experiment we report here, we highlight a few lessons that we believe could be of use to future efforts–on or away from electronic voting.

First, our experience highlights the value of abstraction, despite its initial cost. In particular, plugging a formal definition of Belenios into the formal definitions of privacy developed by Cortier et al. [18] was instrumental in quickly identifying that the definitions of privacy were inappropriate. We did this expecting to simply work, as a trivial validation of the privacy properties of Belenios. This ability to quickly check a belief–rather than trusting that similarity implies equivalence–is a great benefit of formal proofs, and is mainly enabled by the development of abstract and general definitions.

This same ability to quickly check believed results is also what led to the early identification of the definitional issues discussed in this paper. There is a great need for clear security definitions for electronic voting, that can serve as (perhaps parameterized) benchmarks for the critical evaluation of competing solutions. However, these security definitions *must* capture essential details of the voting system that are usually left out of security analyses.

One final lesson, learned early on, is that working with simulation based notions in the random oracles is not as easy as traditionally assumed: one cannot simply add and remove them as they please. There are real security concerns when dealing with split random oracles, part of which can be taken over by a simulator, while others remain independent. Fiore and Nitulescu [27] take first steps towards a more thorough analysis of these issues and solutions to them.

6. Discussion and Conclusion

In this paper, we have refined existing formal definitions for privacy and verifiability of electronic voting protocols including a registrar, and have developed machine-checked proofs that Belenios meets the resulting properties.

Our proof also extends to a few election variants of Belenios beyond its current deployments, considering in particular different revote policies, and different policies regarding the data published on the public bulletin board. These considerations may make Belenios deployable in settings in which its current implementation choices are unacceptable due to regulations or bylaws in place.

More importantly, our formalization identifies several issues that are not captured by existing definitions of privacy and verifiability – in particular when protocols employ registration authorities. Although we solve these issues for the case of Belenios, future work is needed to better understand these issues in general, and develop formal definitions for these properties that apply more widely and can thus be used as benchmarks to evaluate the security of voting solutions in practice. In particular, the need for an additional trust assumption – that the registrar is trusted – is slightly unsatisfactory. Indeed, one would hope that, by splitting the role normally fulfilled solely by the ballot box over two different actors, the trust could be split equally. However, Belenios only achieves the stronger notions of ballot privacy when both ballot box and registrar are trusted. It would be interesting to consider new protocols that would provide ballot privacy as long as the ballot box and registrar do not collude, as is the case for verifiability.

In addition, although we consider election variants that may offer differing security guarantees in terms of coercion-resistance or protections against other specialised attacks, our proofs only consider the basic notions of privacy and verifiability. Formalizing other (stronger) security notions that can better serve to evaluate and compare the practical security of various voting systems – including their election-specific rules – is an interesting direction for further work.

We note that this study illustrates the feasibility of developing machine-checked proofs to study the security of deployed electronic voting systems, and encourages similar

developments for other schemes. We strongly believe that machine-checked proofs, if they are not more readable than pen-and-paper proofs–and even if they do increase the proof writer's burden, do make it easier for non-experts to gain trust in complex cryptographic systems.

In the context of deploying real-world voting systems in particular, such trust is crucial, and could be further reinforced by leveraging machine-checked formal specifications to develop formally-verified independent tally verifiers for specific elections. Doing so would require refining the proofs to cover details of parsing and formatting, including potential checks that must be performed when dealing with concrete representations of cryptographic values.

ACKNOWLEDGEMENTS. This work was partially supported by EPSRC research grant EP/P031811/1 (Trusted and Transparent Voting Systems) and by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement No 645865-SPOOC).

The authors wish to thank the rest of the EasyCrypt development team, with particular thanks to Benjamin Grégoire and Pierre-Yves Strub, for their continued support of the tool. We thank the anonymous reviewers and others involved in the peer-reviewing process for helpful suggestions of improvements to the paper and its presentation.

