Private votes on untrusted platforms: models, attacks and provable scheme

Sergiu Bursuc, Constantin-Catalin Dragan, Steve Kremer

Private votes on untrusted platforms: models, attacks and provable scheme

Sergiu Bursuc
Inria Nancy-Grand'Est,
LORIA, France
sergiu.bursuc@inria.fr

Constantin-Cătălin Drăgan
Department of Computer Science,
University of Surrey, UK
c.dragan@surrey.ac.uk

Steve Kremer
Inria Nancy-Grand'Est,
LORIA, France
steve.kremer@inria.fr

***Abstract*—Modern e-voting systems deploy cryptographic protocols on a complex infrastructure involving different computing platforms and agents. It is crucial to have appropriate specification and evaluation methods to perform rigorous analysis of such systems, taking into account the corruption and computational capabilities of a potential attacker. In particular, the platform used for voting may be corrupted, e.g. infected by malware, and we need to ensure privacy and integrity of votes even in that case.**

We propose a new definition of vote privacy, formalized as a computational indistinguishability game, that allows to take into account such refined attacker models; we show that the definition captures both known and novel attacks against several voting schemes; and we propose a scheme that is provably secure in this setting. We moreover formalize and machine-check the proof in the EasyCrypt theorem prover.

I. Introduction

Electronic voting through the Internet is increasingly popular, being now routinely deployed for professional elections, and even, sometimes, for politically binding elections; Estonia has offered the possibility to vote through the Internet for parliament elections since 2007. An election is however a security sensitive process and should at least guarantee two main security properties: *election integrity* - the announced result should correspond to the tally of the votes cast by eligible voters; and *vote privacy* - particular voter choices should remain secret.

These goals are extremely tricky to achieve, as they should hold even against corrupted election organizers or corrupted software that runs the election. Secret, end-to-end verifiable election systems typically use cryptographic protocols to ensure these properties. Vote privacy is generally enforced by encrypting individual votes and anonymizing them before tally, e.g. using mix-nets, blind signatures or homomorphic encryption. Integrity of the election is ensured by end-to-end verifiability, which relies on a combination of cryptographic proofs and individual or universal auditing for various components of the system.

One particularly challenging problem arises when one makes the (realistic) assumption that the device used for voting is not trustworthy. Dedicated malware could leak a voter's choice or change her choice before casting the ballot. Proof of concept malware has been demonstrated in at least two political elections, one in Estonia [53], and one in France [29].

Benaloh [6] proposed a cut-and-choose method that allows to audit the encryption device, but this method does not prevent privacy leaks and usability studies suggest that most voters do not perform this audit [39]. Other methods require the distribution of personalized code sheets [34], which introduces an additional, security sensitive, burden on the election setup. Finally some protocols compute a vote code representing the choice from a private credential (possibly with the use of an additional device) [31]. This problem is sometimes referred to as the malicious platform problem, and the corresponding security property (for verifiability) is called *cast-as-intended*.

Given the complexity of these protocols, we believe, and show in this paper, that an indispensable element in their security evaluation is the use of rigorous definitions of the properties, the system and the adversary model, together with formal proofs that the given models satisfy the stated definitions.

Our contributions. We study the problem of guaranteeing vote privacy when various parts of the voting platform, in particular the device that is used for computing the ballot to be cast, may be compromised. We propose models that allow to specify voting schemes, voting platforms and their execution in such an adversarial environment. We rely on these models, on one hand, to capture attacks on deployed voting schemes [1], [11], [48] in presence of realistic adversaries and, on the other hand, to propose a new voting scheme that we prove secure.

Models. We provide a new definition of vote privacy, allowing augmented attacker power which takes into account compromised machines. Our definition is expressed as a cryptographic indistinguishability game, in the line of [8], [16], specifying that an attacker cannot extract more information from a real execution of the voting system than from an ideal execution, where ballots perfectly hide votes and the tally is perfect. Getting the "right" definition is however rather tricky, as illustrated by the numerous shortcomings of existing definitions in [8]. Most existing definitions allow an adversary to request a voter to cast one of given votes v_0 or v_1 (depending on the challenge bit b of the experiment). The ballot for vote v_b is then directly added to a bulletin board that records cast votes. In our setting the adversary has the possibility to modify a ballot in an arbitrary way, and decide when and whether it is cast (to model that the

voting platform, or the bulletin board are compromised). The definition is parametrized to allow (or not) modifications of parts of the ballot, credential leakage, revoting policies, voter corruption, etc. This flexibility allows to consider a large variety of election schemes and adversaries.

