# Machine-Checked Proofs of Privacy Against Malicious Boards for Selene & Co

Constantin Cătălin Drăgan
*Surrey Centre for Cyber Security*
*University of Surrey*
Guildford, United Kingdom
c.dragan@surrey.ac.uk

François Dupressoir
*Department of Computer Science*
*University of Bristol*
Bristol, United Kingdom
f.dupressoir@bristol.ac.uk

Ehsan Estaji
*Department of Computer Science & SnT*
*University of Luxembourg*
Esch-sur-Alzette, Luxembourg
ehsan.estaji@uni.lu

Kristian Gjøsteen
*Department of Mathematical Sciences*
*NTNU*
Trondheim, Norway
kristian.gjosteen@ntnu.no

Thomas Haines
*School of Computing*
*Australian National University*
Canberra, Australia
thomas.haines@anu.edu.au

Peter Y. A. Ryan
*Department of Computer Science & SnT*
*University of Luxembourg*
Esch-sur-Alzette, Luxembourg
0000-0002-1677-9034

Peter B. Rønne
*LORIA, CNRS & Univ Lorraine, France*
*University of Luxembourg, Esch-sur-Alzette,*
Luxembourg
0000-0002-2785-8301

Morten Rotvold Solberg
*Department of Mathematical Sciences*
*NTNU*
Trondheim, Norway
mosolb@ntnu.no

2022 IEEE 35th Computer Security Foundations Symposium (CSF) | 978-1-6654-8417-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSF54842.2022.9919663

***Abstract*—Privacy is a notoriously difficult property to achieve in complicated systems and especially in electronic voting schemes. Moreover, electronic voting schemes is a class of systems that require very high assurance. The literature contains a number of ballot privacy definitions along with security proofs for common systems. Some machine-checked security proofs have also appeared. We define a new ballot privacy notion that captures a larger class of voting schemes. This notion improves on the state of the art by taking into account that verification in many schemes will happen or must happen after the tally has been published, not before as in previous definitions.**

**As a case study we give a machine-checked proof of privacy for Selene, which is a remote electronic voting scheme which offers an attractive mix of security properties and usability. Prior to our work, the computational privacy of Selene has never been formally verified. Finally, we also prove that MiniVoting and Belenios satisfies our definition.**

## I. Introduction

Confidence in the validity of the outcome and privacy of the votes is supremely important for elections. We build confidence in elections by using carefully selected methods, routines and election officers. In particular, extensive use of various forms of auditing helps build confidence.

For the analysis of a voting mechanism, we need to know what security means and why the mechanism is secure. The former requires a so-called *security notion*, while the latter is best achieved with a *security proof*, a mathematical argument for why the mechanism satisfies the security notion.

It is easy to have an intuitive notion of what security should mean, but defining privacy for voting mechanisms is non-trivial, as shown by the many attempts to do so in the literature. (Bernhard *et al.* [5] has a good overview of privacy definitions.)

One problem is that many security notions are highly specialised for a particular class of voting mechanisms. But there are a large number of cryptographic voting mechanisms and they exhibit great variety in their form and shape. Defining security notions that usefully and uniformly capture a larger class of voting mechanisms is a good thing in principle, but it is also an essential task if existing security notions do not cover the voting mechanism of interest. ***We need security notions that capture a larger class of voting mechanisms.***

Once we have a security notion, we return to the problem of creating and auditing a security proof. *Machine-checked* proofs is a good way to increase assurance for security proofs. One system designed for handling security proofs is EasyCrypt. A large set of security notions, cryptographic constructions and corresponding security proofs have been written in EasyCrypt, and the system has seen extensive use. ***We need machine-checked security proofs for voting mechanisms.***

Selene [19] is a voting mechanism designed to provide a simple method for verification, while at the same time mitigating the threat of coercion. The key idea in Selene is that every voter is assigned a personal tracking number, and when the election period is over and everyone has cast their vote, the tracking numbers and the votes are published in plaintext on a web bulletin board. This gives the voters a direct and easy to understand way of verifying that their vote was correctly included in the tally. The key innovation in Selene is how to


DOI 10.1109/CSF54842.2022.00021

give the voter a tracking number so that no single party know what tracking number was given to the voter (other than the voter themselves) and the voter can plausibly lie about which tracker they received. The first property is achieved by mixing the trackers as part of the setup.

There is a danger of coercion here, namely that a coercer requires a voter to hand over her tracking number, so that the coercer himself can verify that the voter fulfilled his demands. However, the coercer has a limited window of opportunity, because he needs the coerced voter to hand over her tracker before the trackers and the votes are published. Otherwise, the coerced voter could simply find a vote corresponding to the coercer's demand, and give the coercer the tracker next to this vote. This observation is used in Selene to counter the threat of coercion: the voters first learn their tracking numbers after the trackers and votes are published on the web bulletin board.

Abstractly, Selene has a different order of operations than many existing voting mechanisms. Schemes like MiniVoting (which in some sense models a large class of voting mechanisms including variants of Helios) do voter verification before tallying. For Selene voter verification must happen after tallying, since the personal tracking numbers do not appear until after tallying. This means that Selene does not fit very well into existing security notions for voting mechanisms. This is also true for a number of other systems where voters or their delegates first can (or choose to) verify after tally. The Selene verification mechanism has also been trialled with a commercial partner [20]. ***We need a high-assurance security proof for Selene.***

### A. Our Contribution

In this paper, we define the new security notion *delay-use malicious-ballotbox ballot privacy* (du–mb–BPRIV) to capture the security of schemes that delay the use of verification information to a post-tallying verification step. This is necessary for tracker based schemes (like Selene [19], Electryo [18] and sElect [17]), for in-person voting schemes where the verification is first done later at home, but further it also applies to e-voting schemes where the verification step is not made mandatory before tallying, or often happens after tally, e.g. when verification is delegatable. To construct our definition, we build upon a recent ballot privacy definition called mb–BPRIV [12].

We model our new security notion in the proof assistant EASYCRYPT [1] (https://easycrypt.info), and to validate our security notion, we also model the Labelled-MiniVoting scheme [9] and Belenios [11] and verify that these schemes satisfy ballot privacy both under our new definition and under the original mb–BPRIV. Furthermore, we model the Selene voting system, and prove that this scheme satisfies ballot privacy under our new security definition. The EASYCRYPT code is available at https://github.com/mortensol/du-mb-bpriv.

### B. Related Work

Many authors have tried to capture the notion of ballot privacy using standard cryptographic games. Bernhard *et al.* [5] gives a good general overview of such notions. We give an overview of the history leading up to the recent definition of mb–BPRIV in Section II-G, directly preceding our new security notion.

The need for assurance with respect to voting systems makes cryptographic voting schemes a natural target for formalized security proofs, either through symbolic models and automatic verification or via proof assistants. While symbolic models have historically yielded good insights into the analysis of cryptographic protocols, see e.g. [8], [23] for symbolic analysis of Selene, we prefer a cryptographic analysis.

EASYCRYPT [1] is a proof assistant focused on formalizing computational security proofs in the style of Shoup's *Sequences of Games* [22]. EASYCRYPT supports constructive proofs of concrete security—leaving the complexity analysis of the constructed reduction to be done by hand. For simplicity in the rest of this paper, we discuss asymptotic notions. The formalized proof is concrete.

Cortier *et al.* use EASYCRYPT to prove that Helios is BPRIV-secure [9], and that Belenios is BPRIV-secure and verifiable [10]. Our proof builds upon their framework—we in fact prove that Labelled MiniVoting and Belenios meet our new privacy definition, and further formalize their security in mb–BPRIV.

## II. BACKGROUND

In this section, we first introduce some basic cryptographic models, primitives and algorithms that make up a voting system, before we move on to describe earlier definitions of ballot privacy.

### A. Random Oracle Model

In our analysis, we model hash functions as random oracles [2]. That is, to compute the value of a hash function at a point $x$, any party can make a call to an oracle $\mathcal{O}$, implementing a random function from some domain $D$ to some range $R$. The oracle $\mathcal{O}$ maintains an initially empty table $T$, and whenever someone calls $\mathcal{O}(x)$ for some $x \in D$, the oracle $\mathcal{O}$ checks if there is an entry $(x, y)$ in $T$ for some $y \in R$. If so, it returns $y$; if not, $\mathcal{O}$ randomly generates a $y' \in R$, adds $(x, y')$ to $T$ and outputs $y'$. We will use the Random oracle model implicitly below when modelling the non-interactive zero-knowledge proofs that help ensure privacy in e-voting.

### B. Public Key Encryption

Public key encryption systems are often used in voting protocols, to help protect the privacy of the votes, and possibly other things. In Selene, for instance, both the votes and the voter's personal tracking numbers are encrypted using some form of public key encryption. Formally, a *public key encryption system* (PKE) is defined as follows:

*Definition 1:* A public key encryption scheme (PKE) is a triple of algorithms $\mathsf{E} = (\mathsf{kgen}, \mathsf{enc}, \mathsf{dec})$; where

$\mathsf{kgen}$ is a probabilistic algorithm that takes as input a security parameter $\lambda$ and outputs a key pair $(\mathsf{pk}, \mathsf{sk})$,

enc is a probabilistic algorithm that on input a public key pk and a plaintext $m$ outputs a ciphertext $c$,

dec is a deterministic algorithm that on input a secret key sk and a ciphertext $c$ outputs either a plaintext $m$ or a special error symbol $\perp$ indicating that something went wrong.

We require that decryption "undoes" encryption, i.e. for any key pair $(\mathsf{pk}, \mathsf{sk})$ output by kgen, and any plaintext $m$, we have that $\mathsf{dec}(\mathsf{sk}, \mathsf{enc}(\mathsf{pk}, m)) = m$.