References

[1] Belenios webpage. http://www.belenios.org/.

[2] B. Adida. Helios: Web-based open-audit voting. In P. C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348. USENIX Association, 2008. Helios website: heliosvoting.org.

[3] J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, B. Grégoire, V. Laporte, and V. Pereira. A fast and verified software stack for secure function evaluation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1989–2006, 2017.

[4] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 83–107. Springer, 2002.

[5] G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 689–718. Springer, 2015.

[6] G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub. Easycrypt: A tutorial. In A. Aldini, J. Lopez, and F. Martinelli, editors, *Foundations of Security Analysis and Design VII*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer International Publishing, 2014.

[7] G. Barthe, C. Fournet, B. Grégoire, P. Strub, N. Swamy, and S. Zanella-Béguelin. Probabilistic relational verification for cryptographic implementations. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 193–206, 2014.

[8] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *Annual International Cryptology Conference*, pages 390–420. Springer, 1992.

[9] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[10] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 409–426, 2006.

[11] M. Bellare and A. Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In *Annual International Cryptology Conference*, pages 519–536. Springer, 1999.

[12] D. Bernhard. *Zero-Knowledge Proofs in Theory and Practice*. PhD thesis, University of Bristol, 2014.

[13] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. 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.

[14] D. Bernhard, M. Fischlin, and B. Warinschi. Adaptive proofs of knowledge in the random oracle model. *IET Information Security*, 10(6):319–331, 2016.

[15] D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 626–643. Springer, 2012.

[16] D. Bernhard and B. Smyth. Ballot secrecy with malicious bulletin boards. Cryptology ePrint Archive, Report 2014/822, 2014.

[17] B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *20th International Conference on Automated Deduction (CADE-20)*, Tallinn, Estonia, July 2005.

[18] V. Cortier, C. C. Drăgan, F. Dupressoir, B. Schmidt, P.-Y. Strub, and B. Warinschi. Machine-Checked Proofs of Privacy for Electronic Voting Protocols. In *IEEE Symposium on Security and Privacy, S&P 2017*, 2017.

[19] V. Cortier, C. C. Drăgan, F. Dupressoir, and B. Warinschi. EasyCrypt proofs of privacy and verifiability for Belenios. https://gitlab.com/ec-evoting/belenios.

[20] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. Election verifiability for helios under weaker trust assumptions. In M. Kutylowski and J. Vaidya, editors, *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2014.

[21] V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung. Sok: Verifiability notions for e-voting protocols. In *36th IEEE Symposium on Security and Privacy (S&P'16)*, pages 779–798, San Jose, USA, May 2016.

[22] V. Cortier and B. 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.

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

[24] V. Cortier and C. Wiedling. A formal analysis of the norwegian e-voting protocol. In *Proceedings of the 1st International Conference on Principles of Security and Trust (POST'12)*, volume 7215 of *Lecture Notes in Computer Science*, pages 109–128, Tallinn, Estonia, Mar. 2012. Springer.

[25] S. Delaune, S. Kremer, and M. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[26] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM review*, 45(4):727–784, 2003.

[27] D. Fiore and A. Nitulescu. On the (In)Security of SNARKs in the Presence of Oracles. In *TCC'2016: 14th International Conference on the Theory of Cryptography*, volume 9985 of *LNCS*, pages 108–138, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[28] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *J. Cryptology*, 26(1):80–101, 2013.

[29] R. Küsters, T. Truderung, and A. Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *33rd IEEE Symposium on Security and Privacy (S&P 2012)*, pages 395–409. IEEE Computer Society, 2012.

[30] A. Petcher and G. Morrisett. The foundational cryptography framework. In *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, pages 53–72, 2015.

[31] V. Shoup. A proposal for an ISO standard for public key encryption. *IACR Cryptology ePrint Archive*, 2001:112, 2001.

