Cortier, V., Dragan, C. C., Dupressoir, F., Schmidt, B., Strub, P-Y., & Warinschi, B. (2017). Machine-Checked Proofs of Privacy for Electronic Voting Protocols. In *2017 IEEE 38th IEEE Symposium on Security and Privacy (SP 2017): Proceedings of a meeting held 22-26 May 2017, San Jose, California, USA* (pp. 993-1008). Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/SP.2017.28

Machine-Checked Proofs of Privacy for Electronic Voting Protocols

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

Constantin Cătălin Drăgan
LORIA, CNRS & Inria
catalin.dragan@loria.fr

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

Benedikt Schmidt*
IMDEA Software Institute
beschmi@gmail.com

Pierre-Yves Strub
École Polytechnique
pierre-yves@strub.nu

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

***Abstract*—We provide the first machine-checked proof of privacy-related properties (including ballot privacy) for an electronic voting protocol in the computational model. We target the popular Helios family of voting protocols, for which we identify appropriate levels of abstractions to allow the simplification and convenient reuse of proof steps across many variations of the voting scheme. The resulting framework enables machine-checked security proofs for several hundred variants of Helios and should serve as a stepping stone for the analysis of further variations of the scheme.**

In addition, we highlight some of the lessons learned regarding the gap between pen-and-paper and machine-checked proofs, and report on the experience with formalizing the security of protocols at this scale.

1. Introduction

Ensuring accuracy and security of electronic elections is a challenge that goes far beyond the scope of safety and security as traditionally considered in computer science. Nevertheless, previous audits of voting systems [40], [42] suggest that many of the most mundane issues could be prevented by using open source and formally verified implementations. However, the formal verification of voting systems down to deployed software is particularly challenging, for several reasons. First, defining security properties for voting systems remains an active topic of investigation [13], [22]; moreover, many definitions are expressed in a simulation-based style whereas most efforts to formally verify cryptographic constructions (with a few notable exceptions such as [4]) focus on the game-based style. Second, "real-world" adversary models for electronic voting would need to consider adversarial models that go beyond the usual view of provable security, and account for the possibility that the voting system might be backdoored or run in corrupted environments. Third, protocols often have multiple variants, with subtle but theoretically significant differences in their security analysis. Last, electronic voting systems are distributed, with multiple implementations of voting clients, introducing additional complexity for reasoning about their implementations. Taken together, these challenges make end-to-end formal verification of voting systems out of reach of current technology.

Scope of work. We provide the first machine-checked computational proofs of privacy properties for Helios [2], [26], an emblematic voting protocol that has received significant analysis and has also been used in several elections. Our proofs establish ballot privacy and related properties introduced in [13] on algorithmic descriptions of a broad family of Helios variants. The proofs[1] are built using EasyCrypt [8], [9], an interactive proof assistant focused on constructing proofs of computational security for cryptographic constructions.

Technical contributions. Using EasyCrypt, we develop a machine-checked framework for proving ballot privacy for a broad class of voting protocols from the Helios family. In order to manage the complexity of the proof and to accommodate the large number of instances of the protocol we consider (a few hundred), we factor out the proof in several steps:

We first introduce *Labelled-MiniVoting*, a core voting protocol that enhances the MiniVoting protocol defined in [14] with labels. The construction relies on an arbitrary labelled public-key encryption scheme (used to encrypt the votes) and an abstract proof system used by the tallying authorities to show the validity of the election result. The addition of labels is essential to instantiate our scheme more broadly. We formalize *Labelled-MiniVoting* and build a machine-checked proof that it achieves ballot privacy, as well as strong consistency and strong correctness, as defined in [13]. Informally, these properties respectively capture the idea that voting is anonymous, that tallying does not leak any information, and that honestly created ballots are always considered valid. The proofs are carried out under the assumptions that the underlying encryption scheme

*Now at Google Inc.

1. https://github.com/catalindragan/minivoting-privacy.

achieves (an adaptation to the case of labelled schemes of) IND-1-CCA security, and that the underlying proof system satisfies zero-knowledge and *voting friendliness* (a property that we introduce and which captures simple requirement for an abstract proof system to be compatible with ballot privacy).

We then carry out the proofs of these three properties (ballot privacy, strong correctness and strong consistency) through to a partial instantiation of *Labelled-MiniVoting*, called *PreHelios*, in which the labelled public-key encryption scheme is specified as the composition of a public-key encryption scheme and of a proof system (as in Helios). The machine-checked proofs of privacy for *PreHelios* simply follow from the generic theorems on *Labelled-MiniVoting*. We introduce two broad families of Helios variants:

Helios-mix, a mixnet-based variant that refines *PreHelios* by instantiating the counting function to reveal the multiset of votes and tallying using a secure mixnet, and

Helios-hom, a homomorphic variant that refines *PreHelios* by instantiating the counting function to be the sum of all votes, and tallying by homomorphically computing the sum before decrypting the final result.

In both cases, carrying the results obtained on *PreHelios* down to the relevant variant is not done using a direct instantiation, but requires in-depth changes to the tallying algorithm. This refactoring is supported by machine-checked proofs of black-box equivalence between the tallying algorithms of *PreHelios* and its variants (under suitable assumptions). One main advantage of our proof framework is its modularity, which allows us to easily replace one component of the scheme (here, the tallying algorithm) with an equivalent one, without having to reprove the entire scheme. We use the same proof technique to derive the security of *Helios-hom-IDweed*, a variant of *Helios-hom* where the weeding of invalid ballots performed before tallying–an important process, often overlooked, that may lead to privacy breaches [24]–is made lighter without loss of security. This yields a machine-checked proof of ballot privacy, strong consistency and strong correctness for *Helios-mix*, *Helios-hom*, and *Helios-hom-IDweed*.

Finally, we derive specifications and machine-checked proofs of privacy for large families of secure variants of Helios from *Helios-mix*, *Helios-hom*, and *Helios-hom-IDweed*. The proofs are obtained by further instantiating our earlier results and discharging all additional hypotheses introduced by our abstractions. More precisely, we provide a simple design interface, that allows the user to select various parameters (for example, the counting function, or the function that selects information about the ballot to be published on the bulletin board). The resulting voting scheme can then be automatically generated and proved secure in EasyCrypt. In total, we prove the security of more than 500 variants of Helios. In particular, we retrieve existing implemented variants of Helios such as Helios-v3-mix, Helios-v3-hom, and Helios-v4.

Related work. Automatic proofs of privacy for voting protocols have been provided for some protocols of the literature such as FOO [27], Helios [24], or the Norwegian e-voting protocol [25]. However, these proofs are conducted in symbolic models, which are considerably more abstract than cryptographic models. Moreover, the use of automatic tools (such as ProVerif [17]) often requires significant simplifications of both the protocol and the underlying cryptographic primitives (for example, the ballot box may only accept ballots in a certain order). Helios has also been proved private in $r\mathsf{F}^*$ [20], still assuming perfect cryptographic primitives (the adversary may only call the primitives through an abstract, ideal library). We are not aware of any machine-checked proof of voting schemes in cryptographic models, assuming standard cryptographic primitives. While pen-and-paper proofs for Helios exist [13], [14], we emphasize that our work is *not* a simple translation of those proofs into the EasyCrypt language. First, many crucial details that are typically omitted in hand proofs (for example, a careful treatment of the random oracles) had to be filled-in. More importantly, our formalization considerably generalizes existing proofs, providing machine-checked proofs for hundreds of variants of Helios. This would have been impossible to achieve with pen-and-paper, and with the same degree of confidence.

A note on concrete vs asymptotic security. EasyCrypt adheres to the principles of practice-oriented provable security, and our formal proofs always state concrete bounds on adversarial advantages. For simplicity, we place most of the discussion below in an asymptotic context. For precision, we give concrete statements for our main security theorems. These are backed up by the formal EasyCrypt development, available from https://github.com/catalindragan/minivoting-privacy.

2. Voting

We first introduce some basic cryptographic primitives, used in the voting schemes we consider. We then recall useful security notions for voting systems (ballot privacy, strong consistency and strong correctness) [13], and state our main result.

2.1. Building blocks

We present the cryptographic building blocks used in voting systems. Our presentation is based on two primitives and their associated security notions: *labelled public-key encryption schemes* and *non-interactive proof systems*. All of our models and proofs are set in the random oracle model, which we recall first.

RANDOM ORACLE MODEL. The random oracle model is a model for hash functions: to compute the value of the hash function on a point, any party can invoke an oracle $\mathcal{O}$ that implements a truly random function from some domain D to some range C. One way to think about this oracle is that $\mathcal{O}$ maintains a table T, initially empty. Whenever an algorithm calls $\mathcal{O}(d)$ for some $d \in D$, $\mathcal{O}$ checks if there

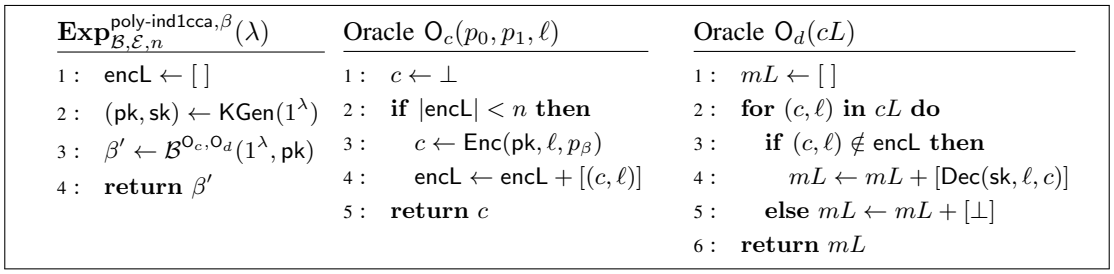

Figure 1. In $\mathsf{Exp}^{\mathsf{poly\text{-}ind1cca},\beta}_{\mathcal{B},\mathcal{E},n}$, the adversary $\mathcal{B}$ has access to the set of oracles $\{\mathsf{O}_c,\mathsf{O}_d\}$. The adversary is allowed to call the O_d oracle at most once.

exists an entry (d,c) in T and, if so, it returns c; otherwise $\mathcal{O}$ randomly generates a value $c' \in C$, adds (d,c') to T, and outputs c'.

Strictly speaking, in the random oracle model all algorithms are given oracle access to $\mathcal{O}$; to avoid cluttered notations, we choose to not explicitly show this dependency, but we emphasize it whenever appropriate.

In particular, the presence of random oracles has a significant impact on the difficulty of the EasyCrypt formalization. We discuss this and associated insights in Section 5.

LABELLED PUBLIC-KEY ENCRYPTION SCHEME. The notion of labelled public-key encryption scheme extends the classical definition of a public-key encryption scheme by including a tag, called a label [35], [38], [39]. Essentially, the tag is data that is non-malleably attached to the ciphertext and can be used, for example, by whoever encrypts to specify the context in which the ciphertext is to be used. In particular, decryption using the incorrect label should not reveal any information about the original plaintext. Formally, a labelled public-key encryption scheme is defined as follows.

Definition 1. A *labelled public-key encryption scheme* with public keys in PK, secret keys in SK, plaintexts in M, ciphertexts in C, and labels in L, is 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 key pair in $\mathsf{PK} \times \mathsf{SK}$;

Enc is a randomized algorithm which on input a public key in PK, a label in L and a plaintext in M outputs a ciphertext in C;

Dec is a deterministic algorithm which on input a secret key in SK, a label in L and a ciphertext in C outputs an element in $\mathsf{M}_\perp$, that is either a plaintext in M or a special error symbol $\perp$.