A *labelled public key encryption scheme (LPKE)* extends the notion of a "regular" PKE by adding some additional, non-malleable data called a *label* [21]. One important property for a labelled PKE is that decrypting a ciphertext using a different label than the one used for encryption, should not reveal anything about the original plaintext. Formally, a labelled PKE is defined similarly to how we define a PKE in Definition 1, but a label $\ell$ is given as additional input in the encryption and decryption algorithms.

The security of the schemes we analyze rely on a security notion for labelled public key encryption called *indistinguishability under chosen ciphertext attack with one parallel decryption query* (IND–1–CCA) [3]. Informally, this notion captures that any efficient adversary is unable to distinguish between encryptions of two messages of the same length, when given access to a batch decryption oracle that can be called once.

A similar security notion, namely poly–IND–1–CCA, allows the adversary to make up to $n$ challenge queries, for some polynomially bounded integer $n$. This notion is formalized in Figure 1, where an adversary $\mathcal{B}$ is given access to an encryption oracle $\mathcal{O}_{\mathsf{enc}}$ and a decryption oracle $\mathcal{O}_{\mathsf{dec}}$. The adversary can make $n$ challenge queries to $\mathcal{O}_{\mathsf{enc}}$, who encrypts one of two plaintexts, depending on the bit $\beta$. The adversary can query $\mathcal{O}_{\mathsf{dec}}$ at most once, and the oracle then decrypts a list of ciphertexts. For any ciphertexts created by the encryption oracle, the decryption oracle returns $\perp$.

The advantage of a poly–IND–1–CCA adversary $\mathcal{B}$ against a labelled public key encryption scheme E is defined as

$$\mathsf{Adv}^{\mathsf{poly\text{–}ind1cca}}_{\mathcal{B},\mathsf{E},n}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{poly\text{–}ind1cca},0}_{\mathcal{B},\mathsf{E},n}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{poly\text{–}ind1cca},1}_{\mathcal{B},\mathsf{E},n}(\lambda) = 1\right]\right|,$$

and we say that the labelled PKE E is $n$-challenge poly–IND–1–CCA-secure if the advantage defined above is negligible in $\lambda$ for all efficient adversaries $\mathcal{B}$.

As noted in [9], if $n = 1$, poly–IND–1–CCA security is essentially reduced to IND–1–CCA security. Indeed, it is possible to prove, through a hybrid argument, that a labelled PKE is IND–1–CCA secure if and only if it is poly–IND–1–CCA secure. This fact was also verified in EASYCRYPT [9], and we were able to reuse this framework in our formalization of the ballot privacy of Selene.

### C. Commitment Protocols

A *commitment protocol* allows a prover P to commit to some value $b$, and send the commitment to a verifier V. The verifier can ask the prover to open the commitment at some later point and verify the output value. Two important properties of a commitment protocol is that it should be *binding* and *hiding*. The first property informally means that once P has committed to a value, he should not be able to open the commitment to another value. The second property informally means that before the commitment is opened, V should not be able to determine what was committed to.

More formally, a commitment protocol is defined as follows:

*Definition 2 (Commitment protocol):* A *commitment protocol* is a triple of algorithms $\mathsf{CP} = (\mathsf{gen}, \mathsf{commit}, \mathsf{open})$, where

gen takes as input a security parameter $\lambda$ and returns a pair $(upk, usk)$ of user public and secret keys,

commit takes as input a user public key $upk$ and a value we want to commit to, and returns a commitment $ct$ and an opening key $d$,

open takes as input a commitment and an opening key and returns the committed value.

Commitments are an important part of the coercion mitigation strategy in Selene, where the election officials make commitments to personal tracking numbers for each voter. These commitments are opened by the voters at the end of the election, allowing the voters to verify that their vote was included in the tally, without being able to hand over their tracker to a coercer before all votes and trackers are published. For coercion-resistance, Selene actually employs trapdoor commitments and the voters have secret trapdoor keys that allow them to open the commitment to a tracker satisfying the coercer.

### D. Proof Systems

We now describe proof systems which are used in Selene to ensure that various operations are performed correctly. We say that a binary relation $\mathcal{R}$ is a subset $\mathcal{R} \subseteq X \times W$, where $X$ is a set of *statements* and $W$ is a set of *witnesses*. A *proof system* for the relation $\mathcal{R}$ is a pair of efficient algorithms $(\mathsf{P}, \mathsf{V})$, where P is called the *prover* and V is called the *verifier*. The prover and verifier work on a common input $x \in X$, and the prover has some additional input $w \in W$. In a *non-interactive proof system*, P uses his input to compute a proof $\Pi$. He sends the proof to V, who, on input $(x, \Pi)$ produces a verification output in $\{0, 1\}$.

A proof system is said to be *complete* if the prover can produce a valid proof whenever the statement is true. More formally, for any $(x, w) \in \mathcal{R}$, if $\Pi$ is a proof output by $\mathsf{P}(x, w)$, then $\mathsf{V}(x, \Pi)$ outputs 1 with probability 1.

A proof system is *sound* if a prover is unable to convince a verifier that a false statement is true.

A proof system is *zero-knowledge* if the proof leaks no information beyond the fact that the relation holds. More formally, we demand the existence of an efficient algorithm Sim, called the *simulator*, that produces valid-looking proofs for a statement $x \in X$ without access to the witness $w$. Formally, we consider a zero-knowledge adversary $\mathcal{B}$ in the following experiments:

$\mathsf{Exp}^{\mathsf{poly\text{–}ind1cca},\beta}_{\mathcal{B},\mathsf{E},n}(\lambda)$

1 : $\mathsf{encL} \leftarrow [\,]$
2 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{kgen}(\lambda)$
3 : $\beta' \leftarrow \mathcal{B}^{\mathcal{O}_{\mathsf{enc}}, \mathcal{O}_{\mathsf{dec}}}(\mathsf{pk})$
4 : **return** $\beta'$

$\mathcal{O}_{\mathsf{enc}}(\ell, m_0, m_1)$

1 : $c \leftarrow \bot$
2 : **if** $|\mathsf{encL}| < n$ **then**
3 : $\quad c \leftarrow \mathsf{enc}(\mathsf{pk}, \ell, m_\beta)$
4 : $\quad \mathsf{encL} \leftarrow \mathsf{encL} + [(c, \ell)]$
5 : **return** $c$

$\mathcal{O}_{\mathsf{dec}}(cL)$

1 : $mL \leftarrow [\,]$
2 : **for** $(c, \ell) \in cL$ **do**
3 : $\quad$ **if** $(c, \ell) \notin \mathsf{encL}$ **then**
4 : $\quad\quad mL \leftarrow mL + [\mathsf{dec}(\mathsf{sk}, \ell, c)]$
5 : $\quad$ **else** $mL \leftarrow mL + [\bot]$
6 : **return** $mL$

Fig. 1. Security experiment for poly–IND–1–CCA [9]

$\mathsf{Exp}^{zk,0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda)$

1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(\lambda)$
2 : $\Pi \leftarrow \bot$
3 : **if** $(\mathcal{R}(x, w))$ **then**
4 : $\quad \Pi \leftarrow \mathsf{P}(x, w)$
5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \Pi)$
6 : **return** $\beta'$

$\mathsf{Exp}^{zk,1}_{\mathcal{B},\mathsf{Sim},\mathcal{R}}(\lambda)$

1 : $(x, w, \mathsf{state}) \leftarrow \mathcal{B}(\lambda)$
2 : $\Pi \leftarrow \bot$
3 : **if** $(\mathcal{R}(x, w))$ **then**
4 : $\quad \Pi \leftarrow \mathsf{Sim}(x)$
5 : $\beta' \leftarrow \mathcal{B}(\mathsf{state}, \Pi)$
6 : **return** $\beta'$

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

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

and we say that a proof system is zero-knowledge if, for any adversary $\mathcal{B}$, there exists a simulator $\mathsf{Sim}$ such that the advantage defined above is negligible.

### E. Voting Systems

We define a *voting system* as being built upon a tuple of algorithms

$$\mathcal{V} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{ValidBoard}, \mathsf{Tally}, \mathsf{VerifyVote}, \mathsf{VerifyTally}, \mathsf{Publish}),$$

where the different algorithms informally work as follows:

- $\mathsf{Setup}(1^\lambda)$: Returns a pair $(\mathsf{pd}, \mathsf{sd})$ of public and secret data, typically including a public encryption key and a secret decryption key, respectively, but this data might also contain other things.
- $\mathsf{Register}(id, \mathsf{pd}, \mathsf{sd})$: Takes as input a user identity and some public and secret data and returns a public credential $pc$ and a secret credential $sc$ for that user.
- $\mathsf{Vote}(\mathsf{pd}, pc, sc, v)$: Takes as input some public data, a user's public and secret credentials, and a vote, and returns the user's public credential, a ciphertext encrypting the vote and a state that the voter later can use for verification.
- $\mathsf{ValidBoard}(\mathsf{BB}, \mathsf{pd})$: Checks the validity of the ballot box BB.
- $\mathsf{Tally}(\mathsf{BB}, \mathsf{pd}, \mathsf{sd})$: Computes the result $r$ of the election, along with a proof $\Pi$ of correct tallying.
- $\mathsf{VerifyVote}(id, \mathsf{state}, \mathsf{BB}, pc, sc)$: Run by a voter to check whether or not her ballot was included in the tally.
- $\mathsf{VerifyTally}((\mathsf{pd}, pbb, r), \Pi)$: Checks that $\Pi$ is a valid proof of correct tally, with respect to the result $r$ and the public part $pbb$ of the ballot box BB.
- $\mathsf{Publish}(\mathsf{BB})$: Returns a public part $pbb$ of the ballot box BB.