Attacks. We show that our definition allows to capture attacks against several voting schemes [1], [3], [49], some of them previously believed to be secure. Helios [1] is a prominent voting scheme that can be used when the voting device is trusted not to leak votes. Yet, as shown in [21], vote-privacy can be compromised by a simple ballot copy attack against the bulletin board. Furthermore, even when techniques to prevent ballot copy are in place at the level of the bulletin board, a similar attack can be performed by compromising other parts of the voting platform in order to force a revote [47] (see Section II-C). Our definition is the first that formally captures (in a computational model) both the original attack and its latter variants, by taking into account refined adversarial control over the platform and the ballot casting process.

For the case when the adversary may corrupt the voting device and leak votes, we consider schemes that rely on computing a vote code: given a vote v and a secret credential w, the vote code entered on the platform is computed to be the cyclic shift $v + w \mod n$, where n is the number of candidates. This way of encoding votes (we call it OnePad in the following) underlies the popular Prêt-à-Voter system [11], [14], [48], and also the remote voting scheme proposed in [3]. Intuitively, vote-privacy is protected in these systems by the fresh, random and secret credential w used to hide each vote. However, these schemes turn out not to satisfy our privacy definition. Indeed, we show that an adversary that controls the voting device and casts $f(v + w \mod n)$, for a suitably chosen function f, can deduce v by analyzing the outcome of the tally, even if there are honest voters whose device is not compromised and whose votes should contribute to the privacy of v. We show that even a seemingly stronger version of Prêt-à-Voter, where general permutations are used instead of a cyclic shift, is also vulnerable to this type of attack.

Provably secure scheme: TokPad. In order to make OnePad secure, we augment the ballot with an authentication tag, that can be computed on a separate device (we call it token in the following). Talliers verify the token before decoding votes and revealing the outcome. Then, in order to perform an attack as above, the adversary has to control both the voting device and the token, thus trust is distributed among the two. Indeed, we formally prove that privacy is satisfied in each of the two cases: honest device (against untrusted token) or honest token (against untrusted device). The proof is based on several game transformations - each relying on cryptographic, platform or protocol assumptions - in order to show that, at the end, the term $v + w \mod n$ does indeed act as a one-time pad that protects the vote v. To increase confidence in our result we have used the special purpose theorem prover EasyCrypt [24] to machine-check the proof.

Voting system based on TokPad. We address several practical aspects related to the instantiation of our scheme. We consider two possible cryptographic instances for the token functions, and show that, even for a small tag length, which may be necessary for the usability of the scheme, we can derive meaningful security guarantees from the generic security of TokPad. For publicly verifiable tallying, encryptions of secret credentials w have to be posted on a bulletin board. We show how, based on any additively homomorphic encryption scheme, ballots can be tallied homomorphically, so we can avoid the use of re-encryption mixes for anonymization. We leave some other deployment issues as open questions, such as generation and distribution of voting credentials and token keys, formal proofs of end-to-end verifiability and putting everything together in a transparent way for the user.

We present our models and attacks in section II, the design and formal security results for TokPad in section III, its deployment options in section IV and related work in section V.

II. Voting schemes and untrusted platforms

We review two schemes that aim to protect votes on untrusted platforms, propose a formal model to analyze such schemes and their privacy properties, and show how several privacy issues, both old and new, are captured by our model.

Notations. For $a, b, c \in \mathbb{N}$, we let $a \oplus_c b = a + b \; mod \; c$ and $a \ominus_c b = a - b \; mod \; c$. A *selection function* for a vector $a = (a_1, \ldots, a_n)$ is defined by a set of indices $\iota = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$: we let $\iota(a) = (a_{i_1}, \ldots, a_{i_k})$ and $\overline{\iota} = \{1, \ldots, n\} \setminus \iota$. We let $\bot, \top$ be the particular selection functions defined by $\emptyset$ and $\{1, \ldots, n\}$ respectively. For a list L, we denote by $L \leftarrow L + a$ the append of a to L, by $L[i]$ the i-th element of L, and by $L[\mathcal{I}]$ the sublist selected by the indices in $\mathcal{I}$. We denote by empty the list with no elements.

For an encryption scheme Enc, we denote by $\mathsf{Enc}_{\mathsf{pk}}(\cdot), \mathsf{Dec}_{\mathsf{sk}}(\cdot)$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$ the operations of encryption, decryption and respectively key generation for a public key pk and corresponding secret key sk. λ represents the security parameter and 1^λ is used as argument by some cryptographic algorithms to determine the size of objects to be constructed. For a set M, we denote by $x \leftarrow\!\!\$\, M$ the fact that x is uniformly sampled from M.

A. Schemes based on a one time pad

A popular idea for protecting private votes from compromised voting platforms is to add a secret mask to the vote. A vote v is encoded as $v \oplus_m w$, where m is the number of candidates and w is a secret credential known only to the voter and to trusted election authorities. Variants of the Prêt-à-Voter scheme [14], [49] and the remote voting scheme of [3] rely on this idea.