We demand that for any $(\mathsf{pk},\mathsf{sk})$ output by KGen, any label $\ell \in \mathsf{L}$ and any message $m \in \mathsf{M}_\perp$, if $C \leftarrow \mathsf{Enc}(\mathsf{pk},\ell,m)$ then $\mathsf{Dec}(\mathsf{sk},\ell,C) = m$.

We note that if the label is fixed or is empty, labelled public-key encryption scheme reduces to the standard notion of public-key encryption.

Encryption schemes used in electronic voting protocols are often required to be *homomorphic*, allowing some limited forms of computation on encrypted data without decrypting it (for example, homomorphic tallying).

Definition 2. A *homomorphic public-key encryption scheme* is a public-key encryption scheme $\mathcal{E}$ together with a deterministic algorithm Add where the space of plaintexts can be equipped with a commutative monoid structure $(\mathsf{M},0,+)$ such that

$$\mathsf{Dec}(\mathsf{sk},\mathsf{Add}(cL)) = \sum_{i=1}^{|cL|} \mathsf{Dec}(\mathsf{sk},cL[i]),$$

for any list of ciphertext cL, and any secret key sk.

That is, addition of ciphertexts has as effect addition on the underlying plaintexts. There are multiple security definitions for encryption schemes. The security of the protocols we consider in this paper relies on *indistinguishability under chosen-ciphertext attack with one parallel decryption query* (IND-1-CCA) [12] of the underlying labelled public-key encryption scheme. Intuitively, IND-1-CCA requires that no adversary can distinguish between encryptions of two messages of the same length with probability significantly greater than $1/2$, even if provided with a one-time access to a batch decryption oracle.

A related, apparently stronger notion considers a multi-challenge version (where the adversary sees polynomially many challenge queries). We write poly-IND-1-CCA for this latter security notion.

These notions are formalized by the experiment $\mathsf{Exp}^{\mathsf{poly\text{-}ind1cca},\beta}_{\mathcal{B},\mathcal{E},n}$ defined in Figure 1. The experiment considers an adversary $\mathcal{B}$ with at most n access to the challenge oracle O_c (that encrypts the left message if $\beta = 0$ and the right message if $\beta = 1$) and one-time access to the decryption oracle O_d defined using the encryption scheme $\mathcal{E}$.

The advantage of the poly-IND-1-CCA adversary $\mathcal{B}$ over the scheme $\mathcal{E}$ is defined as:

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

We say that a labelled public-key encryption scheme $\mathcal{E}$ is n-*challenge* poly-IND-1-CCA-*secure* if $\mathsf{Adv}^{\mathsf{poly\text{-}ind1cca}}_{\mathcal{B},\mathcal{E},1}(\lambda)$ is negligible (as a function of λ) for all p.p.t. $\mathcal{B}$. Note that $\mathsf{Adv}^{\mathsf{poly\text{-}ind1cca}}_{\mathcal{B},\mathcal{E},1}(\lambda)$ is essentially the advantage for the single-challenge IND-1-CCA notion. The following lemma establishes, by a standard hybrid argument, that multi-challenge security is asymptotically the same as single challenge security.

Lemma 1. *A labelled public-key encryption scheme $\mathcal{E}$ is* IND-1-CCA*-secure if and only if it is n-challenge* poly-IND-1-CCA*-secure for some polynomially bounded n. Specifically, for any polynomially bounded adversary $\mathcal{A}$ there exists a polynomially bounded adversary $\mathcal{B}$ such that for any n and any λ the following statement holds:*

$$\mathsf{Adv}^{\mathsf{poly\text{-}ind1cca}}_{\mathcal{B},\mathcal{E},1}(\lambda) = \frac{1}{n} \cdot \mathsf{Adv}^{\mathsf{poly\text{-}ind1cca}}_{\mathcal{A},\mathcal{E},n}(\lambda).$$

For example, El Gamal is a homomorphic encryption scheme, which, together with a proof of knowledge of the randomness used for the encryption, has been shown to be IND-1-CCA-secure [15]. Interestingly, in Helios which uses this scheme, the proof of knowledge is primarily used to ensure that the voter encrypts a valid vote, yet this proof also protects the ciphertext from being modified. Notice that ciphertexts still have an El Gamal component, which can be used to homomorphically calculate over plaintexts.

PROOF SYSTEMS. Proof systems are a useful tool to ensure that certain operations have been performed correctly. The formalization starts with a relation $\mathcal{R} \subseteq \mathsf{X} \times \mathsf{W}$ where the elements of X are called *statements* and those of W are called *witnesses*.[2] For example, in Helios, a typical relation is that the ciphertext (the statement) corresponds to the encryption of 0 or 1. One can use as a witness the randomness used to form the ciphertext. A proof system consists of a prover and a verifier algorithm which work on a common input $x \in \mathsf{X}$; the prover has an additional input $w \in \mathsf{W}$. In a non-interactive proof systems (as those considered in this paper) the prover uses its inputs to compute a proof $\pi \in \mathsf{PR}$ and sends it to the verifier who then takes a binary decision. More formally:

Definition 3. A non-interactive *proof system* for relation $\mathcal{R}$ is a pair of efficient algorithms $\Sigma = (\mathsf{P}, \mathsf{V})$. P has as input a statement in X and a witness in W, and produces a proof $\pi \in \mathsf{PR}$; V takes as input a statement in X and a proof in PR, and produces an output in $\{0,1\}$. For clarity, we write $\Sigma_{\mathcal{R}}$ for a proof system for relation $\mathcal{R}$.

Useful proof systems need to satisfy three properties, soundness, completeness, and zero-knowledge. Here we only recall the latter two, as soundness has no bearing on vote privacy.

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

A proof system $\Sigma_{\mathcal{R}}$ is *zero-knowledge*, if the proof does not leak any information besides the fact that the relation is valid. This is usually formalized by demanding the existence of a p.p.t. simulator S that produces valid-looking proofs for any statement $x \in \mathsf{X}$ without access to a corresponding witness. More formally, consider the zero-knowledge adversary $\mathcal{B}$ in the following experiments:

2. In typical instantiations $\mathcal{R}$ is an NP-relation.

$\mathbf{Exp}^{\mathsf{zk},0}_{\mathcal{B},\mathsf{P},\mathcal{R}}(\lambda)$	$\mathbf{Exp}^{\mathsf{zk},1}_{\mathcal{B},\mathsf{S},\mathcal{R}}(\lambda)$
$(x, w, \mathsf{state}) \leftarrow \mathcal{B}(1^\lambda)$	$(x, w, \mathsf{state}) \leftarrow \mathcal{B}(1^\lambda)$
$\pi \leftarrow \bot$	$\pi \leftarrow \bot$
if $(\mathcal{R}(x,w))$ **then**	**if** $(\mathcal{R}(x,w))$ **then**
$\quad \pi \leftarrow \mathsf{P}(x,w)$	$\quad \pi \leftarrow \mathsf{S}(x)$
$\beta' \leftarrow \mathcal{B}(\mathsf{state}, \pi)$	$\beta' \leftarrow \mathcal{B}(\mathsf{state}, \pi)$
return β'	**return** β'

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:

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

Recall that we work in the random oracle model. Here, the simulator has the additional capability (or responsibility) of answering any calls the adversary makes to random oracles *used in the proof system*. To keep notation simple we do not show this dependence in the above formalization but, of course, we account for it in our proofs.

2.2. Vote privacy in single-pass voting schemes

In this section we recall the syntax and security properties of *single-pass voting* schemes, the class of schemes that we treat in this paper. First we recall their syntax and then give an overview of their desired properties related to vote privacy. A *single-pass voting system* [13] is a tuple of algorithms

$$\mathcal{V} = \big(\mathsf{Setup},\ \mathsf{Vote},\ \mathsf{Valid},\ \mathsf{Publish},\ \mathsf{Tally},\ \mathsf{Verify}\big)\ .$$

$\mathsf{Setup}(1^\lambda, m)$**:** Returns a pair of keys $(\mathsf{pk}, \mathsf{sk})$, and creates a map uL that assigns m voter identities to their associated labels.

$\mathsf{Vote}(id, \ell, v, \mathsf{pk})$**:** Constructs a ballot b that voter id uses to cast their vote v, with label ℓ.

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

$\mathsf{Publish}(\mathsf{BB})$**:** Returns the public view pbb of BB called the public bulletin board; for example, the public bulletin board can contain the whole content of the ballot box, only the labels involved in ballots, or no information at all.

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

$\mathsf{Verify}((\mathsf{pk}, pbb, r), \Pi)$**:** Checks that Π is a valid proof of correct computation for result r and public ballot box pbb.

In this section we recall several properties of single-pass voting schemes. We start with ballot privacy, the key security guarantee that these schemes need to satisfy.

BALLOT PRIVACY PROPERTY. A voting scheme $\mathcal{V}$ ensures ballot privacy [13] (BPRIV, for short) if the ballots themselves do not reveal any information about the votes that were cast. This holds even for adversaries that can cast

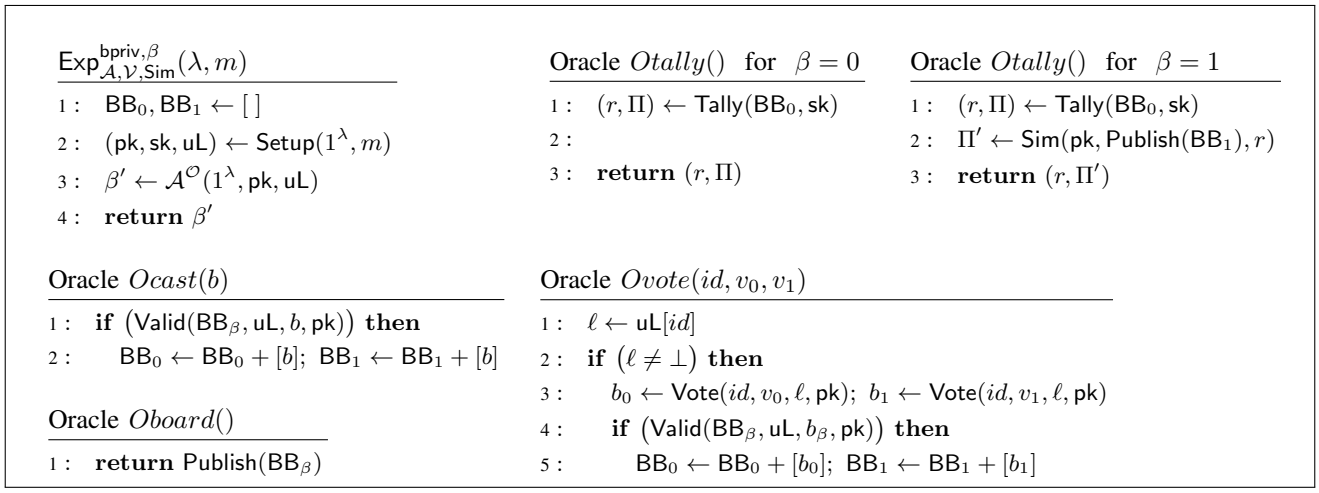

Figure 2. In the experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}$, the adversary $\mathcal{A}$ has access to the set of oracles $\mathcal{O} = \{Ocast, Ovote, Otally, Oboard\}$. The adversary is allowed to call the $Otally$ oracle at most once.

arbitrary ballots in the voting process. This idea is formalized via a game-based definition, that uses the experiment in Figure 2. The goal of the adversary is to distinguish between two worlds. In the process, the adversary chooses votes to be submitted by honest voters, and may submit ballots on behalf of the corrupt users. The adversary gets access to the bulletin board corresponding to the real world ($\beta = 0$) or the ideal world ($\beta = 1$), where fake votes are submitted by honest parties. The adversary always learns the real result, that is, the tally is always performed on BB_0. The result comes with the real proof of validity if $\beta = 0$ or a fake proofs when $\beta = 1$. This fake proof is created by an efficient simulator (who only has access to the visible part of the board). The adversary is expected to determine if it was playing with $\beta = 0$ or $\beta = 1$. Security demands that no adversary can tell the difference between the two world: so the board leaks no information about the content of the ballots, and neither does the proof that accompanies the result.