### F. Voting Friendly Relations

In the voting systems we analyze in this paper, proof systems are used to compute and validate proofs of correct tally. In our analysis and EASYCRYPT formalization, we keep the relation $\mathcal{R}$ abstract, and thus, we need to ensure that the relation is compatible with the result of the election. For this, we adopt the notion of *voting friendly relations*, as defined by Cortier *et al.* [9] and generalize it a bit, so that it also accommodates schemes like Selene. Very informally, a relationship is voting friendly if for any adversarially chosen bulletin board it is possible to find a corresponding tally such that the pair (bulletin board and tally) are in the language.

The relation being compatible with the result of the election means that if $\mathcal{V}$ is a voting system, $(\mathsf{pd}, \mathsf{sd})$ is the public and secret data generated by the $\mathsf{Setup}$ algorithm, the result $r$ of the election corresponds to the tally of the votes obtained by decrypting the ciphertexts in the ballot box BB, and if $pbb$ is the public part of BB, then the relation $\mathcal{R}((\mathsf{pk}, pbb, r), (\mathsf{sk}, \mathsf{BB}))$ holds. In other words, it is possible to prove that $r$ is the correct result. More formally, a voting friendly relation is defined as follows:

*Definition 3 (Voting friendly relation [9]):* Let $\mathcal{V}$ be a voting system and let $\Sigma_\mathcal{R}$ be a proof system for some relation $\mathcal{R}$. We say that $\mathcal{R}$ is a voting friendly relation if, for any efficient adversary $\mathcal{B}$, the following experiment returns 1 with negligible probability:

$\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{B},\mathcal{V},\Sigma_\mathcal{R}}(\lambda)$

1 : $(\mathsf{pd}, \mathsf{sd}) \leftarrow \mathcal{V}.\mathsf{Setup}$
2 : $\mathsf{BB} \leftarrow \mathcal{B}(\mathsf{pd})$
3 : $dbb \leftarrow \mathsf{dec}^*(\mathsf{sd}, \mathsf{BB})$
4 : $r \leftarrow \rho(dbb)$
5 : $pbb \leftarrow \mathcal{V}.\mathsf{Publish}(\mathsf{BB})$
6 : **return** $\neg\mathcal{R}((\mathsf{pd}, pbb, r), (\mathsf{sd}, \mathsf{BB}))$

In the above experiment, the algorithm $\mathsf{dec}^*$ decrypts the ballot box BB line by line using the secret data sd, and returns a list $[(a_1, v_1), \ldots, (a_n, v_n)]$ of votes $v_i$ and some additional information $a_i$, e.g. voter identities as in MiniVoting or tracking numbers as in Selene. We say that $\rho$ is a counting function that takes in a list of the form described above, and returns a result.

### G. Early Definitions of Ballot Privacy

In 2015, Bernhard *et al.* [5] conducted a survey of existing game-based ballot privacy definitions and found that they were all unsatisfactory. Some of the definitions were too weak, declaring protocols that intuitively did not have ballot privacy to be secure. Some definitions were too strong, making any voting protocol with even minimal verifiability impossible to prove private. Finally, some definitions were too limited, restricting the class of captured voting protocols and privacy breaches too much.

Based on this survey, Bernhard *et al.* [5] proposed a new definition of ballot privacy which was named BPRIV. The BPRIV definition captures the idea that a voting system should not leak any information about the votes that are cast, beyond what can be derived from the result of the election. This is formalized by having an adversary attempting to distinguish between two worlds. In one world, the adversary gets to see a ballot box containing real ballots submitted by honest voters, as well as any ballots the adversary has submitted on behalf of dishonest voters. The adversary then gets to see the result corresponding to these ballots and a proof of correct tally. In the other world, the adversary gets to see a fake ballot box, but he still gets to see the result as tallied on the real ballot box. It is also assumed that there exists a simulator that can simulate a proof of correct tally corresponding to the real result, but with respect to the fake ballot box. The adversary also gets to see this simulated proof.

As the name suggests, ballot privacy BPRIV only captures the privacy of the ballots and not of the tally. To account for this Bernhard *et al.* say that any scheme satisfying BPRIV should also satisfy a property called strong consistency which captures the idea that the tally produced by the scheme should not leak more information than an idealised tally function. The exact idealised tally function is a parameter of the definition. We have proved the strong consistency of Selene as part of our work.

To avoid having to trust the voting server, the strategy in many voting systems is to encrypt the votes under a key for which the corresponding decryption key is split into several parts and distributed among several authorities. However, as is pointed out in [12], this trust assumption is not properly captured in the BPRIV definition. In BPRIV, the adversary plays a game where he can control the votes cast by honest parties, but he cannot control the resulting ballots once they are put in the ballot box. This means that BPRIV assumes that every ballot that is put in the ballot box stays in the ballot box and is not tampered with in any way.

To address this, Cortier *et al.* introduced a new definition, which they called mb–BPRIV [12]. The main idea in mb–BPRIV is similar to BPRIV: the adversary has to try and distinguish between two worlds: one where he sees real ballots and the real result, and one where he sees fake ballots, but still sees the real result. The difficulty is that an adversary who is in control of the ballot box is able to remove or tamper with any ballots submitted by honest voters. Since we perform the tally on the real ballots in both situations, we need to somehow determine which of the real ballots to perform the tally on. A bad choice would make distinguishing trivial for the adversary.

Cortier *et al.*'s solution is to parameterize their security definition by a *recovery algorithm*, an approach we also adopt in our definition. Informally, the idea is to use the recovery algorithm on the adversary's board in the fake world, to determine how the adversary has tampered with the ballots on the fake board. We then perform the same transformation on the real board, and tally the resulting board.

Formally, Cortier *et al.* define the aforementioned transformation as a *selection function*, and recovery algorithm as the process of finding the transformation.

*Definition 4 (Selection function [12]):* For integers $m, n \geq 1$, a selection function for $m$ and $n$ is any mapping

$$\pi : \{1, \ldots, n\} \to \{1, \ldots, m\} \cup (\{0,1\}^* \times \{0,1\}^*) .$$

The selection function $\pi$ represents the how the adversary constructs a bulletin board $\mathsf{BB}$ with $n$ ballots, given a bulletin board $\mathsf{BB}_1$ with $m$ ballots. For $i \in \{1, \ldots, n\}$,

- $\pi(i) = j$, with $j \in \{1, \ldots, m\}$ means that this is the $j$th element in $\mathsf{BB}_1$,
- $\pi(i) = (pc, c)$ means that this element is $(pc, c)$.

The function $\overline{\pi}$ associated to $\pi$ maps a bulletin board $\mathsf{BB}_0$ of length $m$ to a board $\overline{\pi}(\mathsf{BB}_0)$ of length $n$ such that for any $j \in \{1, \ldots, n\}$,

$$\overline{\pi}(\mathsf{BB}_0)[j] = \begin{cases} (pc, c) \text{ if } \pi(j) = i \text{ and } \mathsf{BB}_0[i] = (id, (pc, c)) \\ (pc, c) \text{ if } \pi(j) = (pc, c) \end{cases}$$

*Definition 5 (Recovery algorithm [12]):* A recovery algorithm is any algorithm Recover that takes as input two bulletin boards $\mathsf{BB}$ and $\mathsf{BB}_1$ and returns a selection function $\pi$ for $|\mathsf{BB}_1|$ and $n$ for some integer $n$.

We will sometimes abuse notation and write $\mathsf{BB}' \leftarrow \mathsf{Recover}(\mathsf{BB}, \mathsf{BB}_0, \mathsf{BB}_1)$ to denote the process of determining the transformation from $\mathsf{BB}_1$ to $\mathsf{BB}$, and applying this transformation to $\mathsf{BB}_0$, to get the board $\mathsf{BB}'$.

In mb–BPRIV, the tally occurs only if the adversary's board is valid, and if none of the voters are unhappy after they perform some kind of verification. This means that the verification process needs to occur before the tally, so mb–BPRIV does not accommodate voting systems like Selene. Indeed, in Selene, the tally occurs before the verification as a way of mitigating the threat of coercion. Therefore, there is still need for a new privacy definition, that both allows for the voting server (and thus the ballot box) to be dishonest, and that accommodates voting protocols where the verification phase happens only after the tally has been computed.

## III. New Security Notion

In this section, we present a new definition of ballot privacy against a malicious ballot box, which we call *delay-use malicious ballotbox ballot privacy* (du–mb–BPRIV). This definition essentially extends the range of applicable voting schemes to include those where the verification can be *delayed*,