Prêt-à-Voter [14], [49]. Voters receive a paper ballot containing a serial number and a random permutation $[c_{\pi(0)}, \ldots, c_{\pi(m-1)}]$ of the candidate list $[c_0, \ldots, c_{m-1}]$. A scanner associated to the voting device reads the serial number from the ballot; the permutation is only known to the voter, who inputs to the voting device the index u that corresponds to the desired candidate c_v. We have $u = \pi(v)$. The voting device

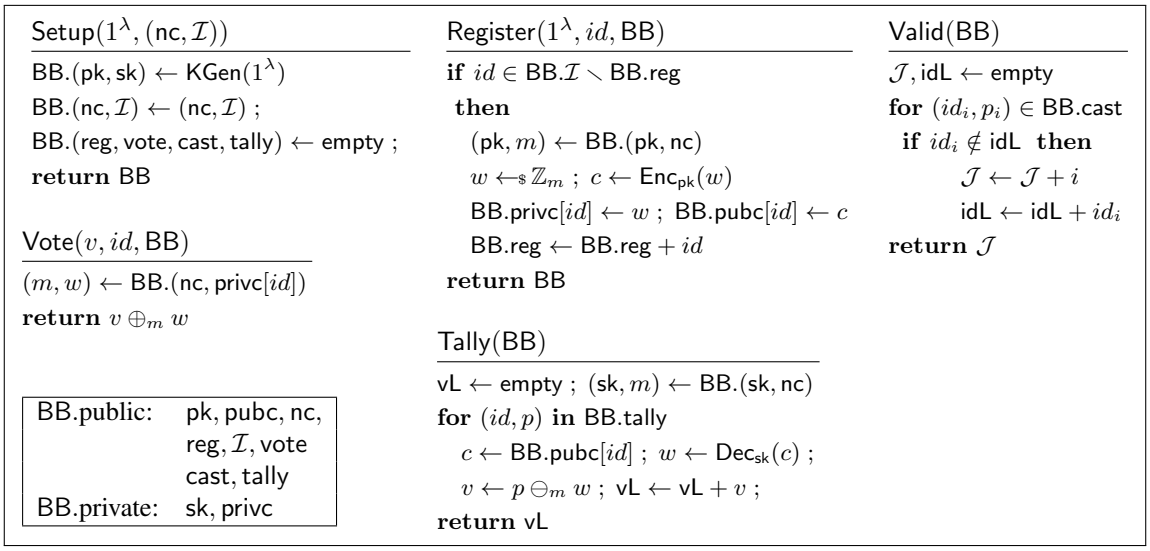

Fig. 1. Algorithms defining the OnePad voting scheme with encryption scheme $(\mathsf{TKGen}, \mathsf{Enc}, \mathsf{Dec})$, number of candidates nc, and set of eligible voters $\mathcal{I}$

posts the index u and the serial number on the bulletin board. The random permutation π is encrypted before the election starts and $\mathsf{Enc}_{\mathsf{pk}}(\pi)$ is associated to the corresponding serial number on the bulletin board. The pairs $[u, \mathsf{Enc}_{\mathsf{pk}}(\pi)]$ can then go through re-encryption mixes [37], [44], after which we can decrypt the permutation π, and obtain the decoded vote $c_{\pi^{-1}(u)}$. In the following we consider the class of permutations that are cyclic shifts which has been suggested in [14], [49]. In this case the permutation π can be represented by an offset w, i.e. $\pi(v) = v \oplus_m w$ and $\pi^{-1}(u) = u \ominus_m w$.

Remote voting [3]. As in Prêt-à-Voter, the vote is hidden from the voting device relying on a mask $v \oplus_m w$. The secret shift w is sent directly to the voter's mobile phone. The corresponding code $v \oplus_m w$, for the desired candidate v, should then be computed by the voter, possibly with help from the mobile phone. The voting device computes a ciphertext $\mathsf{Enc}_{\mathsf{pk}}(v \oplus_m w)$, and sends it to election servers. One election server keeps w; relying on homomorphic properties of Enc, w can be used to compute an encryption of v for tally. A second server is responsible for sending to the voter a confirmation code corresponding to v.

The OnePad voting scheme. We consider an abstract scheme to study the privacy properties of systems described above. It is formally defined in Figure 1, that we explain in more detail after introducing the model in the next section. We concentrate on the core idea that uses a one time pad to mask private votes, considering a simplified tally procedure where talliers are trusted. The problems that we investigate later on are independent of the tally procedure. We also postpone discussions on verifiability, noting that privacy should hold even when ballots or the election have not been properly verified.