We capture the adversarial abilities using the formal definitions of oracles in Figure 2. We provide below informal descriptions of what these abilities represent.

$Ovote$: Receives two potential votes (v_0, v_1) for voter id. Then, using the label ℓ assigned to id (if such a label exists) it creates ballots b_0 from v_0 and b_1 from v_1. If ballot b_β is valid with respect to board BB_β, then b_0 is added to BB_0 and b_1 is added to BB_1;

$Ocast$: Lets the adversary cast a ballot on behalf of a corrupted voter;

$Oboard$: Limits the adversary's view of the bulletin board to its public version, given by the Publish algorithm. This public version of the board can vary very broadly, with the following variants–among others–being used in practice: the exact board, publish only the ciphertexts from the ballots, or the empty set.

$Otally$: Computes the result on board BB_0, and produces the proof of correct computation using the output of the tally algorithm for $\beta = 0$ or the proving simulator Sim for $\beta = 1$.

Definition 4 (Ballot Privacy [13])**.** 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 $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda, m)$ and $\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V},\mathsf{Sim}}(\lambda, m)$ defined in Figure 2. That is, the expression

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

is negligible in λ, for any $m \in \mathbb{N}$.

Intuitively, ballot privacy captures potential privacy breaches that may occur during the voting process. The notion does not account however for breaches that may occur during the tally procedure. It turns out that privacy of the tally phase can be enforced by demanding two additional security properties: strong correctness and strong consistency, as introduced in [13]. Together with ballot privacy, these two additional properties imply simulation-based notions of vote privacy [13].

STRONG CONSISTENCY. A voting scheme $\mathcal{V}$ is called *strongly consistent* [13], if the scheme ensures that its Tally algorithm behaves "as expected", i.e. it returns the result of applying the result function ρ to the votes underlying the (valid) ballots that are stored on a bulletin board. The following definition adapts the one of Bernhard et al. [13] to the slightly more general syntax that we adopt in this paper.

Definition 5 (Strong Consistency [13])**.** A voting scheme $\mathcal{V}$ is *strongly consistent* if there exists:

- An extraction algorithm $\mathsf{Extract}((id, \ell, c), \mathsf{sk})$ that takes as input a secret key sk and a ballot (id, ℓ, c), and outputs the id with either a vote in Vo or the special error symbol $\perp$; and
- A ballot validation algorithm $\mathsf{ValidInd}((id, \ell, c), \mathsf{pk})$ that returns true iff the ballot (id, ℓ, c) 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 $(\mathsf{pk}, \mathsf{sk}, \mathsf{uL})$ obtained from $\mathsf{Setup}(\lambda, m)$, and any (id, ℓ, v) if $b \leftarrow \mathsf{Vote}(id, v, \ell, \mathsf{pk})$ then $\mathsf{Extract}(b, \mathsf{sk})$ returns (id, v) with overwhelming probability.

2) For any adversarially produced (BB, b), if $\mathsf{Valid}(\mathsf{BB}, \mathsf{uL}, b, \mathsf{pk})$ 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 $\mathbf{Exp}^{\mathsf{consis}}_{\mathcal{B},\mathcal{V}}(\lambda, m)$ specified in Figure 3 returns true with a probability negligible in the security parameter.

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

1 : $(\mathsf{pk}, \mathsf{sk}, \mathsf{uL}) \leftarrow \mathsf{Setup}(1^\lambda, m)$
2 : $\mathsf{BB} \leftarrow \mathcal{B}(1^\lambda, \mathsf{pk}, \mathsf{uL})$
3 : $(r, \Pi) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$
4 : **for** i **in** $1..|\mathsf{BB}|$ **do**
5 : $\quad dbb[i] \leftarrow \mathsf{Extract}(\mathsf{BB}[i], \mathsf{sk})$
6 : $r' \leftarrow \rho(dbb)$
7 : **return** $(r \neq r')$

Figure 3. The Strong Consistency experiment

STRONG CORRECTNESS. This property requires that honestly created ballots will not be rejected by the validation algorithm; we demand that this property holds even with respect to an arbitrary ballot box chosen adversarially.

Definition 6 (Strong Correctness [13])**.** A voting scheme $\mathcal{V}$ is *strongly correct* if the advantage of any efficient adversary $\mathcal{B}$, defined by $\mathcal{A}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V}}(\lambda, m) = \Pr[\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V}}(\lambda, m) = 1]$ (where $\mathsf{Exp}^{\mathsf{corr}}_{\mathcal{B},\mathcal{V}}(\lambda, m)$ is defined in Figure 5) is negligible as a function of λ.

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

1 : $(\mathsf{pk}, \mathsf{sk}, \mathsf{uL}) \leftarrow \mathsf{Setup}(1^\lambda, m)$
2 : $(id, v, \mathsf{BB}) \leftarrow \mathcal{B}(1^\lambda, \mathsf{uL}, \mathsf{pk})$
3 : $\ell = \mathsf{uL}.[id]$
4 : $ev \leftarrow true$
5 : **if** $(\ell \neq \perp)$ **then**
6 : $\quad b \leftarrow \mathsf{Vote}(id, v, \ell, \mathsf{pk})$
7 : $\quad ev \leftarrow \mathsf{Valid}(\mathsf{BB}, \mathsf{uL}, b, \mathsf{pk})$
8 : **return** $\neg ev$

Figure 5. The Strong Correctness experiment

3. Labelled-MiniVoting

The MiniVoting scheme was introduced by Bernhard et al. [14] as an abstraction that captures several constructions in the literature. The key feature of this scheme is that it can be used to reason about the privacy properties of schemes in this class only from minimal assumptions on the cryptographic primitives.

In this paper we refine the scheme in two different ways. First, we enlarge the class of schemes that the scheme covers by introducing labels – public information associated to users, yielding the *Labelled-MiniVoting* scheme. Labels can be used to represent arbitrary information such as user's pseudonyms or their public verification keys. Second, since we attempt to carry out most of the proofs at the highest possible level of abstraction, we introduce some additional conditions on the algorithms in the definition of single-pass voting schemes. In particular, we demand that the algorithms that comprise the voting scheme are defined in terms of the following (abstract) algorithms, functions, and relations:

$\mathsf{ValidInd}(b, \mathsf{pk})$**:** Evaluates ballot b with respect to public key pk, and determines if the ballot is valid in isolation.

$\mathsf{Flabel}(id)$**:** Returns the label associated to an identity id.

$\rho((id_i, v_i)_i)$**:** Computes the election's result from a list of identities and votes.

$\mathcal{R}\big((\mathsf{pk}, pbb, r), (\mathsf{sk}, \mathsf{BB})\big)$**:** The relation enforced by the proof system. Instantiating the relation in different ways allows us to deal both with *verifiable* or *non-verifiable* voting schemes.

Usually, ρ is defined as $\rho(L) = \mathsf{Count} \circ \mathsf{Policy}(L)$ for two algorithms Count and Policy. $\mathsf{Policy}(L)$ filters the list L and selects which vote is counted for each voter in case multiple votes appear on the bulletin board. $\mathsf{Count}(L)$ computes the result from the filtered list of votes and may be probabilistic.

The following definition presents our generalization of the MiniVoting scheme, as a particular case of a single-pass voting scheme.

Definition 7. Let $\mathcal{E}$ be a labelled public-key encryption scheme, and $\Sigma_{\mathcal{R}} = (\mathsf{P}, \mathsf{V})$ be a proof system. Given algorithms $\mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}$ and ρ we define the *Labelled-MiniVoting* scheme

$$\mathcal{MV}(\mathcal{E}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho),$$

as the single-pass voting scheme defined by the algorithms in Figure 4, which we informally describe below.

$\mathsf{Setup}(1^\lambda, m)$**:** Generates a pair of keys $(\mathsf{pk}, \mathsf{sk})$ using the key generation algorithm of the encryption scheme $\mathcal{E}$, and creates a map uL where exactly m ids are assigned to labels.

$\mathsf{Vote}(id, \ell, v, \mathsf{pk})$**:** Constructs a ballot that contains the identity id of the voter, the label ℓ assigned to that voter, and a ciphertext obtained by the encryption algorithm $\mathcal{E}$ using the public key pk over the vote v and label ℓ.

$\mathsf{Valid}(\mathsf{BB}, \mathsf{uL}, (id, \ell, c), \mathsf{pk})$**:** Performs three checks to ensure that the ballot is well-formed with respect to the board. First, it checks that the ciphertext and label (ℓ, c) are unique with respect to the existing ballots in the ballot box BB. Then, it verifies the use of the correct label ℓ for user id using the map uL. Finally, it calls the ValidInd algorithm. Remark that this last step depends

$\mathsf{Setup}(1^\lambda, m)$

1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
2 : **for** i **in** $1..m$ **do**
3 : $\quad id \leftarrow_\$ \mathsf{ID}$
4 : $\quad \mathsf{uL}[id] \leftarrow \mathsf{Flabel}(id)$
5 : **return** $(\mathsf{pk}, \mathsf{sk}, \mathsf{uL})$

$\mathsf{Vote}(id, \ell, v, \mathsf{pk})$

1 : $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \ell, v)$
2 : **return** (id, ℓ, c)

$\mathsf{Valid}(\mathsf{BB}, \mathsf{uL}, b, \mathsf{pk})$

1 : $(id, \ell, c) \leftarrow b$
2 : $e_1 \leftarrow \forall id'.\ (id', \ell, c) \notin \mathsf{BB}$
3 : $e_2 \leftarrow \big(\ell = \mathsf{uL}[id]\big)$
4 : $e_3 \leftarrow \mathsf{ValidInd}(b, \mathsf{pk})$
5 : **return** $(e_1 \wedge e_2 \wedge e_3)$

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$

1 : $dbb = [\,]$
2 : **for** i **in** $1..|\mathsf{BB}|$ **do**
3 : $\quad (id, \ell, c) = \mathsf{BB}[i]$
4 : $\quad dbb[i] \leftarrow (id, \mathsf{Dec}(\mathsf{sk}, \ell, c))$
5 : $r \leftarrow \rho(dbb)$
6 : $pbb \leftarrow \mathsf{Publish}(\mathsf{BB})$
7 : $\Pi \leftarrow \mathsf{P}((\mathsf{pk}, pbb, r), (\mathsf{sk}, \mathsf{BB}))$
8 : **return** (r, Π)

Figure 4. Algorithms defining the *Labelled-MiniVoting* scheme

on an abstract algorithm that might be instantiated to not perform any additional checks.