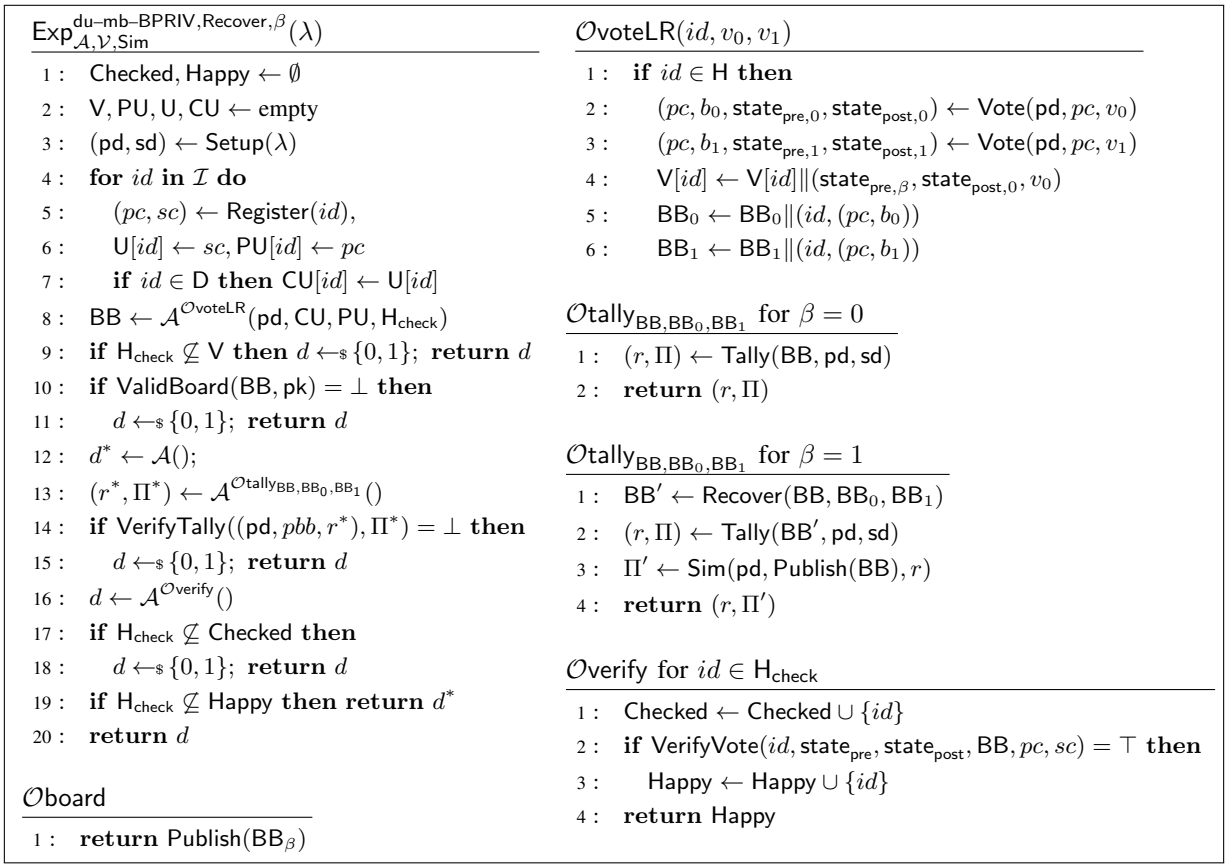


Fig. 2. The new security notion for ballot privacy against a dishonest ballot box.

i.e. happening after tallying, and also includes schemes where a secret key is needed in the verification step. Our new definition is similar to mb–BPRIV, the most notable difference being that in our definition, the adversary gets to see the tally after we check if his board is valid, and then he gets to see the result of the verification phase. A formal description of the security game for du–mb–BPRIV is found in Figure 2. The relation between du–mb–BPRIV and mb–BPRIV is studied in more detail in Section III-B.

In the following, let $\mathcal{I} = \mathsf{H} \cup \mathsf{D}$ be a set of voter identities, partitioned into a set $\mathsf{H}$ of honest voters and a set $\mathsf{D}$ of dishonest voters. Furthermore, let $\mathsf{H}$ be partitioned into the set $\mathsf{H}_{\mathsf{check}}$ of voters who we assume will perform some verification check and the set $\mathsf{H}_{\overline{\mathsf{check}}}$ of voters who we assume will not verify.

The security experiment begins with the generation of some public and secret data $\mathsf{pd}$ and $\mathsf{sd}$ (which typically includes the public and secret keys used to encrypt and decrypt the votes). Then, a number of voters in a set $\mathcal{I}$ are registered. In the registration phase, public and secret credentials $pc$ and $sc$ are generated for each voter, and stored in finite maps $\mathsf{PU}$ and $\mathsf{U}$, respectively. Furthermore, we store the secret credentials of a set $\mathsf{D}$ of dishonest voters in a finite map $\mathsf{CU}$.

The adversary is now given $\mathsf{pd}, \mathsf{PU}$ and $\mathsf{CU}$ as input. In addition, he gets access to a vote oracle, that on input $(id, v_0, v_1)$ computes two ballots for this user. The first ballot is stored in a list $\mathsf{BB}_0$ and the second ballot is stored in a list $\mathsf{BB}_1$. The vote oracle also records a state, containing any information the voter later needs to verify that her vote was correctly cast and counted; we split the state into two components. The first component ($\mathsf{state}_{\mathsf{pre}}$) covers information checked which is generated before tallying and the second component ($\mathsf{state}_{\mathsf{post}}$) covers information generated after tallying; this is necessary due to recovery which ensures that information after tallying (including the tally) always acts as if $\beta = 0$. The adversary may also call on a publish oracle, allowing him to see, essentially, $\mathsf{BB}_\beta$ for a secret bit $\beta$.

Using this information, the adversary creates a public bulletin board $\mathsf{BB}$. If the board is invalid, we output a random bit. If the board is valid, we allow the adversary to make an initial guess at the secret bit $\beta$, based on the information he has seen so far. This guess is stored in a variable $d^*$, possibly to be returned by the experiment at a later point.

The adversary now gets access to the tally, and is allowed to add some extra information to the bulletin board, namely a result $r^*$ and a proof $\Pi^*$ that this result corresponds to the votes. If this result and proof fails to pass verification, we output a random bit.

Otherwise, the adversary gets to make a guess $d$, given access to a verification oracle $\mathcal{O}\mathsf{verify}$. The verification oracle records the users who have verified in a set called $\mathsf{Checked}$, and the users who are happy with the verification are recorded

in the set Happy. If anyone who we expect should verify actually does not verify, we output a random bit. If a voter is unhappy with the verification process, we output the initial guess $d^*$ the adversary made before seeing the tally. Otherwise, the experiment outputs the guess $d$ that the adversary made after calling the verify oracle.

When given access to the tally oracle, the adversary can call this oracle only once, and the behavior of the tally oracle depends on whether we are in the left world ($\beta = 0$) or in the right world ($\beta = 1$). If we are in the left world, the tally is performed directly on the board BB created by the adversary. The adversary then gets to see a real result and a proof of correct tally. In the right world, however, we first run the recovery algorithm to detect how the adversary has tampered with the ballots in $\mathsf{BB}_1$, to create BB. We then change the ballots on $\mathsf{BB}_0$ accordingly, yielding a new board $\mathsf{BB}'$, which we tally. The adversary then gets to see the result $r$ corresponding to $\mathsf{BB}'$ and a simulated proof $\Pi'$ of correct tally, with respect to $r$ and the adversarial board BB.

*Definition 6:* Let $\mathcal{V}$ be a voting system, and let Recover be a recovery algorithm. We say that $\mathcal{V}$ satisfies du–mb–BPRIV with respect to Recover if there exists an efficient simulator Sim, such that for any efficient adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) = \Big| \Pr\left[\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},0}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},1}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) = 1\right]\Big|,$$

is negligible in the security parameter $\lambda$, where $\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},\beta}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}$ is the game defined in Figure 2.

### A. Recovery function

The mb–BPRIV definition is well defined for many recovery functions, but three are suggested in the body of [12]. All three recovery functions when used with mb–BPRIV are only satisfied by schemes which prevent the adversary casting ballots on behalf of the honest voters. (This observation is not made in [12], but is implicit in the introduction of a fourth recovery function in Appendix J of [12], where Helios is analysed.) We introduce a simple new recovery function (Fig. 3) which we call $\mathsf{Recover}^{\mathsf{del,reorder'}}_{\mathsf{U}}$ which is essentially identical to $\mathsf{Recover}^{\mathsf{del,reorder}}_{\mathsf{U}}$ in [12], but does not require ballot authentication. mb–BPRIV with the recovery functions originally suggested implies the satisfying scheme is equivalent to some ideal functionalities. Future versions of du–mb–BPRIV may wish to prove similar results in which a different recovery function is likely necessary.

$\mathsf{Recover}^{\mathsf{del,reorder'}}_{\mathsf{U}}(\mathsf{BB}_1, \mathsf{BB})$

1 : $L \leftarrow [\,]$
2 : **for** $(pc, c) \in \mathsf{BB}$ :
3 : **if** $\exists j, id : \mathsf{BB}_1[j] = (id, (pc, c))$ :
4 : $L \leftarrow L \parallel j$ (if several such $j$ exist, pick the first one)
5 : **else** :
6 : $L \leftarrow L \parallel (pc, c)$
7 : **return** $(\lambda i.\ L[i])$

Fig. 3. The $\mathsf{Recover}^{\mathsf{del,reorder'}}_{\mathsf{U}}$ algorithm.

### B. Comparison of du–mb–BPRIV to mb–BPRIV

In this section we briefly analyse the differences between the mb–BPRIV definition and our new du–mb–BPRIV definition. As mentioned above, the main difference is that the verification oracle is first available after the tally oracle has been called. This accommodates schemes where the verification first happens after tally and allows a secret key to be used for the verification process, however, it naturally also applies to schemes with early verification. We have changed some parts of the definition to adapt to the delayed use of the verification, but also to make it optimised and precise enough for EASYCRYPT.

The main differences are:

- We only have one voter map V but the state stored depends on secret bit $\beta$, see line 4 in the definition of $\mathcal{O}$voteLR in Figure 2. However, the state is split into a part relevant before tally and a post-tally part (only relating to $\beta = 0$ which is the board used for tallying). This is necessary for the state handling in EASYCRYPT oracles, and was not necessary in mb–BPRIV since the stateful verification happened before tallying.
- If the adversary does not output a valid board, the experiment outputs a random guess bit, whereas mb–BPRIV allows the adversary to output a guess but without tally access. In both cases this corresponds to real life, where a board will not be tallied if it is not valid. Outputting a random bit makes our proofs in EASYCRYPT slightly easier, and it is actually equivalent to mb–BPRIV unless the ValidBoard algorithm always outputs $\perp$.
- In our definition the experiment also outputs a random guess bit if the verification of the tally fails via the algorithm VerifyTally. Again, this corresponds to not allowing any adversarial advantage when the tally fails publicly. This case was not explicitly considered in mb–BPRIV, but is natural in our case where the verification step will not proceed on an invalid tally output.
- In du–mb–BPRIV the verification status of the honest voters contained in Happy is directly output to the adversary. In mb–BPRIV this is not defined precisely. In both definitions the appearance of failed verifications inside the set of checking honest voters $\mathsf{H}_{\mathsf{check}}$ will in any case imply that the guess bit has to be determined without knowing the tally result. This is to punish the adversary for creating a board with verifications failing which would cause complaints in real life protocols.