B. Formal model and definitions

Our model of voting schemes and ballot privacy follows the approach from [8], [16]: a set of algorithms and oracles describes precisely the interaction of an adversary with the voting scheme, and privacy is defined by an indistinguishability experiment. To capture malicious voting platforms and schemes that aim to counter them, we generalize certain oracles, allowing more adversary influence on cast ballots. We also have a more general structure for the bulletin board, in order to store data related to voting credentials, and any other data that may be needed to capture the execution of voting schemes.

In our experiment, a *bulletin board* BB is a data structure with several attributes that can be accessed by agents participating in a voting scheme. The attributes can be *public* (unrestricted read but restricted write, as specified by the voting scheme) or *private* (restricted read and restricted write). We assume a basic BB structure that contains the following attributes, which may be augmented as needed by particular schemes: $\mathsf{BB}.\mathsf{sk}, \mathsf{BB}.\mathsf{pk}$ are respectively the private and public key of the election; $\mathsf{BB}.\mathsf{privc}, \mathsf{BB}.\mathsf{pubc}$ are maps that assign to voter identities a vector of private and public credentials, respectively; $\mathsf{BB}.\mathsf{nc}$ is the number of candidates; $\mathsf{BB}.\mathcal{I}$ is the list of eligible voters; $\mathsf{BB}.\mathsf{reg}$ is the list of registered voters; $\mathsf{BB}.\mathsf{vote}$ is a map that assigns to ids the list of ballots constructed by the respective voter - as we assume that the platform is possibly compromised, not all of them are necessarily cast to the election server; $\mathsf{BB}.\mathsf{cast}$ is the list of ballots cast to the election server; $\mathsf{BB}.\mathsf{tally}$ is the list of ballots to be tallied - it may be different from $\mathsf{BB}.\mathsf{cast}$, because some cast ballots may be deemed invalid before tally. As most voting systems allow revoting, we use the notation $\mathsf{BB}.\mathsf{vote}[id]$ to specify the list of all ballots constructed for a given voter id; and $\mathsf{BB}.\mathsf{vote}[id, i]$ to specify the ith ballot in this list.

A *voting scheme* $\mathcal{V}$ is defined by a bulletin board and the following algorithms:

- $\mathsf{Setup}(1^\lambda, \rho)$: generates initial data required by the voting scheme, for example a public key for the election and a corresponding secret key. It stores and returns the

generated data on a fresh bulletin board. λ is the security parameter and ρ are parameters that may be required by a voting scheme, e.g. the number of candidates.

- $\mathsf{Register}(1^\lambda, id, \mathsf{BB})$: generates the private credential vector $\mathsf{BB.privc}[id]$ and the public credential vector $\mathsf{BB.pubc}[id]$ for the voter with the respective id. It returns the new state of the bulletin board.
- $\mathsf{Vote}(v, id, \mathsf{BB})$: takes as input a vote v, a voter identity id and parameters from the bulletin board, in particular the private credential stored in $\mathsf{BB.privc}[id]$. It returns a ballot b, which may be a vector of elements. In the security definition, we will assume that the Vote algorithm is executed on a platform that is not corrupted. For example, in OnePad, Vote models the computation of the one-time pad, while in Helios [1] it models the operation of encrypting the vote, assumed to be performed on an honest device. The corruption capabilities of the attacker will be modeled by dedicated constructions in the security definition.
- $\mathsf{Valid}(\mathsf{BB})$: takes as input a bulletin board containing, in particular, the list of cast ballots, the secret key of the talliers, and public credentials of registered voters. It returns the sequence of indices $\mathcal{J}$ for ballots in $\mathsf{BB.cast}$ that can be tallied. So we have $\mathsf{BB.tally} = \mathsf{BB.cast}[\mathcal{J}]$.
- $\mathsf{Tally}(\mathsf{BB})$ returns the outcome of tallying ballots from $\mathsf{BB.tally}$.

Figure 1 formally defines the above algorithms for the OnePad voting scheme described in subsection II-A. The Setup generates a key pair and stores it on the bulletin board. Register computes a private and public credential for a given voter id, and stores them on the bulletin board. Vote returns the sum of the vote and the private credential, in a cyclic group determined by the number of candidates. Tally relies on the secret key to decrypt private credentials and unmask the vote.

A voting scheme $\mathcal{V}$ defined as above is used in our *privacy experiment*, formally defined in Figure 2, that models its execution in a given environment, where an adversary $\mathcal{A}$ may call the algorithms in a certain order, with certain parameters, corrupt parties and obtain associated private data, etc. This experiment is specified via oracles that may enforce restrictions on the interaction of $\mathcal{A}$ with $\mathcal{V}$. The restrictions are justified by assumptions about the voting infrastructure (we call them platform assumptions) or voter behavior (we call them election assumptions). Together with the specification of $\mathcal{V}$, these assumptions will be parameters for the voting experiment. Finally, the oracles also allow us to control the voting experiment in order to avoid trivial attacks.