$\mathsf{Publish}(\mathsf{BB})$**:** Publishes (arbitrary) information about the content of the ballot box.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$**:** Computes the result of the voting process in two steps: first it decrypts the entire board, and then it applies a result function ρ (that might or might not filter some of the votes based on a predefined voting policy, for example: keep the last valid vote for any id). Additionally, it provides a proof that the tally was done correctly by calling the prover P.

$\mathsf{Verify}((\mathsf{pk}, pbb, r), \Pi)$**:** Calls the verify algorithm of the verifier V, to show that the tally was done correctly.

VOTING FRIENDLY RELATION. Since we keep the relation $\mathcal{R}$ that is used to certify the tally procedure abstract, we need to ensure that the relation $\mathcal{R}$ is compatible with the result of the election (computed with ρ). That is, we require that if the result r corresponds to the votes obtained by decrypting the ballot box BB using the key sk, if pbb corresponds to the public board of BB and pk is the public key associated to sk then r can be proved to be the correct result, that is, $\mathcal{R}((\mathsf{pk}, pbb, r), (\mathsf{sk}, \mathsf{BB}))$ holds.[3] Note that this notion is unrelated to *voting friendly encryption* as defined in [14], which designates a class of IND-1-CCA encryption schemes that have embedded a homomorphic part.

The formal definition of a voting-friendly relation, uses the following convenient notation. Given a bulletin board $\mathsf{BB} = [(id_1, \ell_1, c_1), \ldots, (id_n, \ell_n, c_n)]$ (seen as an ordered list of bulletin board entries), we write Dec^* for the algorithm that decrypts each line and returns the ordered list of plaintexts:

$$\mathsf{Dec}^*(\mathsf{sk}, \mathsf{BB}) = \big((id_1, \mathsf{Dec}(\mathsf{sk}, \ell_1, c_1)), \ldots, (id_n, \mathsf{Dec}(\mathsf{sk}, \ell_n, c_n))\big)$$

Definition 8. Let $\mathcal{E}$ be a labelled public-key encryption scheme, and $\Sigma_\mathcal{R}$ be a proof system for some relation $\mathcal{R}$. Given the abstract algorithms ρ and Publish, we say that $\mathcal{R}$ is a *voting-friendly relation* with respect to ρ and Publish, if for any efficient adversary $\mathcal{B}$, the following experiment returns 1 with negligible probability.

$\mathbf{Exp}^{\mathsf{vfr}}_{\mathcal{B},\mathcal{E},\Sigma_\mathcal{R},\rho,\mathsf{Publish}}(\lambda)$

1 : $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
2 : $\mathsf{BB} \leftarrow \mathcal{B}(1^\lambda, \mathsf{pk})$
3 : $dbb \leftarrow \mathsf{Dec}^*(\mathsf{sk}, \mathsf{BB})$
4 : $r \leftarrow \rho(dbb)$
5 : $pbb \leftarrow \mathsf{Publish}(\mathsf{BB})$
6 : **return** $\neg\, \mathcal{R}\big((\mathsf{pk}, pbb, r), (\mathsf{sk}, \mathsf{BB})\big)$

We provide machine-checked proofs [1] that the *Labelled-MiniVoting* scheme is ballot private, strongly consistent and strongly correct.

Strong correctness is implied by the IND-1-CCA security assumption of the encryption scheme: any p.p.t. adversary can only obtain a collision with already-produced ciphertexts when making an encryption query with negligible probability.

Theorem 1 (strong correctness)**.** [4] *Let* $\mathcal{V} = \mathcal{MV}(\mathcal{E}, \Sigma_\mathcal{R}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho)$ *with* $\mathsf{ValidInd}((id, \ell, c), \mathsf{pk})$ *true for* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \ell, v)$*, and any* pk, ℓ, v, id*. For any p.p.t. adversary* $\mathcal{A}$*, there exists an adversary* $\mathcal{B}$ *such that:*

$$\mathcal{A}^{\mathsf{corr}}_{\mathcal{A},\mathcal{V}}(\lambda, m) \leq \mathsf{Adv}^{\mathsf{poly\text{-}ind1cca}}_{\mathcal{B},\mathcal{E},1}(\lambda),$$

for any m *voters.*

The next theorem establishes that the *Labelled-MiniVoting* scheme satisfies ballot privacy, under standard assumptions on its components. One quirk that is worth remarking is that we actually prove a slightly stronger statement which requires the weaker hypothesis that the underlying encryption scheme is strongly correct. For simplicity, we use the above theorem and rely on the non-malleability of the underlying encryption scheme which, in turn, implies strong correctness.

Theorem 2 (ballot privacy)**.** [5] *Let* $\mathcal{V} = \mathcal{MV}(\mathcal{E}, \Sigma_\mathcal{R}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho)$ *with* $\mathsf{ValidInd}((id, \ell, c), \mathsf{pk})$ *true for* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, \ell, v)$*, and any* pk, ℓ, v, id*. For any* m *voters, and any adversary* $\mathcal{A}$ *that makes at most* n *voting*

3. We do not put any restrictions on the relation $\mathcal{R}$; in particular $\mathcal{R}$ may depend on some random oracle.

4. Lemma scorr in ../MiniVotingSecurity.ec

5. Lemma bpriv in ../MiniVotingSecurity.ec

queries, there exists a simulator S and three adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ such that:

$$\mathsf{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V},\mathsf{S}}(\lambda, m) \leq 2 \times \Pr[\mathsf{Exp}^{\mathsf{vfr}}_{\mathcal{D},\mathcal{E},\Sigma_{\mathcal{R}},\rho,\mathsf{Publish}}(\lambda) = 1] + \mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B},\mathsf{P},\mathsf{S},\mathcal{R}}(\lambda) + 3n \times \mathsf{Adv}^{\mathsf{poly\text{-}ind1cca}}_{\mathcal{C},\mathcal{E},1}(\lambda).$$

The intuition for the ballot privacy component of the above theorem is based on two key points. First, one can replace with negligible loss of security a proof from a zero-knowledge proof system with a simulated one, provided that the relation is (with overwhelming probability) satisfied by a ballot privacy adversary. At this stage of the proof the relation is left unspecified, thus bounded by a voting friendly constraint. Secondly, one uses the IND-1-CCA security assumption for the encryption scheme to replace the view of the adversary on the ballot box.

Theorem 3. *(strong consistency)* [6] *Let $\mathcal{V} = \mathcal{MV}(\mathcal{E}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho)$. Then, $\mathcal{V}$ is strongly consistent.*

The proof for strong consistency easily follows from the definition of the *Labelled-MiniVoting* scheme, using the extractor defined by

$$\forall\, id, \ell, c.\ \mathsf{Extract}((id, \ell, c), \mathsf{sk}) = (id, \mathsf{Dec}(\mathsf{sk}, \ell, c)).$$

4. Applications

We now show how to apply the general results discussed in Section 3 to the security of several hundred variants of Helios, including most of its existing versions–either implemented, or mentioned in publications. We first instantiate *Labelled-MiniVoting* to a scheme we call *PreHelios*, instantiating the labelled public-key encryption scheme with a construction that combines a public-key encryption scheme with a proof of knowledge. *PreHelios* then serves as a basis for further instantiations and refinements.

In particular, we show that *PreHelios* corresponds, assuming secure mixnets, to the variant *Helios-mix* of Helios with mixnets. We further prove that the security of *PreHelios* is equivalent, assuming a secure homomorphic encryption scheme, to that of a variant *Helios-hom*, which uses homomorphic tally. In both cases, we show that remaining parameters of *Helios-mix* and *Helios-hom* (for example, validity check or result functions ...) can be instantiated in many ways, yielding about 540 secure variants each, all equipped with a machine-checked proof automatically based on our framework theorem. *Labelled-MiniVoting* requires–quite strictly–that the ballot box carefully discards duplicate ciphertexts. We also explain how to (securely) relax this condition, yielding another variant *Helios-hom-IDweed* that only discards duplicate ciphertexts when they correspond to the same ID. This check is more practical since it can easily done while deadling with revoting. Finally, we point to several implemented voting schemes, whose privacy properties are directly captured as machine-checked instantiations of one of our Theorems.

Our results are summarized in Figure 6. We equip all the schemes we discuss with machine-checked proofs of privacy. Rectangular nodes are not fully instantiated, and represent families of schemes whose privacy may rely on some non-standard hypotheses. All leaves represent fully-instantiated schemes: their security relies only on that of their cryptographic primitives (encryption scheme and proof systems).

4.1. PreHelios

Helios constructs its labelled public-key encryption scheme by composing El Gamal encryption with a proof of knowledge.[7] Let us consider a labelled public-key encryption scheme $\mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$, built from a public-key encryption scheme $\mathcal{E}'$ and a proof system $\Sigma'_{\mathcal{R}'}$. The proof system typically proves validity of the vote v (for example, that at most one candidate is selected by the ballot), and may use a label (such as the voter's identity) inside the statement. The encryption algorithm returns a ciphertext produced by $\mathcal{E}'$ on v, together with a proof of validity π that links the ciphertext, public key and label to the underlying vote and randomness used during encryption. The decryption algorithm checks the validity of the proof before decrypting the ballot. A more formal description of the construction is given in Figure 7.

$\mathsf{KGen}(1^\lambda)$

1 : **return** $\mathsf{KGen}'(1^\lambda)$

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

1 : $r \leftarrow \mathbb{Z}_q$
2 : $c \leftarrow \mathsf{Enc}'(\mathsf{pk}, m; r)$
3 : $\pi \leftarrow \mathsf{P}'((c, \mathsf{pk}, \ell), (m, r))$
4 : **return** (c, π)

$\mathsf{Dec}(\mathsf{sk}, \ell, (c, \pi))$

1 : **if** $\mathsf{V}'((c, \mathsf{pk}, \ell), \pi)$ **then**
2 : **return** $\mathsf{Dec}'(\mathsf{sk}, c)$
3 : **return** $\perp$

Figure 7. Algorithms of $\mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$, with $\Sigma'_{\mathcal{R}'} = (\mathsf{P}', \mathsf{V}')$.

We define *PreHelios* as *Labelled-MiniVoting* instantiated with an LPKE construction as its labelled public-key encryption scheme.

Definition 9. Let $\mathcal{E}'$ be an encryption scheme, and $\Sigma_{\mathcal{R}}$, $\Sigma'_{\mathcal{R}'}$ be two proof systems. Given $\Gamma = \mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$, and some algorithms ValidInd, Publish, Flabel and ρ, the PreHelios *scheme* constructed from these primitives is defined as

$$\mathcal{PH}(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho) = \mathcal{MV}(\Gamma, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho).$$

The following corollary of Theorems 1, 2, 3, states that *PreHelios* inherits the security of *Labelled-MiniVoting*.

6. Lemmas consis1, consis2, consis3 in ../MiniVotingSecurity.ec

7. Even for the mixnet variant of Helios, El Gamal is used in conjunction with a proof of knowledge – of the randomness used in mixing rather than the plaintext – in order to obtain an IND-1-CCA scheme.