Consider a voting scheme where the verification step does not depend on a secret key and can be done before the tally. For such schemes both privacy definitions can be applied to the scheme. We claim that under reasonable conditions our

definition is stronger. Further, for voting schemes where the outcome of the individual verifications can be computed by the adversary using the data from the bulletin board, as e.g. happens in Helios, we have equivalence of the two definitions. We now sketch why this is the case.

We show that du–mb–BPRIV privacy implies mb–BPRIV assuming that ValidBoard does not constantly output $\perp$, and that VerifyTally never fails on an honestly computed tally. Finally, we also assume that the verification status Happy is not output to the adversary in mb–BPRIV or that the verification status can be computed using public data, as e.g. happens in Helios. To prove the implication we assume that we have an attack algorithm for mb–BPRIV. We use the vote choices from the mb–BPRIV algorithm. If the attack algorithm outputs an invalid board, we change the board that is being output to a valid board which we have assumed exists. Since the tally also does not fail we are allowed to output a guess and we use the one from the attack algorithm and will win with the same advantage since in this case no verification will be done in mb–BPRIV. If the attack algorithm outputs a valid board, we use the same board in the du–mb–BPRIV experiment. In du–mb–BPRIV line 12 the adversary has to output a guess bit $d^*$ which will be used if a verification fails for the honest verifier set $\mathsf{H}_{\mathsf{check}}$. Here we use what will be output from the mb–BPRIV attack algorithm in case of failed verifications, which by assumption either does not depend on Happy, which we do not yet have access to at this stage in the du–mb–BPRIV experiment, or it can be computed from the public data on the board. In the experiment du–mb–BPRIV line 13 we now get the tally before verification (and the tally verifies by assumption) but we ignore this at first and choose the same verifying voters as in the attack algorithm. At this point the two experiments will be equivalent. If the verification fails we are forced to go back to $d^*$ but this was from the attack algorithm and will be equivalently successful. If no verification fails in $\mathsf{H}_{\mathsf{check}}$ then we output the guess from the attack algorithm using the tally result we got earlier as input. The advantage will be the same as for the mb–BPRIV attack algorithm. This was the important implication direction since it demonstrates that our definition is not too weak, i.e. if an early verification scheme is declared du–mb–BPRIV then it is also mb–BPRIV private which in turn has ideal functionality implications under assumptions such as strong consistency [12].

Considering whether mb–BPRIV privacy implies du–mb–BPRIV, the main problem is that the choice of verifying voters in du–mb–BPRIV could depend on the tally output. However, for voting schemes where the outcome of the honest voters' individual verification can be computed by the adversary, we can prove the implication. We thus assume we have an attack algorithm for du–mb–BPRIV with non-negligible advantage. In the experiment mb–BPRIV we use the votes and output the board from this attack algorithm. If the board is not valid, we simply output a random bit in the experiment mb–BPRIV, which is equivalent to what happens in du–mb–BPRIV. If the board is valid, the next step in mb–BPRIV is to use the verification oracle. Here we simply let all the required honest voters $\mathsf{H}_{\mathsf{check}}$ perform verifications. These will also all have to verify in the attack against du–mb–BPRIV to get an advantage since the experiment will otherwise output a random bit. If a verification fails we will use the output $d^*$ in the attack algorithm which was computed without tally access. If none of the verifications fail, we will get tally access in the experiment mb–BPRIV. If more voters were chosen to verify in the attack algorithm we can compute the outcomes by assumption. If we encounter a failure in the verification here, we again output $d^*$, otherwise the output from the attack algorithm. The experiments will be equivalent at this point.

## IV. Labelled-MiniVoting and Belenios

The MiniVoting scheme was first introduced by Bernhard *et al.* [6] as an abstraction meant to capture several existing constructions in the literature. It is based on two building blocks: public key encryption and a zero-knowledge proof system, and assumes the ballot has the form $(id, c)$, for a voter's identity $id$ and for a ciphertext $c$ - that simply encrypts the voter's choice (or vote) $v$.

This scheme was later refined by Cortier *et al.* [9], resulting in the *Labelled-MiniVoting* scheme. Here, the class of captured voting schemes was broadened by introducing some public information associated to the users, called *labels*, and creating a strong link between a voter identity in a particular election and its ciphertext - now parameterized by this label via the use of labelled public key encryption. The labels can be used to represent generic information about the election (as in the case of Helios) and/or information pertaining to a voter's public persona: pseudonym or public verification key (as in Belenios). The ballot in Labelled-MiniVoting takes the form $(id, \ell, c)$ with $id$ the voter's identity and $(\ell, c)$ the label-ciphertext pair created by the labelled public key encryption scheme.

A predominant feature of the MiniVoting class of schemes is the enforcement of unique label-ciphertext pairs, via a procedure called weeding [11]. This step, done by the ValidBoard algorithm, prevents trivial attacks on privacy, where an adversary can cast copied ciphertexts (and their label) and observe the changes in the election result.

In Figure 4 we refine Labelled-MiniVoting to align with the voting notations from previous sections, such that we treat the public credential $pc$ as the label and consider the following ballot format $(pc, c)$. The removal of $id$ doesn't have a significant impact, as we use the operation Flabel to model the link between identity and public credential:

$$pc \leftarrow \mathsf{Flabel}(id).$$

Typical instantiations for this operator depend on the assumptions over the voter's identity and the degree of separation we want to capture in the public credential. Here, we consider the voter's identity a pseudonym or a public encryption key and as such we implement Flabel as the identity function $\mathsf{Flabel}(x) = x$; an approach also taken by Selene. Other options may consider the real voter's identity (e.g. email, name) as input and create a pseudonym or a public credential

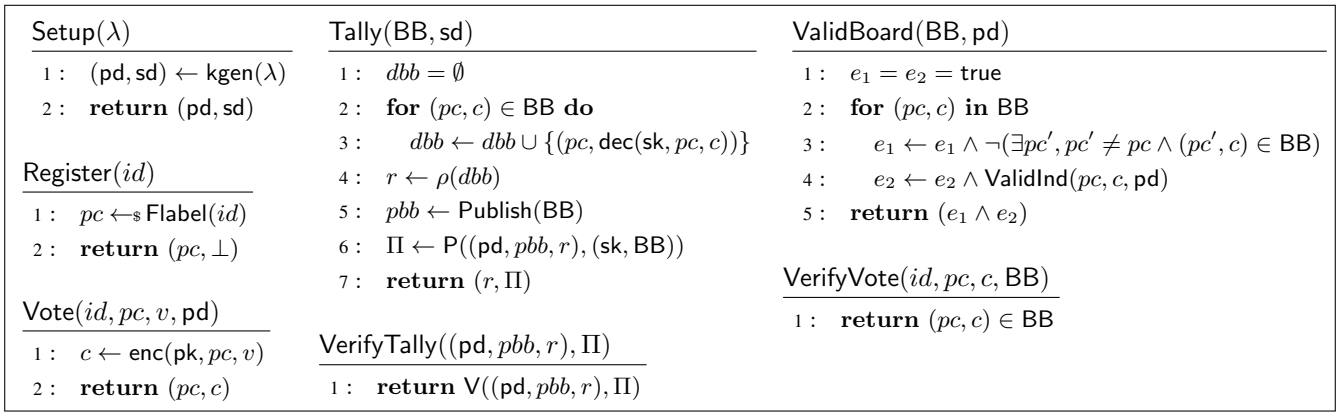


Fig. 4. Algorithms defining the Labelled-MiniVoting scheme for the labelled PKE $\mathsf{E} = (\mathsf{kgen}, \mathsf{enc}, \mathsf{dec})$, and the proof system $\Sigma = (\mathsf{P}, \mathsf{V})$.

(especially if signatures are considered). In all cases, there has to be an injectivity assumption over Flabel, which is trivially satisfied by the identity function.

Labelled-MiniVoting is further parameterized by three other classic operators:

$\mathsf{ValidInd}(pc, c, \mathsf{pd})$: $\{0,1\}$**.** Checks if the label-ciphertext pair $(pc, c)$ is well-formed.

$\rho((pc_i, v_i)_i)$**.** This function returns the election result in a predefined format, e.g. lexicographic order for mixnet tally or a value for homomorphic tally. This is done by first deciding which votes to keep using Policy, and then computing the result over the votes kept by Policy using Count.

- Policy is fixed to "last vote counts" for a particular voter, in the modelling of both Labelled-MiniVoting and Selene.
- Count is left abstract for Labelled-MiniVoting, and made concrete for Selene.

$\mathsf{Publish}(\mathsf{BB})$: $\{0,1\}^*$**.** This is an abstraction of the public bulletin board, and most of the times it is identical to the ballot box.

*Definition 7:* Let $\mathsf{E}$ be a poly–IND–1–CCA secure labelled PKE, and $\Sigma = (\mathsf{P}, \mathsf{V})$ be a zero-knowledge proof system. Given the operators $\mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish}$ defined as above, we define the Labelled-MiniVoting scheme

$$\mathsf{MiniVoting}(\mathsf{E}, \Sigma, \mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish}) = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Vote}, \mathsf{ValidBoard}, \mathsf{Tally}, \mathsf{VerifyVote}, \mathsf{VerifyTally}, \mathsf{Publish})$$

as the single-pass voting scheme whose algorithms are presented in Figure 4, and which we informally present below:

$\mathsf{Setup}(\lambda) : (\mathsf{pd}, \mathsf{sd})$. Given the security parameter $\lambda$ it returns the output $(\mathsf{pd}, \mathsf{sd})$ of the key generation algorithm for the encryption scheme.

$\mathsf{Register}(id) : (pc, sc)$. It applies Flabel over $id$ to build the public credential $pc$, and does not consider a secret credential, i.e. $sc \leftarrow \perp$.