A voting scheme may assume several voting devices, e.g. a mobile phone and a browser. Furthermore, various voting credentials, like passwords and cryptographic keys, may be used on each device. We will use selection functions, introduced in notation paragraph of this section, to specify scenarios where some of the devices and credentials are trusted, while others are under the control of the adversary. A *platform assumption* for $\mathcal{V}$ is a pair of selection functions (γ, μ), where γ specifies what part of private credentials *may leak* to the adversary, and μ specifies what part of honest ballots *may be modified* by the adversary. Consider, for example, a ballot that is a vector of 2 elements $b = (b_1, b_2)$. When $\mu = \top$ the adversary may modify b at will, while $\mu = \bot$ does not allow any modification and $\mu = \{i\}$ ($i \in \{1, 2\}$) allows modification of b_i only. The effect of platform assumptions is captured in oracles from Figure 2:

- Credential leakage in $Oreg$: a part of the private credential $\mathsf{BB.privc}[id]$, selected by γ, is returned to the adversary at registration time.
- Ballot modification in $Ocast$ following $Ovote$: the adversary obtains a ballot b created by an honest voter, and submits a possibly different ballot b'. Such a corruption attempt will succeed only if $\overline{\mu}(b') = \overline{\mu}(b)$, i.e. at most $\mu(b)$ is modified by the adversary.

A voting scheme may also assume a certain behavior from honest voters with respect to the voting procedure. An *election assumption* for $\mathcal{V}$ is a pair of predicates $(\Phi\mathsf{v}, \Phi\mathsf{c})$ specifying when voting and respectively voter corruption are possible. Formally, a predicate $\Phi \in \{\Phi\mathsf{c}, \Phi\mathsf{v}\}$ takes as inputs a voter id and a bulletin board BB.

Example 1. *The following forbids re-voting and unregistered voters ($\Phi\mathsf{v}$) and forbids corruption after vote ($\Phi\mathsf{c}$):*

$$\begin{aligned} \Phi\mathsf{v}(id, \mathsf{BB}) &= |\,\mathsf{BB.vote}[id]\,| < 1 \;\&\; id \in \mathsf{BB.reg} \\ \Phi\mathsf{c}(id, \mathsf{BB}) &= |\,\mathsf{BB.vote}[id]\,| < 1 \end{aligned}$$

The set of oracles and privacy experiment for the execution of a voting scheme $\mathcal{V}$, with platform and election assumptions $\Psi = (\gamma, \mu, \Phi\mathsf{v}, \Phi\mathsf{c})$, are defined in Figure 2, which we explain in the following. The adversary $\mathcal{A}$ is modeled by a probabilistic polynomial time (ppt) algorithm provided with a set of oracles.

Real vs Ideal world execution: As in [8], [9], [16], [52], we model vote privacy as the inability of $\mathcal{A}$ to distinguish between two runs of the voting experiment: one where $\mathcal{A}$ interacts with a real execution of the voting scheme and one where $\mathcal{A}$ interacts with an idealized execution. A uniform randomly drawn index $\beta \in \{0, 1\}$ determines the case we are in: $\beta = 0$ the real world, and $\beta = 1$ the ideal world. Intuitively, the real world models a normal adversarial execution of the voting scheme, whereas the ideal world models an execution where, for honest voters: (i) ballots returned to the adversary contain votes that may be arbitrarily distinct from the ones in the real world (the adversary is free to choose them, e.g. they can be constant or randomly drawn) and (ii) the outcome of the tally is perfect, i.e. even if the adversary managed to modify the vote in a cast ballot, it is the original unmodified vote that is part of the outcome. Then, being able to distinguish the real execution from the ideal (i.e. returning the correct guess $\beta' = \beta$), means that $\mathcal{A}$ is able to exploit a (real world) weakness in the voting scheme: at the cryptographic level of the ballot, at the protocol level defined in oracles, or a combination of both.

Voter registration: oracle $Oreg(id)$ registers voters with a given id. Fresh credentials are created for id and stored on the bulletin board. $\mathcal{A}$ can obtain the public credentials from the bulletin board via the $Oboard()$ oracle. $\mathcal{A}$ also obtains parts of the private credential as specified by the leakage function.