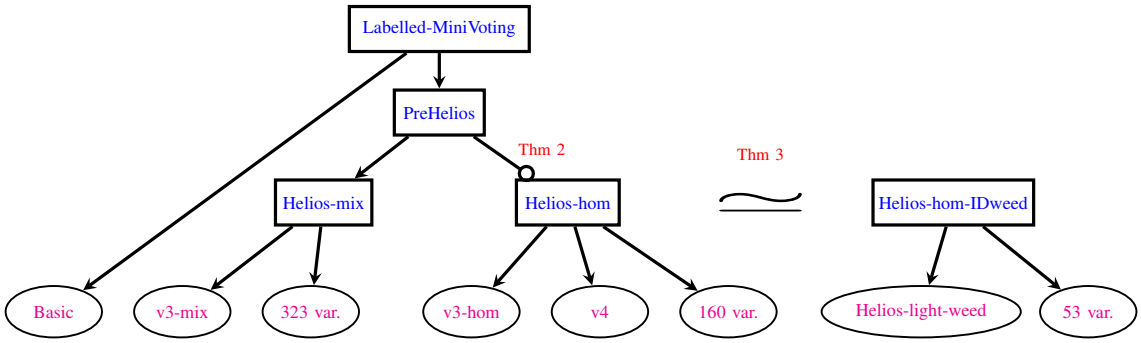

Figure 6. Relations between our schemes. Arrows represent direct instantiations, o-arrows represent instantiations (where some equivalence property is used); $\simeq$ is observational equivalence. The leafs contain either concrete instances (e.g. Basic, (Helios) v3-mix), or the number of variants that have been obtained. All constructions satisfy ballot privacy, strong consistency, strong correctness (under some assumptions).

Corollary 1. *Let $\mathcal{E}'$ be an encryption scheme, and $\Sigma_{\mathcal{R}}$, $\Sigma'_{\mathcal{R}'}$ be two proof systems. Let $\Gamma = \mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$, and* ValidInd *that returns true for valid ciphertexts. The scheme $\mathcal{PH}(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho)$ is*

- *ballot private, provided that $\mathcal{R}$ is voting friendly, Γ is* IND-1-CCA*, the proof system $\Sigma_{\mathcal{R}}$ is zero-knowledge.*
- *strongly consistent.*
- *strongly correct if Γ is* IND-1-CCA

This follows directly from Theorems 1, 2, 3. A typical example of an IND-1-CCA encryption scheme is El Gamal encryption with Chaum-Pedersen proofs [15].

4.2. Security by Refinement

However, it should be clear from the definition of *Labelled-MiniVoting* that only the most basic voting systems can be produced purely by instantiating *PreHelios* further. Indeed, *Labelled-MiniVoting* specifies an ideal Tally algorithm that decrypts the ballot box line-by-line before computing the election result using ρ, whereas any voting system that means to provide resilience against corrupted tallying servers cannot follow this course of action.

To support the application of our Theorem 2 to such schemes–capturing in particular most published Helios variants, we show that privacy properties are preserved when substituting a functionally equivalent algorithm for Tally. We now define the necessary notions of functional equivalence and algorithm substitution.

Definition 10 (Functional Equivalence)**.** Let f be a (stateful) algorithm. We write

$$\Pr[f(e), m \rightsquigarrow (r, m_r)]$$

for the probability that the execution of $f(e)$ leads to the final memory m_r with result r when executed in the initial memory m. Given a predicate ϕ over inputs and memories, two procedures f_1 and f_2 are *functionally equivalent under* ϕ, written $f_1 \simeq_\phi f_2$, iff

$$\Pr[f_1(e), m \rightsquigarrow (r, m_r)] = \Pr[f_2(e), m \rightsquigarrow (r, m_r)]$$

for any input value e, output value r and memories m, m_r such that $\phi(e, m)$. For the constantly true predicate T, $\simeq_{\mathsf{T}}$ expresses *unconditional* functional equivalence, which we simply denote with $\simeq$.

We note that this notion of equivalence captures algorithms that are both probabilistic and stateful, intuitively requiring that equivalent algorithms produce the same joint distributions on outputs and final state given the same inputs and initial state.

Given a voting scheme $\mathcal{V} = (\mathsf{S}, \mathsf{Vo}, \mathsf{Va}, \mathsf{P}, \mathsf{T}, \mathsf{Ve})$ and an alternative tallying algorithm Tally', we define *the variant of scheme $\mathcal{V}$ that uses algorithm* Tally' *for tallying* as $\mathcal{V}[\mathsf{T} \leftarrow \mathsf{Tally}'] = (\mathsf{S}, \mathsf{Vo}, \mathsf{Va}, \mathsf{P}, \mathsf{Tally}', \mathsf{Ve})$. Similarly, given an alternative validation algorithm Valid', we define *the variant of scheme $\mathcal{V}$ that uses algorithm* Valid' *for ballot validation* as $\mathcal{V}[\mathsf{Va} \leftarrow \mathsf{Valid}'] = (\mathsf{S}, \mathsf{Vo}, \mathsf{Valid}', \mathsf{P}, \mathsf{T}, \mathsf{Ve})$. When the nature of the alternative algorithm is clear from context, we simply write $\mathcal{V}[\mathsf{Tally}']$ or $\mathcal{V}[\mathsf{Valid}']$ for the relevant algorithm substitution.

Lemma 2 (Tally-Equivalence preserves Privacy)**.** *Given a voting scheme $\mathcal{V}$ with tallying algorithm* Tally*, and some alternative tallying algorithm* Tally' *such that* $\mathsf{Tally} \simeq \mathsf{Tally}'$*. If $\mathcal{V}$ is ballot private (resp. strongly correct; strongly consistent) then $\mathcal{V}[\mathsf{Tally}']$ is ballot private (resp. strongly correct; strongly consistent).*

Proof. The theorem is a simple consequence of the definitions for the three properties. We note in particular, that, for $\beta \in \{0, 1\}$, $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}}$ and $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V}[\mathsf{Tally}']}$ are strictly equivalent until the adversary queries its $Otally$ oracle, so the corresponding queries to Tally and Tally' necessarily occur in a pair of contexts where inputs and states are equal. A similar observation applies for strong consistency. Finally, tallying is not used for strong correctness. □

4.3. Helios-mix

Mixnets were introduced by Chaum [19] as a method to implement anonymous communications in the absence of a trusted authority. A mixnet takes as input a set of encrypted messages and outputs the underlying plaintexts in a way that hides the relation between the input ciphertexts and the output plaintexts. Interest in their applications (which go

significantly beyond electronic voting) resulted in a large body of literature that covers different constructions [30], [32], [36], [37], [41], and security models and proofs for some of the constructions [32], [33], [41].

Concrete implementations typically "chain" several mixers so that the output of one is passed as input to the next. Each intermediate shuffle comes with a proof that mixing was implemented correctly. Since we are not concerned here with dishonest tally authorities, we simply view mixnets as an abstract algorithm TallyMix which, given a list of ballots and the secret key, returns their decryptions in some order together with a proof that the list of decrypted votes corresponds to the initial list of ballots. Existing mixnet constructions return the plaintexts either in random order [36] or in lexicographic order [32].

The following definition fixes the class of protocols obtained by replacing the tally algorithm with a mixnet, whose operations are modelled as a probabilistic algorithm TallyMix. In our definition, we make use of the Count function multiset that can be instantiated with functions that return the sequence of votes in lexicographic or random order.

Definition 11. Let $\mathcal{E}'$ be an encryption scheme, $\Sigma'_{\mathcal{R}'}$ and $\Sigma_{\mathcal{R}}$ be two proof systems, ValidInd, Publish, Flabel, and Policy be abstract algorithms as specified, and TallyMix be a mixnet functionality. Given $\rho = \mathsf{multiset} \circ \mathsf{Policy}$, we define the Helios-mix *scheme* constructed from these primitives as

$$\mathcal{HM}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \mathsf{Policy}\right) = \mathcal{PH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho\right)[\mathsf{TallyMix}].$$

Based on this definition, if one can find an instance $\mathcal{PH}$ of *PreHelios* whose Tally algorithm is such that TallyMix $\simeq$ Tally, the following corollary identifies sufficient conditions for the privacy of the *Helios-mix* variants constructed from the same primitives.

Corollary 2. *Let $\mathcal{E}'$ be an encryption scheme, $\Sigma_{\mathcal{R}}$, $\Sigma'_{\mathcal{R}'}$ be two proof systems, $\Gamma = \mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$, and* ValidInd*,* Publish*,* Flabel*, and* Policy *be abstract algorithms as specified. The scheme*

$$\mathcal{HM}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \mathsf{Policy}\right)$$

is ballot private, strongly consistent, and strongly correct provided that i. TallyMix $\simeq$ Tally *(with the specified primitives); ii.* ValidInd *returns true for valid ciphertexts; iii. $\mathcal{R}$ is voting-friendly; iv. Γ is* IND-1-CCA*; and v. $\Sigma_{\mathcal{R}}$ is zero-knowledge.*

Our EasyCrypt formalization of the result supports two separate instantiations for multiset, using either lexicographic ordering (as *Helios-mix-ord*), or random ordering (as *Helios-mix-perm*). In particular, we formally prove the required functional equivalences for Tally and TallyMix.

4.4. Helios-hom

Similarly, *PreHelios* can be refined in a similar way to prove privacy properties for homomorphic variants of Helios.

The Helios-hom *scheme* is defined as an instantiation of a *PreHelios* scheme with some homomorphic public key encryption scheme $\mathcal{E}'$, and whose Tally algorithm is modified to be as shown in Figure 8.

TallyHom(BB, sk)

1 : $sbb \leftarrow$ valid ballots based on proof check from BB
2 : $fbb \leftarrow \mathsf{Policy}(sbb)$
3 : $c \leftarrow \mathsf{Add}(fbb)$
4 : $r \leftarrow \mathsf{Dec}(c, \mathsf{sk})$
5 : $pbb \leftarrow \mathsf{Publish}(\mathsf{BB})$
6 : $\Pi \leftarrow \mathsf{P}((\mathsf{pk}, pbb, r), (\mathsf{sk}, \mathsf{BB}))$
7 : **return** (r, Π)

Figure 8. *Helios-hom* TallyHom algorithm

Definition 12. Let $\mathcal{E}'$ be a public-key encryption scheme, Add be a candidate homomorphic operation, $\Sigma_{\mathcal{R}}$ and $\Sigma'_{\mathcal{R}'}$ be two proof systems, and ValidInd, Publish, Flabel, and Policy be abstract algorithms as specified. Given $\rho = \mathsf{addition} \circ \mathsf{Policy}$, we define the Helios-hom *scheme* constructed from these primitives as:

$$\mathcal{HH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho\right) = \mathcal{PH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho\right)[\mathsf{TallyHom}].$$

Theorem 4. *Let $\mathcal{E}'$ be a homomorphic encryption scheme, $\Sigma'_{\mathcal{R}'}, \Sigma'_{\mathcal{R}'}$ be two proof systems, and* ValidInd*,* Publish*,* Flabel*,* Policy *be abstract algorithms as specified such that:*

$$\forall\, b, \mathsf{BB}.\ b \in \mathsf{Policy}(\mathsf{BB}) \implies b \in \mathsf{BB}\textit{; and}$$
$$\forall\, \mathsf{sk}, \mathsf{BB}.\ \mathsf{Dec}^*(\mathsf{sk}, \mathsf{Policy}(\mathsf{BB})) = \mathsf{Policy}(\mathsf{Dec}^*(\mathsf{sk}, \mathsf{BB})).$$

Let TallyHom *be the tallying algorithm of the scheme $\mathcal{HH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \mathsf{Policy}\right)$, and* Tally *be the tallying algorithm defined by the scheme $\mathcal{PH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \rho\right)$ with $\rho = \mathsf{addition} \circ \mathsf{Policy}$. We have* TallyHom $\simeq$ Tally.