$\mathsf{Vote}(id, pc, v, \mathsf{pd}) : (\mathsf{pd}, c)$. Encrypts a vote $v$ using the public credential $pc$ as the label, and outputs the ciphertext together with the voter's public credential. The secret credential $sc$ is omitted from the input just for simplicity as it is unused.

$\mathsf{ValidBoard}(\mathsf{BB}, \mathsf{pd}) : \{0,1\}$. Returns true if all ballots $(pc, c)$ are well-formed, according to ValidInd, and that each ciphertext is always used with the same public credential (weeding property is respected); and false otherwise.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sd}) : (r, \Pi)$. Computes the result $r$ of the election by applying the counting function $\rho$ over the list of $(pc, v_\perp)$, where $v_\perp$ is the decryption of $(pc, c)$ from the ballot box. It also provides a proof of correct decryption $\Pi$ using the P algorithm of the proof system $\Sigma$.

$\mathsf{VerifyVote}(id, pc, c, \mathsf{BB}) : \{0,1\}$. It checks if its last ballot $(pc, c)$ is in the ballot box BB. The voter may have a state with all cast ballots, but the verification is done with respect to the last vote that was cast.

$\mathsf{Verifytally}((\mathsf{pd}, pbb, r), \Pi) : \{0,1\}\}$. Run the V algorithm of the proof-system to check if the tally proof is valid w.r.t the given statement.

$\mathsf{Publish}(\mathsf{BB})$ : Calls the Publish operator over the ballot box BB.

We prove in EASYCRYPT that MiniVoting satisfies du–mb–BPRIV under standard cryptographic assumptions for the encryption scheme E and proof system $\Sigma$. We also consider the identity function for Flabel, and "last vote counts" for Policy. The following theorem corresponds to lemma `du_mb_bpriv` in the `MiniVotingSecurity_mb.ec` file.

*Theorem 1:* Let $\mathcal{V} = \mathsf{MiniVoting}(\mathsf{E}, \Sigma, \mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish})$ be defined as in Definition 7. Then, for any du–mb–BPRIV adversary $\mathcal{A}$, there exists a simulator Sim and three adversaries $\mathcal{B}, \mathcal{C}$ and $\mathcal{D}$, such that

$$\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) \leq 2 \cdot \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D},\mathcal{V},\Sigma}(\lambda) = 1\right] + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B},\mathsf{P},\mathsf{Sim},\mathcal{R}}(\lambda) + \mathsf{Adv}^{\mathsf{poly\text{–}ind1cca}}_{\mathcal{C},\mathsf{E},n}(\lambda).$$

### A. du–mb–BPRIV *for Belenios*

We have used Labelled-MiniVoting to validate our privacy definition, and to infer proof strategies and assumptions that then can be applied to other e-voting systems, e.g. Belenios [11] and Selene. Belenios can be viewed as an instance of

Labelled-MiniVoting with some concrete decisions for operators and algorithms. This translates into directly applying the EASYCRYPT proof developed for MiniVoting to Belenios without the need to re-do it - as highlighted in `Belenios.ec`.

We take the labelled encryption scheme $\mathsf{E}$ and realize it by combining the ElGamal encryption scheme $\mathsf{E_B}$ with a zero-knowledge proof system $\Sigma_\mathsf{B}$. As the public credential $pc$ is included in the statement for the proof system $\Sigma_\mathsf{B}$ we use $\pi[pc]$ to express this fact. Encrypting the vote $v$ under some public credential $pc$ by $\mathsf{E}$ becomes $(pc, c) = (pc, (\mathsf{c_B}, \pi[pc]))$, and decrypting by $\mathsf{E}$ returns the decryption of $\mathsf{c_B}$ by $\mathsf{E_B}$ only if the proof $\pi[pc]$ verifies, and $\perp$ otherwise.

This form of the ciphertext also has an impact on the $\mathsf{ValidInd}$ algorithm that now uses $\pi[pc]$ to decide if a ciphertext is well-formed.

For the result of the election, we only need to instantiate the $\mathsf{Count}$ operator, as we already consider the last vote policy. We can model any type of ideal counting function that can be performed over votes, and instantiate it to lexicographic order `lex-order` to model an ideal verifiable shuffle.

In Belenios, the public bulletin board computed by $\mathsf{Publish}$ shows only the last ballot cast by a voter (policy applied over the public credential) together with a hash of that ballot. With verification done against the hash compared to the entire ballot. However, nothing prevents voters from checking their full ballot against the ballot box (as we have modelled in Labelled-MiniVoting). Moreover Cortier *et al.* [10] have modelled in EASYCRYPT different options of $\mathsf{Publish}$ for Belenios and as one would expect the two approaches are equivalent modulo hash collisions.

*Definition 8:* Let $\mathsf{E} = (\mathsf{E_B}, \Sigma_\mathsf{B})$ be a $\mathsf{poly–IND–1–CCA}$ secure labelled PKE, and $\Sigma$ be a zero-knowledge proof system. Given the operators $\mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish}$ defined as above, we define $\mathsf{Belenios}(\mathsf{E_B}, \Sigma_\mathsf{B}, \Sigma)$ as

$$\mathsf{MiniVoting}(\mathsf{E}, \Sigma, \mathsf{Flabel}, \mathsf{ValidInd}, \rho, \mathsf{Publish}).$$

The privacy result for Belenios follows directly by simply applying Theorem 1 with the concrete values highlighted here.

*Theorem 2:* Let $\mathcal{V} = \mathsf{Belenios}(\mathsf{E_B}, \Sigma_\mathsf{B}, \Sigma)$ be defined as in Definition 8. Then, for any $\mathsf{du–mb–BPRIV}$ adversary $\mathcal{A}$, there exists a simulator $\mathsf{Sim}$ and three adversaries $\mathcal{B}, \mathcal{C}$ and $\mathcal{D}$, such that

$$\mathsf{Adv}^{\mathsf{du–mb–bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) \leq 2 \cdot \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D},\mathcal{V},\Sigma}(\lambda) = 1\right] + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B},\mathsf{P},\mathsf{Sim},\mathcal{R}}(\lambda) + \mathsf{Adv}^{\mathsf{poly–ind1cca}}_{\mathcal{C},\mathsf{E},n}(\lambda).$$

Both Theorem 1 and 2 are proven with respect to the recovery algorithm described in Section III-A.

### B. mb–BPRIV *for MiniVoting and Belenios*

We also stated the original mb–BPRIV definition [12] in EASYCRYPT and proved that Theorem 1 and 2 also hold with respect to this privacy definition. The proof strategy was essentially the same, but the proofs had to be re-worked due to the differences between the definitions, especially when the verification happens. This constitutes the first machine-checked proof of mb–BPRIV. The corresponding EASYCRYPT lemma `mb_bpriv` is found in the `MiniVotingSecurity_omb.ec` and `Belenios_omb.ec` files.

## V. SELENE

Although Selene offers properties such as verification and coercion mitigation, we focus—in this paper—on formalizing its ballot privacy properties. Like Cortier *et al.* [9], [10], we abstract away the verifiable shuffles (assuming they are non-interactive) and the ElGamal + PoK construction (assuming instead an abstract IND–1–CCA-secure labelled PKE).

More precisely, we replace the verifiable shuffle—which is used in the tally phase to mix the encrypted votes and trackers while erasing connections between the votes and the voters—with a parametric $\mathsf{Multiset}$ distribution, which takes as parameters a list of vote/tracker pairs $(v_i, \mathsf{tr}_i)_i$, calculates all possible permutations of the original list, and defines the uniform distribution over the result. Sampling in $\mathsf{Multiset}(\ell)$ captures the semantics of a perfect shuffle on $\ell$, with a proof of correct shuffle computed separately.[1] Formalizing proofs for interactive protocols, such as verifiable shuffles, in EASYCRYPT remains a complex task, and is somewhat orthogonal to our contributions here. In particular, in a setting—like ours—in which the tallier is honest, the shuffle is indeed indistinguishable from our idealization. Prior work [14], [15] has proved that the interactive variants of the verifiable shuffles suggested for use with Selene are zero-knowledge proofs which leak no information; this would suffice to prove equivalence with our idealisation. However, this has not been machine checked for the interactive variant due to issues the currently available tools have with handling random oracles.

Before being cast, the votes in Selene are encrypted using the ElGamal public key encryption system [13], and a non-interactive proof of knowledge of the underlying plaintext is appended to the ciphertext. In our EASYCRYPT formalization, we abstract away details of the underlying cryptosystem, and encrypt the votes using an abstract labelled PKE that we assume is IND–1–CCA secure. The ElGamal with proof of knowledge construction used in Selene has in fact been proven to be IND–1–CCA secure [4], [7]. This proof remains out of reach of machine-checking due to its use of the rewinding lemma.

In addition, since we focus on privacy in this paper, we also remove the signatures Selene[2] includes on cast ballots to prevent ballot stuffing. The signatures cannot compromise privacy: The signing keys are independent of the encryption of the ballot, the ciphertexts that are signed are public, and the signatures are anyway stripped before shuffling, so they cannot be correlated to any plaintext ballot.

These simplifications, taken together, yield the following model for Selene.

[1] We note that $\mathsf{Multiset}$ itself is not probabilistic polynomial time. We treat it as an ideal functionality for verifiable shuffles, whose complexity would normally be probabilistic polynomial time.

[2] We also have an EasyCrypt proof for Selene with signatures. This is available with the other proofs at https://github.com/mortensol/du-mb-bpriv.