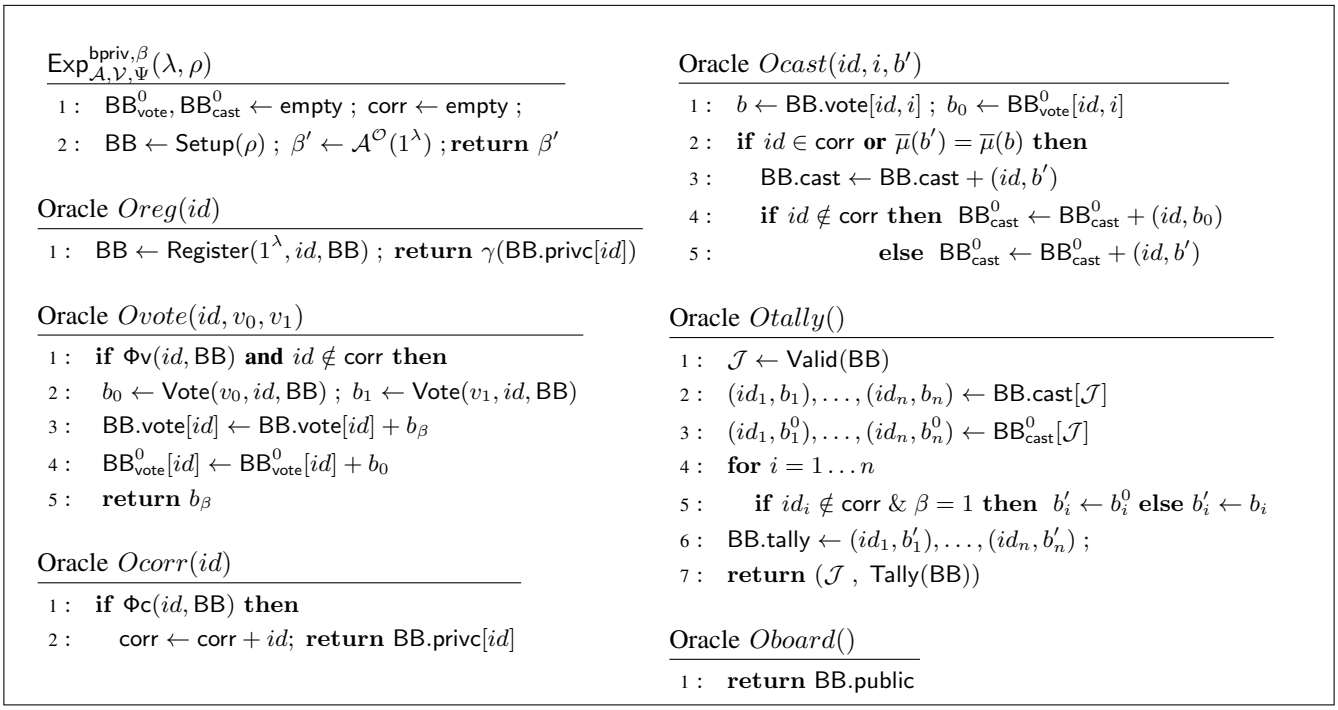

Fig. 2. Privacy experiment for a scheme $\mathcal{V}$ with adversary $\mathcal{A}$ and oracles $\mathcal{O} = \{Oreg, Ocorr, Oboard, Ovote, Ocast, Otally\}$. Platform and election assumptions are represented by $\Psi = (\gamma, \mu, \Phi\mathsf{v}, \Phi\mathsf{c})$.

Voter corruption: oracle $Ocorr(id)$ allows corruption of registered voters. It reveals to $\mathcal{A}$ the private credentials of the corresponding voter. The election assumption $\Phi\mathsf{c}$ may prevent corruption in some cases, for instance after voting. We call uncorrupted registered voters honest.

Voting: oracle $Ovote(id, v_0, v_1)$ invokes the voting procedure for honest voters. We construct two ballots: b_0 encoding v_0, i.e. the real vote, and b_1 encoding v_1, i.e. the ideal vote; $\mathcal{A}$ obtains b_β, where β is the challenge bit in the privacy experiment. The ballot b_β is also stored in $\mathsf{BB.vote}$. To be able to perform the ideal tally, which requires the perfect voter ballots, the experiment also stores b_0 on $\mathsf{BB}^0_{\mathsf{vote}}$. The election assumption $\Phi\mathsf{v}$ may prevent the execution of $Ovote$ in some cases, for example when id is corrupted or has already voted. For corrupted voters, $\mathcal{A}$ can construct the ballots itself, because it has the private credentials.

The $Ovote$ oracle models the construction of the ballot in a trusted environment, before being passed on to the untrusted platform controlled by the adversary: for OnePad, it models the voter or the mobile device that computes the masked vote; for Helios, it models the voting device that encrypts the chosen vote and posts it to the bulletin board. The $Ocast$ oracle models the part of the voting process that is controlled by the attacker, and which may affect the ballots constructed with $Ovote$: the ballot casting application, the network, the bulletin board, etc. For OnePad, the device where the voter inputs the shifted vote is also modeled by $Ocast$, because it is assumed corrupted.