[32] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman. Security analysis of the estonian internet voting system. In *21st ACM Conference on Computer and Communications Security (CCS'14)*, Scottsdale, AZ, USA, 2014.

[33] S. Wolchok, E. Wustrow, D. Isabel, and A. Halderman. Attacking the Washington, D.C. Internet Voting System. In *Financial Cryptography 2012*, pages 114–128, 2012.

Appendix

To prove verifiability of Belenios, we introduce intermediary security definitions as in [20], namely tally uniqueness and accuracy.

TALLY UNIQUENESS. Intuitively, this property ensures that no two different results can be simultaneously accepted by the verification algorithm in the context of the same election. The adversary has to produce a ballot box, and election key and two different results ($r \neq r'$) with two proofs. Formally, we define the advantage of the adversary $\mathcal{B}$ for any voting scheme $\mathcal{V}$ in the experiment $\mathsf{Exp}^{\mathsf{tuniq}}_{\mathcal{B},\mathcal{V}}$ as :

$$\Pr\left[\mathsf{Exp}^{\mathsf{tuniq}}_{\mathcal{B},\mathcal{V}}() = 1\right].$$

$\mathsf{Exp}^{\mathsf{tuniq}}_{\mathcal{B},\mathcal{V}}()$

1 : $(\mathsf{BB}, \mathsf{pk}, r, \pi, r', \pi') \leftarrow \mathcal{B}$
2 : $ev_1 \leftarrow \mathsf{Verify}((\mathsf{pk}, \mathsf{Publish}(\mathsf{BB}, \mathsf{pk}), r), \pi)$
3 : $ev_2 \leftarrow \mathsf{Verify}((\mathsf{pk}, \mathsf{Publish}(\mathsf{BB}, \mathsf{pk}), r'), \pi')$
4 : **return** $ev_1 \wedge ev_2 \wedge (r \neq r')$

This definition imposes a determinism over the output of Tally, as the result has to be unique for a given ballot box. In the case of mixnets, the counting function multiset must preserve this determinism. A simple instantiation to lexicographic order maintains this property.

ACCURACY. Intuitively, any voting protocol should ensure that the result of an election (computed by calling the Tally algorithm) can be successfully verified. Formally, a voting scheme $\mathcal{V}$ is *accurate* if for any efficient adversary $\mathcal{B}$ and for any set of voters I, the following probability is negligible in λ:

$$\Pr\left[\mathsf{Exp}^{acc}_{\mathcal{B},\mathcal{V},I}(\lambda) = 0\right].$$

$\mathsf{Exp}^{acc}_{\mathcal{B},\mathcal{V},I}(\lambda)$

1 : $\mathsf{pk}, \mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $\forall id \in I$ **do** $\mathsf{uL}[id] \leftarrow \mathsf{Register}(id)$
3 : $\mathsf{BB} \leftarrow \mathcal{B}(\mathsf{sk}, \mathsf{uL})$
4 : $(r, \pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$
5 : **return** $\mathsf{Verify}((\mathsf{pk}, \mathsf{Publish}\ \mathsf{BB}, r), \pi)$

We prove (in EasyCrypt) that Belenios satisfies tally uniqueness and accuracy.

Lemma 1. *Let* Policy *be an abstract algorithm, then* Belenios (Policy) *is tally unique and accurate.*

Proof.

Tally uniqueness. To ensure that no two different results can be verified successfully by the verifier of the $\Sigma_{\mathcal{R}}$ proof system, it is sufficient for $\Sigma_{\mathcal{R}}$ to be a proof of knowledge for a relation that implies the existence of a unique secret key matching the given public key and such that the given result corresponds to tallying the ballots obtained by decrypting the given ballot box.

Accuracy. The probability of verifying the proof output by Tally using the Verify algorithm is exactly that of verifying a valid proof–the completeness of the proof system. Note that the relation $\mathcal{R}$ checks if the result corresponds to the decryption of the board. □