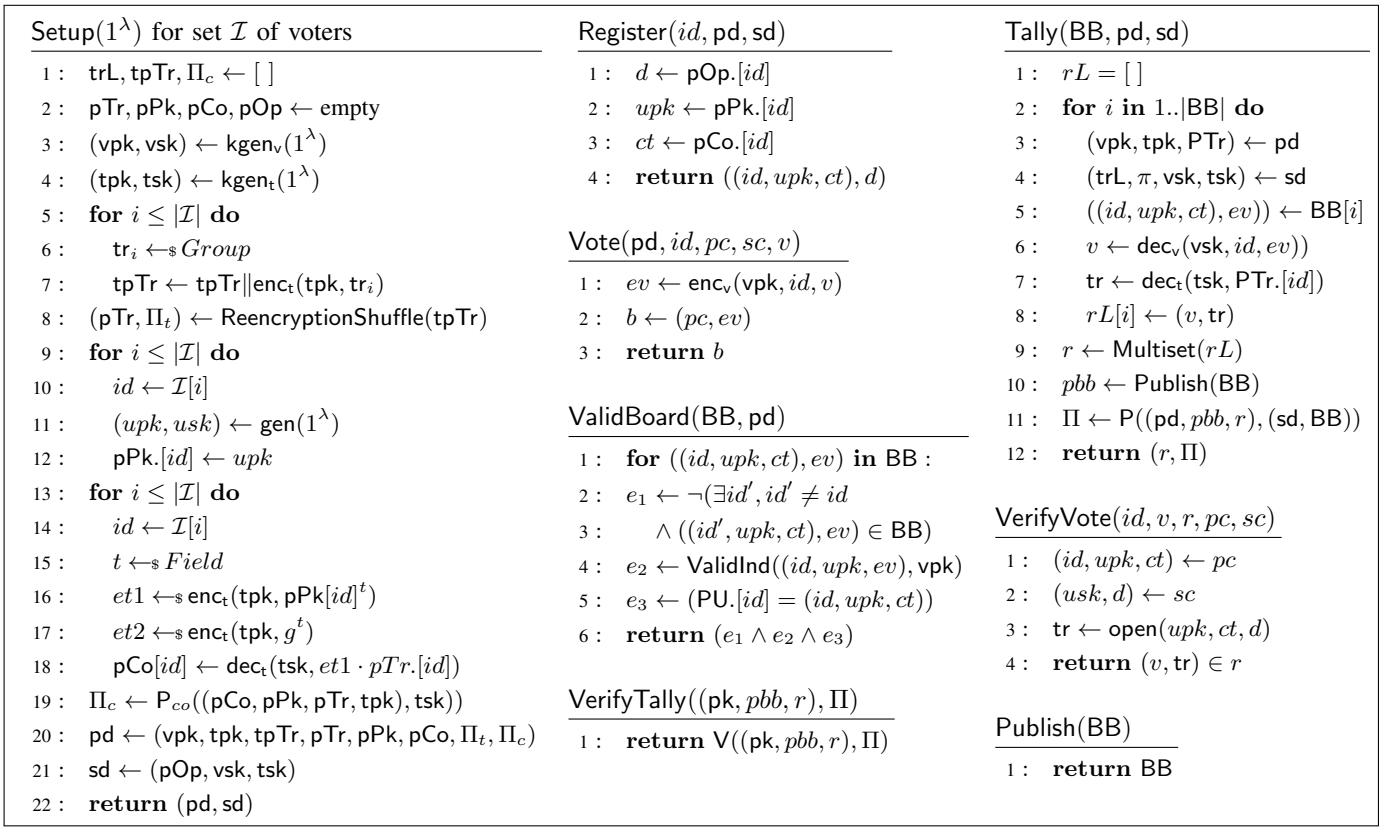


Fig. 5. Algorithms defining the Selene[$\mathsf{E_v}, \mathsf{E_t}, \mathsf{C}, \Sigma$, ValidInd] voting scheme, given an IND–1–CCA secure labelled PKE scheme $\mathsf{E_v} = (\mathsf{kgen_v}, \mathsf{enc_v}, \mathsf{dec_v})$, an IND–CPA secure homomorphic encryption scheme $\mathsf{E_t} = (\mathsf{kgen_t}, \mathsf{enc_t}, \mathsf{dec_t})$, zero-knowledge proof systems $\Sigma_{ta} = (\mathsf{P}_{ta}, \mathsf{V}_{ta})$, $\Sigma_{tsh} = (\mathsf{P}_{tsh}, \mathsf{V}_{tsh})$ and $\Sigma_{co} = (\mathsf{P}_{co}, \mathsf{V}_{co})$ for the tally proof, the tracker shuffle proof and the proof that commitments are correctly formed, respectively, a commitment scheme $\mathsf{CP} = (\mathsf{gen}, \mathsf{commit}, \mathsf{open})$, and the abstract operator ValidInd.

```
lemma du_mb_bpriv &m : BP.setidents{m} = BP.setH{m} `|` BP.setD{m} =>
(** The du-mb-BPRIV advantage of some adversary A upper bounded by the sum of: **)
`| Pr[DU_MB_BPRIV_L(Selene(Et,Ev,P,Ve,C,CP),A,HRO.ERO,G).main() @ &m: res]
  - Pr[DU_MB_BPRIV_R(Selene(Et,Ev,P,Ve,C,CP),A,HRO.ERO,G,S,Recover').main() @ &m: res]|
(** - the advantages of BVFR(G) and BVFR(S) in breaking the Voting Friendly Relation; **)
<= Pr[VFRS(Et,Ev,BVFR(Selene(Et,Ev,P,Ve,C,CP),A,CP),R,HRO.ERO,G).main() @ &m: res]
 + Pr[VFRS(Et,Ev,BVFR(Selene(Et,Ev,P,Ve,C,CP),A,CP),R,HRO.ERO,S).main() @ &m: res]
(** - the ZK advantage of adversary BZK against the underlying NIZK; and **)
 + `| Pr[ZK_L(R(Et,Ev,HRO.ERO),P,BZK(Et,Ev,P,C,Ve,A,CP,HRO.ERO),G).main() @ &m: res]
    - Pr[ZK_R(R(Et,Ev,HRO.ERO),S,BZK(Et,Ev,P,C,Ve,A,CP,HRO.ERO)).main() @ &m: res]|
(** - the IND1-CCA advantage of adversary BCCA against the ballot-encryption scheme **)
 + `| Pr[Ind1CCA(Ev,BCCA(Selene(Et,Ev,P,Ve,C,CP),Et,CP,A,S),HRO.ERO,Left).main() @ &m: res]
    - Pr[Ind1CCA(Ev,BCCA(Selene(Et,Ev,P,Ve,C,CP),Et,CP,A,S),HRO.ERO,Right).main() @ &m: res]|
```

Fig. 6. EASYCRYPT lemma establishing that Selene satisfies du–mb–BPRIV. HRO.ERO and G are independent random oracles. S is the ZK simulator.

*Definition 9:* Let $\mathsf{E_v}$ be a poly–IND–1–CCA secure labelled PKE, let $\mathsf{E_t}$ be an IND–CPA secure PKE, let $\Sigma_{\mathcal{R}} = (\mathsf{P}, \mathsf{V})$ be a proof system for a relation $\mathcal{R}$ and let $\mathsf{CP} = (\mathsf{gen}, \mathsf{commit}, \mathsf{open})$ be a commitment protocol. We define *Selene* as the voting system built upon the algorithms given in Figure 5, which we informally describe below.

Setup: Takes as input a security parameter $\lambda$ and returns a key pair (vpk, vsk) used to encrypt and decrypt votes, a key pair (tpk, tsk) used to encrypt and decrypt trackers, a list tpTr of encrypted trackers, finite maps pTr, pPk, pCo and pOp from voter identities to encrypted trackers, public commitment keys, tracker commitments and openings, respectively, as well as a proof $\Pi_t$ of correct shuffle of the trackers and a proof $\Pi_c$ that the tracker commitments are correctly formed. The public data is

$$\mathsf{pd} = (\mathsf{vpk}, \mathsf{tpk}, \mathsf{tpTr}, \mathsf{pTr}, \mathsf{pPk}, \mathsf{pCo}, \Pi_t, \Pi_c)$$

and the secret data is

$$\mathsf{sd} = (\mathsf{pOp}, \mathsf{vsk}, \mathsf{tsk}).$$

Register: Takes as input a voter identity and a pair $(\mathsf{pd}, \mathsf{sd})$ of public and secret data, and returns the voter's public commitment key, the commitment to her tracker and an opening to the commitment.

Publish: Outputs the public part of the ballot box.

Vote: Encrypts a vote $v$ and outputs the ciphertext together with the voter's public credential.

VerifyTally: Run the V algorithm of the proof-system to check if the tally proof is valid.

ValidBoard: For each element in the ballotbox BB, we perform three checks: every ballot contains a unique voter identity, every ballot is well-formed, and every public credential corresponds to the correct identity.

Tally: Decrypts every encrypted vote in the ballot box BB and the tracker for each voter. Returns the multiset of all vote/tracker pairs $(v, tr)$.

VerifyVote: To verify, a voter opens her tracker commitment and checks if her vote $v$ and tracker tr is in the list of vote/tracker pairs returned by Tally.

The following theorem establishes that Selene satisfies du–mb–BPRIV.