Ballot casting: oracle $Ocast(id, i, b')$ casts a ballot on the bulletin board. $\mathcal{A}$ may construct the ballot b' by manipulation of previously created ballots. If the corresponding voter is honest, the index i specifies which of the previously constructed honest ballots is replaced by b'; the platform assumption may protect some parts of this honestly constructed ballot, so the oracle ensures that $\mathcal{A}$ cannot modify them. For dishonest voters, the $Ocast$ operation is the same, except there is no restriction on the cast ballots: $\mathcal{A}$ has the private credentials, fully controls the platform, and can cast any ballot. To be able to perform the ideal tally, for honest voters we copy the original, unaltered ballot from $\mathsf{BB}^0_{\mathsf{vote}}$ to $\mathsf{BB}^0_{\mathsf{cast}}$.

Tally outcome: oracle $Otally()$ outputs the result of tallying the cast ballots, ensuring that the ideal world ($\beta = 1$) gets a perfect tally.

Privacy experiment and adversary: $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V},\Psi}(\lambda,\rho)$, with $\beta \in \{0,1\}$, is the privacy experiment described above. λ is the security parameter, ρ are parameters required by $\mathcal{V}$ and Ψ are platform and election assumptions. BB is a bulletin board following the structure defined by the voting scheme $\mathcal{V}$; it is empty at the beginning and filled with keys, credentials and ballots during the execution of the experiment.

Definition 1. A voting scheme $\mathcal{V}$ with platform and election assumptions Ψ satisfies vote privacy with bound ζ for a class of adversaries $\mathcal{C}$ if no adversary $\mathcal{A} \in \mathcal{C}$ can distinguish between the games $\mathsf{Exp}^{\mathsf{bpriv},0}_{\mathcal{A},\mathcal{V},\Psi}(\lambda,\rho)$ and $\mathsf{Exp}^{\mathsf{bpriv},1}_{\mathcal{A},\mathcal{V},\Psi}(\lambda,\rho)$ from Figure 2 with advantage more than $\zeta(\lambda)$. That is, for any election parameters ρ, and $\mathsf{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V},\Psi}(\lambda,\rho)$ defined by

$$\left| \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathcal{V},\Psi}(\lambda,\rho) = \beta \mid \beta \leftarrow_{\$} \{0,1\} \right] - \frac{1}{2} \right|$$

we have $\mathsf{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V},\Psi}(\lambda,\rho) < \zeta(\lambda)$

C. Attacks

We describe how our definition captures three attacks against voting schemes: the attack of Cortier and Smyth based on ballot copy [21], a variant of this attack by Rønne [47] based on the observation that ballot weeding is not sufficient to counter the attack of [21], and a new attack against the OnePad scheme of Figure 1. The attacks of [21], [47] are against the Helios voting scheme [1]. For brevity, we show how our definition captures them for a simplified version of the scheme. The new attack we describe against OnePad implies in particular that the scheme of [3] and Prêt à Voter [14], [49] do not satisfy our proposed privacy definition when the voting platform is compromised. They may satisfy weaker notions of privacy, for instance when the voting platform is honest but curious. We stress the novelty of our definition: among the three attacks, only the first one can be captured with previous definitions.

1) Attacks violating the definition: Consider a scheme where the Setup creates a key pair $(\mathsf{pk},\mathsf{sk})$ for an encryption scheme, Vote computes $\mathsf{Enc}_{\mathsf{pk}}(v)$ and Tally considers the cast ciphertexts, decrypts them and outputs the result. For this scheme, we consider the platform assumption $(\bot,\top)$, meaning that the private credentials are not leaked - in fact, there are no private credentials for this scheme - but the adversary can cast any ballot (for example, the adversary may control the channel or the server that uploads the ballot to the bulletin board). The election assumption $(\Phi\mathsf{v},\Phi\mathsf{c})$ does not allow revoting, as in Example 1, but may allow corruption after vote.

The attack of [21] assumes one corrupted voter, but we note that it is not necessary to control the voter, being sufficient to control the voting platform. The intuition of the attack is as follows: assume two registered voters id_1, id_2 that attempt to cast their ballots b_1 and b_2; the attacker discards b_2 and, instead, submits a copy of b_1 in the name of id_2 to the bulletin board. The outcome of the election will then contain two identical votes, which reveal the choice of id_1.

Example 2 (attack of [21] against Helios)**.** *Formally, for two different votes v_1, v_2, $\mathcal{A}$ does:*

$$Oreg(id_1)\ ;\ Oreg(id_2)\ ;$$
$$b \leftarrow Ovote(id_1, v_1, v_1)\ ;\ Ovote(id_2, v_2, v_2)\ ;$$
$$Ocast(id_1, 1, b)\ ;\ Ocast(id_2, 1, b)$$

We then have:

$$\mathsf{BB.cast} \;=\; (\ id_1, \mathsf{Enc}_{\mathsf{pk}}(v_1)\ ), (\ id_2, \mathsf{Enc}_{\mathsf{pk}}(v_1)\ )$$
$$\mathsf{BB}^0_{\mathsf{cast}} \;=\; (\ id_1, \mathsf{Enc}_{\mathsf{pk}}(v_1)\ ), (\ id_2, \mathsf{Enc}_{\mathsf{pk}}(v_2)\ )$$

When $\beta = 0$, $Otally()$ returns $\{v_1, v_1\}$. When $\beta = 1$, $Otally()$ returns $\{v_1, v_2\}$. Therefore $\mathcal{A}$ can determine β with probability 1.