Theorem 4 is proved in EasyCrypt. The equivalence follows from the homomorphic property of the encryption scheme and the commutativity and membership properties of the Policy algorithm. We then easily deduce privacy properties of *Helios-hom*, via Lemma 2.

Corollary 3. *Let $\mathcal{E}'$ be an encryption scheme, with* Add *a candidate homomorphic operation for $\mathcal{E}'$, let $\Sigma_{\mathcal{R}}$, $\Sigma'_{\mathcal{R}'}$ be two proof systems, $\Gamma = \mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$, and* ValidInd*,* Publish*,* Flabel*, and* Policy *be abstract algorithms as specified. The scheme*

$$\mathcal{HH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \mathsf{Policy}\right),$$

is ballot private, strongly consistent, and strongly correct provided that i. $\mathcal{R}$ is voting friendly; ii. Γ is IND-1-CCA*; iii. $\Sigma_{\mathcal{R}}$ is zero-knowledge; iv. $\mathcal{E}'$ and* Add *form a homomorphic public-key encryption scheme; v.* ValidInd *returns*

true for valid ciphertexts; vi. Policy *commutes with* Dec^**; vii.* $\forall\, b, \mathsf{BB}.\ b \in \mathsf{Policy}(\mathsf{BB}) \implies b \in \mathsf{BB}$.

4.5. Various realizations of Helios

We recall that *Labelled-MiniVoting* is parameterized by a labelled public-key encryption scheme, a proof system, and six abstract algorithms.

$$\begin{array}{l}\mathsf{ValidInd} : ((\mathsf{ID}, \mathsf{L}, \mathsf{C}), \mathsf{PK}) \to \{0,1\},\\ \mathsf{Publish} : (\mathsf{ID}, \mathsf{L}, \mathsf{C})^* \to \mathsf{PBB},\\ \mathsf{Flabel} : \mathsf{ID} \to \mathsf{L},\\ \mathsf{Count} : (\mathsf{ID}, \mathsf{Vo}_\perp)^* \to \mathsf{R},\\ \mathsf{Policy} : (\mathsf{ID}, \mathsf{Vo}_\perp)^* \to (\mathsf{ID}, \mathsf{Vo}_\perp)^*,\\ \mathcal{R} : \big((\mathsf{PK}, \mathsf{PBB}, \mathsf{R}), (\mathsf{SK}, (\mathsf{ID}, \mathsf{L}, \mathsf{C})^*)\big) \to \{0,1\}\end{array}$$

To illustrate the versatility of our framework, we list some interesting instances of these algorithms.

Typical choices for ValidInd include the constantly true function, which lets the ballot box accept all non-duplicated ballots, and algorithm V' itself, which checks that the ballot is equipped with a valid proof before accepting it.

Many choices for Publish have been considered, from trivial publication policies that reveal no information about the ballot box (empty), or publish the ballot box itself (identity) to more involved publication algorithms that reveal the last cast ballot for each voter, with (last view) or without the id (anonymous-view), or reveal a hash of each entry along with the entry itself (hash-view). Further, entries could be reordered, or dummy entries inserted before publication of the bulleting board.

Algorithm Flabel, which produces the label from the voter id, is usually instantiated trivially, either as a constant function (constant), or as the identity function (identity). Other interesting choices would include pseudo-random labels (where, for example, the label is produced during setup using a PRP whose key is discarded afterwards) that could serve as pseudonyms.

The revote policy Policy can either enforce a *single vote* per id or take *multiple votes* into account, for example where votes can be bought or are linked to some form of stake. Single vote policies can be as simple as choosing the last (last vote) or first vote cast by each voter, and as complex as those used in Estonia (where priority is given to ballots cast in person, over electronic ballots). Multiple vote policies could sum all votes, or average all votes cast by each voter before computing the final result.

The relation $\mathcal{R}$ proved by the tallying authorities is usually instantiated either as the trivial true relation, or as the relation that relates plaintexts and associated datas to all valid ciphertexts that encrypt them (corr-dec).

Lastly, choices for the counting algorithm Count are as numerous as there are types of democracy and elections. We list the most common ones.

- addition. Tells how many votes each candidate received.
- multiset. Returns the sequence of all votes. Two main categories are considered:
  - order. All votes are given in lexicographic order.
  - permutation. Votes are returned in a random order.
- majority. Discloses only the winner.
- weighted. Some voters may have a more important role in making the decision.
- condorcet. Voters rank candidates in order of preference. A single winner is selected by conducting a series of fictitious one-on-one elections between every possible pairing of two candidates.
- STV (Single Transferable Vote). Voters rank the candidates in order of preference. Votes for candidates above some threshold are transfered to other candidates based on the order of preference indicated on the ballots.

Hundreds of secure variants of *Labelled-MiniVoting*. These options can be combined arbitrarily, subject to a few constraints, that are imposed in particular by voting-friendliness and other non-cryptographic premises.

- If $\mathsf{Publish} : (\mathsf{ID}, \mathsf{L}, \mathsf{C})^* \to \mathsf{PBB}$ returns an empty bulletin board then $\mathcal{R}$ must be true.
- For *Helios-hom*, the counting function has to be addition and the policy cannot be average.
- For *Helios-hom-IDweed*, Flabel must yield an injective mapping from identities to labels, the counting function has to be addition, and the policy cannot be average.
- For *Helios-mix*, the counting function has to be multiset, and the policy cannot be average.

This yields 162 variants for *Helios-hom*, 54 variants for *Helios-hom-IDweed* and 324 variants for *Helios-mix*. All these variants were automatically generated and equipped with (checked) EasyCrypt proofs of ballot privacy, strong consistency, and strong correctness in less than 31 minutes overall. An overview is provided in Table 2. To avoid generating all the variants anytime a single instance is desired, we also provide a simple design interface where the user selects the options of her choice for each parameter. The resulting voting scheme is then automatically generated along with a proof of its security, checked with EasyCrypt. We note in particular that, once all algorithms are fully instantiated, the non-cryptographic premises of our security theorems and corollaries are automatically discharged. This includes the voting-friendliness properties of $\mathcal{R}$ w.r.t. the other algorithms for all instances, the commutativity and regularity properties of Policy for homomorphic instances, and the injectivity property of Flabel for instances with reduced-weeding.

Insecure variants of *Labelled-MiniVoting*. *Labelled-MiniVoting* has been designed to ensure that ballot privacy, together with strong consistency and strong correctness, are satisfied with only (minimal) assumptions on the cryptographic primitives. Many features and restrictions have been hard-coded (in the algorithms or the cryptographic primitives) such that it should be difficult to find instantiations of *Labelled-MiniVoting* that are not ballot private.

One such feature is the weeding process modeled by the Valid algorithm. It has been well documented [15], [23], [24] that not weeding ballots carefully leads to insecure schemes. In particular, this is also why Helios as implemented in

versions 3 and 4 do not satisfy ballot privacy. For the Helios variants that we analyze weeding is encapsulated in the Valid algorithm.

4.6. Existing Variants

Combining the results obtained in the previous sections yields the security of several hundred variants of electronic voting schemes. In this section, we point to some variants that correspond to existing published schemes.

We consider more specific cryptographic primitives. Let $\mathcal{E}'$ be the exponential El Gamal encryption scheme [16], [29], and $\Sigma'_{\mathcal{R}'} = (\mathsf{P}', \mathsf{V}')$ the disjunctive Chaum-Pedersen proof system [18][8] over the relation $\mathcal{R}'$ that ensures that a vote satisfies the requirements of the election. Additionally, let $\Sigma_{\mathsf{corr\text{-}dec}}$ be the Chaum-Pedersen proof system for correct decryption. We assume the El Gamal encryption scheme with Chaum-Pedersen proofs of knowledge (formally defined as $\Gamma = \mathsf{LPKE}(\mathcal{E}', \Sigma_{\mathcal{R}'})$) to be IND-1-CCA (a pen-and-paper proof is given in [15]). We further assume that the correct decryption proof system $\Sigma_{\mathsf{corr\text{-}dec}}$ is zero-knowledge. We can then deduce the following results on the practical schemes listed in Table 1.

HELIOS VERSION 3 WITH HOMOMORPHIC TALLY. *Helios version 3 with homomorphic tally* [2] (Helios-v3-hom, for short) corresponds to *Helios-hom* instantiated with exponential El Gamal encryption, last vote policy, no label, and addition as counting mode. Formally, Helios version 3 with homomorphic tally is defined as

$$\textit{Helios-v3-hom} = \mathcal{HH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathsf{corr\text{-}dec}}, \mathsf{ValidInd}_{\mathsf{V}'}, \mathsf{Publish}_{\mathsf{last\ view}}, \mathsf{Flabel}_{\mathsf{empty}}, \mathsf{Policy}_{\mathsf{last\ vote}}\right).$$

By setting the identity of the voters to be either true names or aliases, we can cover here two sub-variants of Helios version 3:

Helios-v3-hom with true identities. This version has been used since the introduction of Helios in version 1 [2].

Helios-v3-hom with aliases. Version that was initially introduced for the 2009 election at Louvain in Helios version 2 [3], and was later made more broadly available in Helios version 3.

HELIOS VERSION 3 WITH MIXNETS. *Helios version 3 with mixnets* [2] (called Helios-v3-mix, for short) corresponds to *Helios-mix* instantiated with last vote policy, no labels, and the multiset counting mode. Formally, Helios version 3 with mixnets is defined as

$$\textit{Helios-v3-mix} = \mathcal{HM}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathsf{corr\text{-}dec}}, \mathsf{ValidInd}_{\mathsf{V}'}, \mathsf{Publish}_{\mathsf{last\ view}}, \mathsf{Flabel}_{\mathsf{empty}}, \mathsf{Policy}_{\mathsf{last\ vote}}\right).$$

Like homomorphic Helios-v3, this variant supports both elections where the name of voters is in the clear, or election with aliases.

HELIOS VERSION 4. This version of Helios is closely related to Helios version 3 with homomorphic tally. We point out some small but important differences.

8. We use the strong Fiat-Shamir transformation.

First, Helios version 4 uses more robust proofs of knowledge, that contain additional information such as election hash, or question number (that must therefore appear in the label). Furthermore, to ease the readability of the ballot by voters, Helios version 4 applies a hash over the ballot and publishes it along with the ballot. Lastly, Helios v4 does not support any mixnet-based variants, and is solely based on homomorphic encryption. Therefore, algorithms ValidInd, ρ and $\mathcal{R}$ are identical to Helios version 3 with homomorphic tally, Publish produces a hash for each bulletin board entry that is published alongside it, and Flabel is the constant function returning election hash, question numbers and choice numbers.