*Theorem 3:* Let $\mathcal{V} = \mathsf{Selene}(\mathsf{E_v}, \mathsf{E_t}, \Sigma_{\mathcal{R}}, \mathsf{CP}, \mathsf{ValidInd})$, where $\mathsf{ValidInd}(\mathsf{pc}, \mathsf{vpk}) = \top$ for $c \leftarrow \mathsf{enc_v}(\mathsf{vpk}, id, v)$ and any public encryption key vpk, identity $id$, public credential $pc$ and vote $v$. For any du–mb–BPRIV adversary $\mathcal{A}$, there exists a simulator Sim and four adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}_S$ and $\mathcal{D}_G$, such that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{du\text{–}mb\text{–}bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda) \leq\ & \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D}_G,\mathcal{V},\Sigma_{\mathcal{R}}}(\lambda) = 1\right] \\ & + \Pr\left[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D}_S,\mathcal{V},\Sigma_{\mathcal{R}}}(\lambda) = 1\right] \\ & + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B},\mathsf{P},\mathsf{Sim},\mathcal{R}}(\lambda) + \mathsf{Adv}^{\mathsf{poly\text{–}ind1cca}}_{\mathcal{C},\mathsf{E_v},n}(\lambda).\end{aligned}$$

Theorem 3 corresponds to lemma `du_mb_bpriv` in `SeleneBpriv.ec`. Figure 6 displays the EASYCRYPT formulation of Theorem 3. The lemma itself is inside a section which quantifies over all core components: `Et` and `Ev` denote the encryption schemes used for the trackers and the votes, respectively, `P` and `Ve` denote the NIZK's prover and verifier, `C` denotes the ValidInd algorithm, `CP` denotes the commitment protocol. `A` is the adversary. The zero-knowledge simulator `S` and the random oracles for tracker encryption (`HRO.ERO`) and for the NIZK proof system (`G`) are defined concretely in the code, but not fully displayed in this paper. The VFR, zero knowledge and poly–IND–1–CCA security experiments are denoted by `VFRS`, `ZK_L`, `ZK_R` and `Ind1CCA`, respectively, while the respective reductions are denoted by `BVFR`, `BZK` and `BCCA`. These are also given concrete definitions. Also note that the `Ind1CCA` module is parameterized by a left-or-right module, representing the case where $\beta = 0$ and the case where $\beta = 1$, respectively. The du–mb–BPRIV security experiment is parameterized by a recovery algorithm, and we use the concrete recovery algorithm described in Section III-A.

We now sketch the proof of Theorem 3. The EASYCRYPT formalization of the full proof is found in the file `SeleneBpriv.ec`. Unless explicitly stated, all the modules and lemmas we refer to in the following are also found in this file.

In our EASYCRYPT formalization, we split the security experiment $\mathsf{Exp}^{\mathsf{du\text{–}mb\text{–}BPRIV},\mathsf{Recover},\beta}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}$ into two different games, one for $\beta = 0$ and one for $\beta = 1$. The difference between the two games is, as described earlier, what the tally algorithm does. These games are modeled in the modules `DU_MB_BPRIV_L` and `DU_MB_BPRIV_R`, respectively, which are found in the file `VotingSecurity_mb.ec`.

Starting out from the left side security experiment, the first step is to replace the tally proof produced by the prover in the proof system, by a simulated proof produced by the simulator Sim. This change is modeled in the game `G1L`. Provided that the proof system is zero-knowledge, the adversary cannot distinguish between the original game and the game where we simulated the proof, except with negligible probability.

We then define a new game, `G2L`, where we stop decrypting honestly created ciphertexts, and instead use the ciphertexts stored in V (as described in Section III). Ciphertexts submitted by the adversary are decrypted as usual. We also remove one of the bulletin boards from the vote oracle, so that only the left-side votes are stored. We prove that `G1L` and `G2L` are equivalent. The equivalence follows from the correctness property of the encryption scheme used to encrypt the votes, and the fact that the adversary only gets to see $\mathsf{BB}_0$ in the left side security experiment.

Starting out from the right side security experiment (`DU_MB_BPRIV_R`), we first stop decrypting honest ciphertexts, and prove that the resulting game (`G1R`) is equivalent to `DU_MB_BPRIV_R`. The intuition is the same as for the equivalence between `G1L` and `G2L`.

We then define a game `G2R` where we stop performing recovery on the adversarially created ballot box, and simply perform the tally on the ballot box the adversary outputs. We prove that `G1R` and `G2R` are equivalent. Intuitively, this holds because the honest votes no longer come from the adversary's board, but from V, and the ballots submitted by the adversary are present both on the adversary's board BB and on the recovered board, by definition of our recovery algorithm.

In the final game, `G3R`, we remove one of the bulletin boards in the vote oracle. This is similar to what we did in `G2L`, but now only the right side votes are stored. We prove that `G2R` and `G3R` are equivalent. This also shows that the final game on the right side, `G3R`, is completely equivalent to `DU_MB_BPRIV_R`.

Finally, we show that the probability in distinguishing between the games `G2L` and `G3R` is equivalent to the probability of winning the IND–1–CCA game.

## VI. CONCLUDING REMARKS AND FUTURE WORK

In this work we presented a refined version of the mb–BPRIV privacy definition which we call delay-use malicious-ballotbox ballot privacy (du–mb–BPRIV). Our new definition allows the modeling of schemes (such as Selene) where verification occurs after tallying. The security claim is also more explicit. We formalised our new definition in the interactive theorem prover EasyCrypt and showed that labelled

MiniVoting, Belenios and Selene all satisfy the definition. We also proved that MiniVoting and Belenios satisfy the original mb–BPRIV privacy definition.

While we have encoded Selene correctly in what we firmly believe is the most appropriate privacy definition in literature, our work highlights certain defiances in privacy definitions which future work should address. The defiances are fairly deep and addressing them is far out of scope for this work. We will, however, briefly mention two principal defiances of mb–BPRIV and related definitions. First, the definition, while handling a malicious bulletin board, assumes honest setup. Secondly, du–mb–BPRIV and related definitions are highly calibrated to schemes where the auxiliary data produced by tally to be used in verification are zero-knowledge proofs. In particular, schemes like Selene which output trackers to use for verification are difficult to express in these definitions.

Further, the BPRIV style of definition implies certain restrictions. As an example, the adversary can only see the result and verification from the left side board which constrains the attacker model. In particular, this means that we cannot detect privacy attacks relying on inducing candidate-specific errors for an observed voter while giving the adversary access to whether the corresponding voter verification fails or not, see e.g. [16] for such a style of attack. Of course, such attacks can be ruled out by considering recovery functions preventing any changes to honestly cast votes as in [12], but it is not the case in general. An important line of future research is thus to find alternative definitions capturing both more general and more transparent attacker models, e.g. by decreasing the generality of the definition or to consider simulation based security.

## Acknowledgment

We thank the anonymous reviewers at IEEE Computer Security Foundations Symposium for their helpful comments. This work was supported by the Luxembourg National Research Fund (FNR) and the Research Council of Norway (NFR) for the joint project SURCVS (FNR project ID 11747298, NFR project ID 275516), and by the FNR project STV (C18/IS/12685695/IS/STV/Ryan). We thank the (rest of the) EasyCrypt team for the continued development of the tool and its libraries.

## References

[1] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. *EasyCrypt: A Tutorial*, pages 146–166. Springer International Publishing, Cham, 2014.

[2] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[3] Mihir Bellare and Amit Sahai. Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 519–536. Springer, Heidelberg, August 1999.

[4] David Bernhard. *Zero-knowledge proofs in theory and practice*. PhD thesis, University of Bristol, 2014.

[5] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*, pages 499–516. IEEE Computer Society Press, May 2015.

[6] David Bernhard, Véronique Cortier, Olivier Pereira, Ben Smyth, and Bogdan Warinschi. Adapting helios for provable ballot privacy. In Vijay Atluri and Claudia Díaz, editors, *ESORICS 2011*, volume 6879 of *LNCS*, pages 335–354. Springer, Heidelberg, 2011.

[7] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Heidelberg, December 2012.

[8] Alessandro Bruni, Eva Drewsen, and Carsten Schürmann. Towards a mechanized proof of selene receipt-freeness and vote-privacy. In *International Joint Conference on Electronic Voting*, pages 110–126. Springer, 2017.

[9] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *2017 IEEE Symposium on Security and Privacy*, pages 993–1008. IEEE Computer Society Press, May 2017.

[10] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, and Bogdan Warinschi. Machine-checked proofs for electronic voting: Privacy and verifiability for belenios. In Steve Chong and Stephanie Delaune, editors, *CSF 2018Computer Security Foundations Symposium*, pages 298–312. IEEE Computer Society Press, 2018.

[11] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 327–344. Springer, Heidelberg, September 2014.

[12] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In Limin Jia and Ralf Küsters, editors, *CSF 2020Computer Security Foundations Symposium*, pages 17–32. IEEE Computer Society Press, 2020.

[13] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18. Springer, Heidelberg, August 1984.

[14] Thomas Haines, Rajeev Goré, and Bhavesh Sharma. Did you mix me? formally verifying verifiable mix nets in electronic voting. In *IEEE Symposium on Security and Privacy*, pages 1748–1765. IEEE, 2021.

[15] Thomas Haines, Rajeev Goré, and Mukesh Tiwari. Verified verifiers for verifying elections. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 685–702. ACM Press, November 2019.

[16] Steve Kremer and Peter B Rønne. To du or not to du: A security analysis of du-vote. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 473–486. IEEE, 2016.

[17] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. select: A lightweight verifiable remote voting system. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 341–354. IEEE, 2016.

[18] Peter B Rønne, Peter YA Ryan, and Marie-Laure Zollinger. Electryo, in-person voting with transparent voter verifiability and eligibility verifiability. *arXiv preprint arXiv:2105.14783*, 2021.

[19] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 176–192. Springer, Heidelberg, February 2016.

[20] Muntadher Sallal, Steve A. Schneider, Matthew Casey, Constantin Catalin Dragan, François Dupressoir, Luke Riley, Helen Treharne, Joe Wadsworth, and Phil Wright. VMV: augmenting an internet voting system with selene verifiability. *CoRR*, abs/1912.00288, 2019.

[21] Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. https://eprint.iacr.org/2001/112.

[22] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. https://eprint.iacr.org/2004/332.

[23] Marie-Laure Zollinger, Peter B Rønne, and Peter YA Ryan. Short paper: Mechanized proofs of verifiability and privacy in a paper-based e-voting scheme. In *International Conference on Financial Cryptography and Data Security*, pages 310–318. Springer, 2020.