The *Helios version 4* [2] can formally be defined as

$$\textit{Helios-v4} = \mathcal{HH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathsf{corr\text{-}dec}}\right), \mathsf{ValidInd}_{\mathsf{V}'}, \mathsf{Publish}_{\mathsf{hash\text{-}view}}, \mathsf{Flabel}_{\mathsf{constant}}, \mathsf{Policy}_{\mathsf{last\ vote}}\right).$$

Note that our proofs do not cover Helios version 1 or version 2, since their underlying encryption scheme, based on the weak Fiat-Shamir transformation, is not IND-1-CCA. Replacing the weak Fiat-Shamir transformation with the strong Fiat-Shamir transformation in their protocol descriptions would yield *Helios-v3-mix* and *Helios-v3*, respectively, although particulars of the primitives differ.

BASIC ELECTION SCHEME. One of the most basic election schemes consists in sending votes encrypted with a public key to a (trusted) voting server. No revote is allowed. During tally, the server simply shuffles the ballots and decrypts them line by line. Of course, this does not offer any verifiability. Such a basic election scheme is at the core of several simple commercial voting systems currently in use.

Let $\mathcal{E}$ be a labelled public-key encryption scheme with an empty label, and $\Sigma_{\mathsf{true}} = (\mathsf{P}, \mathsf{V}_{\mathsf{true}})$ a proof system where both the verifier $\mathsf{V}_{\mathsf{true}}$ and relation are constantly true.

This basic voting scheme is formally defined as

$$\mathcal{MV}\left(\mathcal{E}, \Sigma_{\mathsf{true}}, \mathsf{ValidInd}_{\mathsf{true}}, \mathsf{Publish}_{\mathsf{no\text{-}revote}}, \mathsf{Flabel}_{\mathsf{empty}}, \rho_{\mathsf{addition} \circ \mathsf{no\text{-}revote}}\right).$$

4.7. Weeding

Cortier et al [23], [24] and Bernhard et al. [15] have shown the need for weeding in the context of ballot privacy. For example, if an adversary may copy the ballot of an honest voter (typically available on the bulletin board) and re-cast it on his own behalf, he obtains some information on the vote once the result is published, hence breaking privacy. In particular, the ballot privacy of *Labelled-MiniVoting* is based on a strong weeding policy, preventing pairs of ciphertexts and labels being replayed even with different voter identifiers, being enforced by the Valid algorithm.

We show that it is possible to weaken the weeding policy by weeding only exact ballot duplicates. More precisely, we show that instead of rejecting a ballot (id, ℓ, c) as soon as (id', ℓ, c) occurs in BB for some id', we only reject it if exactly (id, ℓ, c) occurs in BB. This may speed up the weeding algorithm since this latter check remains local to

TABLE 1. SOME PRACTICAL VARIANTS THAT CAN BE OBTAINED FROM THE *Labelled-MiniVoting* SCHEME.

	Voting Base	ValidInd	Publish	Flabel	ρ	$\mathcal{R}$
Helios-v3-hom	*Helios-hom*	verify ballot proof	last view	empty	addition ∘ last vote	corr-dec
Helios-v3-mix	*Helios-mix*	verify ballot proof	last view	empty	multiset ∘ last vote	corr-dec
Helios-v4	*Helios-hom*	verify ballot proof	last view	constant	addition ∘ last vote	corr-dec
Helios-light-weeding	*Helios-hom-IDweed*	verify ballot proof	last view	identity	addition ∘ last vote	corr-dec
Basic Scheme	*Labelled-MiniVoting*	return true	empty	empty	addition ∘ no-revote	return true

a particular voter's ballots, perhaps avoiding an expensive scan over the entire bulletin board. Formally, we consider a variant ValidLight of the Valid algorithm, displayed in Figure 9. If Flabel is injective, then ValidLight is functionally equivalent to Valid. This includes some interesting choices for Flabel, such as the identity function (or the function $x \mapsto (x, f(x))$ for any function f), or a PRP whose key is discarded immediately after setup.

Theorem 5. *Let ϕ be the predicate defined by*

$$\begin{aligned}\phi(m) = \quad & \mathsf{uL} \textit{ is injective} \\ & \wedge\, \forall\, i \in \mathsf{dom}(\mathsf{BB}). \\ & \qquad \exists\, id_i, c_i.\ \mathsf{BB}[i] = (id_i, \mathsf{uL}[id_i], c_i),\end{aligned}$$

where uL, BB *are the values of these variables in memory m. The following holds:*

$$\mathsf{Valid} \simeq_{\phi} \mathsf{ValidLight}.$$

Theorem 5 is proved in EasyCrypt and relies on the fact that each voter is assigned a unique label, and that all ballots from the ballot box have a matching label for a voter w.r.t to some pre-existing map that contains all voters and their assigned label.

Valid(BB, uL, b, pk)	ValidLight(BB, uL, b, pk)
1 : $(id, \ell, c) \leftarrow b$	1 : $(id, \ell, c) \leftarrow b$
2 : $e_1 \leftarrow \forall id'.\ (id', \ell, c) \notin \mathsf{BB}$	2 : $e_1 \leftarrow \big(b \notin \mathsf{BB}\big)$
3 : $e_2 \leftarrow \big(\ell = \mathsf{uL}[id]\big)$	3 : $e_2 \leftarrow \big(\ell = \mathsf{uL}[id]\big)$
4 : $e_3 \leftarrow \mathsf{ValidInd}(b, \mathsf{pk})$	4 : $e_3 \leftarrow \mathsf{ValidInd}(b, \mathsf{pk})$
5 : **return** $(e_1 \wedge e_2 \wedge e_3)$	5 : **return** $(e_1 \wedge e_2 \wedge e_3)$

Figure 9. Valid algorithm of *Labelled-MiniVoting* (left) and ValidLight, variant with light weeding (right).

Definition 13. Given a homomorphic encryption scheme $\mathcal{E}'$, two proof systems $\Sigma'_{\mathcal{R}'}$, $\Sigma_{\mathcal{R}}$, and abstract algorithms ValidInd, Publish, Flabel and Policy as specified, we define the Helios-hom-IDweed *scheme* constructed from these primitives as the scheme

$$\begin{aligned}&\mathcal{HW}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \mathsf{Policy}\right) = \\ &\quad \mathcal{HH}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathcal{R}}, \mathsf{ValidInd}, \mathsf{Publish}, \mathsf{Flabel}, \mathsf{Policy}\right) \\ &\qquad\qquad [\mathsf{ValidLight}].\end{aligned}$$

The following result follows from Theorem 5 and Corollary 3. The proof, verified in EasyCrypt, involves carefully checking that ϕ holds at all points where the validation algorithm is (or may be) called.

Corollary 4. *Let $\mathcal{E}'$ be an encryption scheme and $\Sigma_{\mathcal{R}}$, $\Sigma'_{\mathcal{R}'}$ be two proof systems, let $\Gamma = \mathsf{LPKE}(\mathcal{E}', \Sigma'_{\mathcal{R}'})$, and* ValidInd, Publish, Flabel, *and* Policy *be abstract algorithms as specifed. The* Helios-hom-IDweed *scheme constructed from these is ballot private, strongly consistent, and strongly correct whenever all of the following hold: i. $\mathcal{R}$ is voting friendly; ii. Γ is* IND-1-CCA*; iii. $\Sigma_{\mathcal{R}}$ is zero-knowledge; iv. $\mathcal{E}'$ is homomorphic; v.* ValidInd *that returns true for valid ciphertexts; vi.* Policy *commutes with* Dec**; vii.* Policy *is such that $b \in \mathsf{Policy}(\mathsf{BB}) \implies b \in \mathsf{BB}$; and viii.* Flabel *is injective.*

HELIOS WITH LIGHT WEEDING. We define *Helios with light weeding* as

$$\begin{aligned}&\textit{Helios-light-weed} = \mathcal{HW}\left(\mathcal{E}', \Sigma'_{\mathcal{R}'}, \Sigma_{\mathsf{corr\text{-}dec}},\right. \\ &\left.\mathsf{ValidInd}_{\mathsf{V}'}, \mathsf{Publish}_{\mathsf{last\ view}}, \mathsf{Flabel}_{\mathsf{identity}}, \mathsf{Policy}_{\mathsf{last\ vote}}\right).\end{aligned}$$

Helios-light-weed is a variant for Helios v3 that uses labels that uniquely identify voters and performs lighter weeding checks. Following Theorem 5, this is done without loss of privacy. Additionally, this method for weeding yields another defense mechanism against ballot privacy attacks [23] that has not yet been implemented in Helios. We give an overview of the practical variants of Helios our EasyCrypt proofs cover in Table 1.

5. Formalization

We now discuss the formalization, and highlight some of the key points it unveiled.

5.1. EasyCrypt

EasyCrypt [8], [9] is an interactive proof assistant for reasoning about concrete security of cryptographic constructions; to date, EasyCrypt has been used primarily for proving security of cryptographic primitives rather than more complex systems, with the notable exceptions of [4], [7].

EasyCrypt features a module system which combines facilities from module systems in programming languages, with a capability mechanism for restricting adversarial access to oracles or memories. The module system allows proving general principles once and for all, and later instantiating these principles in a particular setting. In addition, EasyCrypt features a theory mechanism that supports instantiation of types and operators used in a formalization. Our formalization heavily relies on these mechanisms to achieve modularity and make verification of several hundreds of variants tractable.

The EasyCrypt formalization of ballot privacy closely follows the development outlined in the previous section, but with two important differences. First, security statements in EasyCrypt are concrete, i.e. the advantage of a (constructed) adversary is given as an arithemetic expression of its capacities and of the advantage of sub-adversaries—whereas for readability, our presentation in the previous sections follows the usual style of asymptotic security. More importantly, EasyCrypt uses a relational program logic to formalize code-based game-based reductionist arguments. The latter uses a series of probabilistic programs with adversarial code, called *games*, and of probabilistic claims relating the probability of one or more events in one or more games, to establish its main claim. In EasyCrypt, probabilistic claims are derived using probabilistic Relational Hoare Logic (pRHL), which generalizes Relational Hoare Logic [11] to a probabilistic setting. pRHL is a program logic whose judgments are of the form $\{\Phi\}\ c_1 \sim c_2\ \{\Psi\}$, where c_1 and c_2 are games, and Φ and Ψ are relations on program states. The rules of pRHL allow the user to derive *valid* judgments. When Ψ is of an appropriate form (concretely, $E_1\langle 1\rangle \Rightarrow E_2\langle 2\rangle$, where $\langle i\rangle$ is used to denote the memory in which the event is interpreted), validity of the above judgment implies the probabilistic claim:

$$\Pr[c_1, m_1 : E_1] \leq \Pr[c_2, m_2 : E_2]$$

stating that the probability of event E_1 after executing c_1 in initial memory m_1 is upper-bounded by the probability of event E_2 after executing c_2 in m_2.

Interestingly, our formalization highlighted one limitation of EasyCrypt. In order to achieve maximal generality, our proof of ballot privacy is modular and assumes IND-1-CCA security of the underlying encryption scheme. An important goal for future work is to prove in EasyCrypt that encryption schemes commonly used in Helios, including El Gamal with zero-knowledge proofs, achieve IND-1-CCA security. This component does not require changes for the *Labelled-MiniVoting* scheme, and can be viewed as a stand-alone component. However, it involves extending EasyCrypt to reason about rewinding arguments since existing IND-1-CCA proofs use Bellare and Neven's General Forking Lemma [10].

5.2. Issues with Pen-and-Paper Proofs

The formalization effort presented here highlighted two shortcomings in existing pen-and-paper proofs for *Labelled-MiniVoting* and its applications.

SIMULATION PROOFS WITH RANDOM ORACLES. Starting from existing pen-and-paper proofs of BPRIV for *Labelled-MiniVoting*, and attempting to formalize them led us to deeper considerations on the interactions of simulation-based security notions and proofs with random (or stateful) oracles, in a way similar to Fiore and Nitulescu [28]. Indeed, in the proof of *Labelled-MiniVoting*, it is highly important to split random oracles between those that are taken over by the simulator and those that need to remain independent (in particular, so they can be taken over in lower-level simulations for the primitives). Formally, this highlighted the need to strike a careful balance between abstraction–which supports proof reuse and enforces that the realization of a component be irrelevant to the current security proof, and the need to have a full specification of the system on which the proof operates. In particular, the need for formality in the treatment of random oracles caused some false starts in the development, as core definitions and theorem statements had to be adapted, first to make room for random oracles, then to clearly distinguish those random oracles that are protocol-relevant (those used in the proof of correct decryption) and need to be simulated from those that are only relevant to lower-level primitives and can thus be kept abstract (those used in the labelled public-key encryption scheme), and finally to support zero-knowledge relations with access to the abstract random oracles.[9] However, we note that this formal issue does not imply the existence of attacks if the same hash function is used for computing the proof of decryption and for other purposes. Still, in the spirit of recent standardisation efforts, we do recommend that domain separation be used in this case.

A MISSING ASSUMPTION. Existing proofs of ballot privacy for abstract systems similar to *Labelled-MiniVoting* [14] do not make use of the strong correctness, whereas it is in fact a necessary assumption. In practice, the game transition where strong correctness is used implies an additional term (corresponding exactly to the upper-bound on the strong correctness advantage) in the final security bound for ballot-privacy, which does not appear in asymptotic security treatments, but may be critical when evaluating security margins to determine concrete security parameters.

5.3. Discussion

The final `qed` took about one person-year to complete. The statement and proof of Theorem 2 took about 75% of the effort, while the specialization component made up the rest (including some changes to the statement of Theorem 2 and related definitions to support zero-knowledge relations with random oracles). Table 2 shows the development size and verification times; the figures were obtained running EasyCrypt on an HP ZBook with i7-4800MQ CPU and 32GB RAM, running Ubuntu 14.01.

EXTENSIONS, REFINEMENTS AND GENERALISATIONS. It is clear that much of the effort could have been spared if we had chosen to forgo generality and consider a specific instance of Helios. However, the benefits of a general proof is that it should now be easy to adapt our framework (and the corresponding proofs) to: formalizing privacy of other voting systems, formalizing other security properties, and carrying our formal guarantees down to implementations, also considering security under weaker trust assumptions. For example, a refinement of our framework would explicitly distinguish between the public and the secret part

9. A theorem that does not consider the latter two issues, although much easier to state and prove, would not be instantiatable as broadly as ours.

TABLE 2. STATISTICS ON THE EASYCRYPT PROOF DEVELOPMENT FOR E-VOTING.

Voting Stages		Definition (no. lines)	Proof (no. lines)	Verif. Time (seconds)
General Framework	General Concepts	961	2891	158
	Labelled-MiniVoting	231	4466	447.8
Specialization	*PreHelios*	145	918	10.8
	Helios-hom	164	662	51.3
	Helios-hom-IDweed	22	842	67.3
	Helios-mix	95	788	5.7
Examples	Helios v.* + Basic	658	2493	28.7
	540 variants	540 * 132	540 * 492	540 * 3.4

of a user's credential. Schemes where such distinction is crucial e.g. Belenios [21] or Civitas [31], could then be instantiated in the resulting framework. Of course, any extensions would require re-working the formal results to deal with the changes. However in this case the new proofs would mirror closely the development that we present in this paper and may need to only locally account for the changes. Other security properties of interest include in particular verifiability [22], accountability [34], receipt-freeness, and coercion-resistance [31]. Extending our work to these properties would first require to fully formalize them but then we believe we could again rely on the genericity of our approach to consider classes of protocols.

TOWARDS VERIFIED IMPLEMENTATIONS. We conclude this discussion section by observing that the family of machine-checked specifications we produce gets us one step closer to the production of machine-checked implementations for voting protocols. Indeed, EasyCrypt itself provides several ways of producing machine-checked implementations from machine-checked specifications. The first is to extract, from the EasyCrypt specification, a specification in the WhyML language, a specification language suitable for use with many program verifiers. This technique, which then carries the final result over to implementations by manual and verified refinement, was used in the past to obtain proofs of security, even in the presence of side-channel adversaries, for RSA-OAEP [5] and MEE-CBC [6]. Although the use of random oracles are an obstacle to the verification of the whole system, hash functions are in fact a very small part of the system–and not much can usually be established about their security. For example, Almeida et al. [6] simply exclude symmetric primitives from the refinement proof, but include them in the side-channel-freeness check. At present, these techniques would not scale to the needs of systems of the size and complexity of Helios. The second is to directly extract, from the EasyCrypt specification, a working OCaml implementation of the protocol. This mechanism was previously used to produce relatively efficient implementations of secure and verifiable computation protocols [4]. In the present case, because we leave a number of components unspecified or abstract (in particular, we only consider functional abstractions of the tally process and bulletin board), our specification is not sufficient to produce a full working implementation of the voting scheme. However, it would be sufficient to produce a working implementation of a voting client, be it device-based or direct-recording electronic (DRE) based. We do not here make any claims on the practical security of this extracted code.

6. Conclusion and future work

We have developed a machine-checked framework for proving ballot privacy of electronic voting systems. Our framework is sufficiently general to cover hundreds of variants of Helios, including most existing implemented ones.

We reduced ballot privacy to security of the underlying primitives, relying for example on an IND-1-CCA-secure encryption scheme and on secure mixnets. ElGamal together with proofs of knowledge has been proved secure in [15]. However, formalizing this proof in EasyCrypt would be challenging since it requires support for rewinding adversaries. Similarly, proving security of mixnets is hard and developing an EasyCrypt framework for mixnets would form an interesting research project.

As future work, we plan to extend our framework to other voting systems as discussed in the previous section. We also plan to consider other properties such as verifiability, accountability, receipt-freeness, or coercion-resistance. Verifiability seems of easy reach: it requires to formalise this property, for example along the lines of [21] and identify under which conditions *Labelled-MiniVoting* preserves verifiability (e.g. depending on what is published). This should allow us to prove verifiability of many of the variants of Helios. Accountability would require more work. Indeed, while Helios is close to being accountable, it is necessary to first specify exactly who is responsible e.g. for the data displayed on the ballot box and the distribution of the credentials and how this is cryptographically enforced. We would first need to enrich *Labelled-MiniVoting* to account for these extra procedures. Then receipt-freeness and coercion-resistance would require even more extensions to *Labelled-MiniVoting* since Helios and its variants, and therefore the current *Labelled-MiniVoting* scheme, are *not* coercion-resistant. Considering these various properties under varying trust assumptions on each of the parties would also be interesting.

Acknowledgments

The authors thank Gilles Barthe for useful input and discussions and to David Bernhard and David Galindo for their early involvement in this work. We also thank Matt

Fredrikson who has shepherded this paper to its current form, and the anonymous reviewers at IEEE Security & Privacy for their helpful comments. This work has received funding from the Office of Naval Research (grant agreement NO N000141512750) and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC).

References

[1] *Easycrypt code for Labelled-MiniVoting privacy*. https://github.com/catalindragan/minivoting-privacy/tree/master/proof.

[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] B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. USENIX/ACCURATE Electronic Voting Technology (EVT 2009), 2009.

[4] J. B. Almeida, M. Barbosa, G. Barthe, G. Davy, F. Dupressoir, B. Grégoire, and P.-Y. Strub. Verified implementations for secure and verifiable computation. Cryptology ePrint Archive, Report 2014/456, 2014. http://eprint.iacr.org/2014/456.

[5] J. B. Almeida, M. Barbosa, G. Barthe, and F. Dupressoir. Certified computer-aided cryptography: efficient provably secure machine code from high-level implementations. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 1217–1230, 2013.

[6] J. B. Almeida, M. Barbosa, G. Barthe, and F. Dupressoir. Verifiable side-channel security of cryptographic implementations: constant-time MEE-CBC. In *23rd International Conference on Fast Software Encryption (FSE)*, Mar. 2016. To appear.

[7] 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.

[8] 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.

[9] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011. Best Paper Award.

[10] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399. ACM, 2006.

[11] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*, page 43. ACM, January 2004.

[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, V. Cortier, O. Pereira, B. Smyth, and B. Warinschi. Adapting helios for provable ballot privacy. In *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, volume 6879 of *Lecture Notes in Computer Science*, pages 335–354. Springer, 2011.

[15] D. Bernhard, O. Pereira, and B. Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 626–643. Springer, 2012.

[16] D. Bernhard and B. Warinschi. Cryptographic voting - A gentle introduction. In A. Aldini, J. Lopez, and F. Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604, pages 167–211. Springer, 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] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, 1992.

[19] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, Feb. 1981.

[20] V. Cortier, F. Eigner, S. Kremer, M. Maffei, and C. Wiedling. Type-based verification of electronic voting protocols. In *Proceedings of the 4th Conference on Principles of Security and Trust (POST'15)*, volume 9036 of *Lecture Notes in Computer Science*, pages 303–323, London, UK, April 2015. Springer.

[21] 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.

[22] V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung. Sok: Verifiability notions for e-voting protocols. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P'16)*, San Jose, CA, USA, May 2016. IEEE Computer Society Press.

[23] 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.

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

[25] 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.

[26] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology (EUROCRYPT'97)*, pages 103–118, 1997.

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

[28] D. Fiore and A. Nitulescu. *On the (In)Security of SNARKs in the Presence of Oracles*, pages 108–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[29] T. E. Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

[30] M. Jakobsson, A. Juels, and R. L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium*, pages 339–353, Berkeley, CA, USA, 2002. USENIX Association.

[31] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections*, pages 37–63, 2010.

[32] S. Khazaei, T. Moran, and D. Wikström. A mix-net from any CCA2 secure cryptosystem. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer, 2012.

[33] R. Küsters and T. Truderung. Security analysis of re-encryption RPC mix nets. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 227–242. IEEE, 2016.

[34] R. Küsters, T. Truderung, and A. Vogt. Accountabiliy: Definition and Relationship to Verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 2010)*, pages 526–535. ACM Press, 2010.

[35] C. H. Lim and P. J. Lee. Another method for attaining security against adaptively chosen ciphertext attacks. In *In Advances in Cryptology–Crypto '93*, pages 420–434. SpringerVerlag, 1993.

[36] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, CCS '01, pages 116–125, New York, NY, USA, 2001. ACM.

[37] C. Park, K. Itoh, and K. Kurosawa. *Efficient Anonymous Channel and All/Nothing Election Scheme*, pages 248–259. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[38] V. Shoup. A proposal for an iso standard for public key encryption (version 2.0), 2001.

[39] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Journal of Cryptology*, pages 1–16. Springer-Verlag, 1999.

[40] 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.

[41] D. Wikström. A universally composable mix-net. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2004.

[42] 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.