For the next example, we strengthen the scheme by considering non-malleable encryption, ensuring that $\mathcal{A}$ cannot construct a ballot b'_1 encoding the same vote as b_1, and ballot weeding, ensuring two copies of b_1 cannot be accepted in tally. The strengthened scheme satisfies the privacy definition of [21], however, as observed by [47], it suffers from the same type of attacks when the attacker controls parts of the voting infrastructure. Indeed, if the attacker controls the ballot casting application or the bulletin board, it can claim that ballot casting was not successful. The voter may then try a second attempt with the same vote v_1. So the attacker obtains two different ballots b_1, b'_1 containing both v_1: one can be cast for id_1, and the other for id_2, so we derive the same attack as above.

Our definition captures such an attack as follows: (i) we consider a weaker election assumption $\Phi\mathsf{v}$ that allows multiple calls to the $Ovote$ oracle for the same id - this corresponds to a voter being allowed to revote if he believes his ballot did not make it to the bulletin board; (ii) the $Ocast$ oracle allows the adversary to exploit the list of ballots created with $Ovote$; some can be cast for the id corresponding to the same voter, others can be copied and cast for a different id.

Example 3 (attack of [47] against Helios with weeding)**.** *Adversary $\mathcal{A}$ registers the same voters as in Example 2. The other oracle queries are:*

$$b_1 \leftarrow Ovote(id_1, v_1, v_1); b'_1 \leftarrow Ovote(id_1, v_1, v_1);$$
$$Ovote(id_2, v_2, v_2); Ocast(id_1, 1, b_1);\ Ocast(id_2, 1, b'_1)$$

As in Example 2, the outcome of the tally for these ballots will allow $\mathcal{A}$ to detect β with probability 1.

Next we present an attack against schemes where the ballot is of the form $v \oplus_m w$, for some secret w that may associated to each voter - like in OnePad - or to each voter-candidate pair - like in Prêt-à-Voter with general permutations; a permutation π can be represented by a list of shifts $w_1, \ldots, w_m$, one for each candidate. Again, the platform assumption is $(\bot,\top)$: leakage $\bot$ means that private credentials used for computing the vote shift are not revealed to $\mathcal{A}$, unless the voter is corrupted; ballot modification $\top$ means $\mathcal{A}$ controls the device where the voter inputs the shifted vote and may arbitrarily modify it before casting. Assume there are m candidates and let $v_1, v_2 \in \mathbb{Z}_m$ with $v_1 \neq v_2$.

Intuitively, the attack works as follows: $\mathcal{A}$ exploits an algebraic property of a ballot $b = v \oplus_m w$, namely that $b' = b \oplus r = (v \oplus r) \oplus_m w$ represents a valid ballot encoding a vote for $v \oplus_m r$. $\mathcal{A}$ can substitute b with b' during ballot casting, so the output of the tally will then contain $v \oplus_m r$, instead of v as it should. We show, in the next section, that appropriately chosen values of r allow $\mathcal{A}$ to derive the vote of *both* the targeted voter and of a second voter whose machine $\mathcal{A}$ does not even have to control.

Example 4 (new attack on OnePad and PaV)**.** *Adversary $\mathcal{A}$ registers two voters id_1, id_2; it does not corrupt any. It modifies the ballot b of id_1 to cast $b \oplus_m r$ instead, for some $r \in \mathbb{Z}^*_m$. Formally, $\mathcal{A}$ does:*

$$Oreg(id_1);\ Oreg(id_2);$$
$$b_1 \leftarrow Ovote(id_1, v_1, v_1);\ b_2 \leftarrow\ Ovote(id_2, v_2, v_2);$$
$$Ocast(id_1, 1, b_1 \oplus_m r);\ Ocast(id_2, 1, b_2)$$

From definitions and the algebraic properties of $\oplus_m$, we have the following cast ballots to be tallied:

$$\begin{array}{rcl} \mathsf{BB.cast} & = & (id_1, (v_1 \oplus_m r) \oplus w_1)\ ,\ (id_2, v_2 \oplus_m w_2\ ); \\ \mathsf{BB}^0_{\mathsf{cast}} & = & (id_1, v_1 \oplus_m w_1\ )\ ,\ (id_2, v \oplus_m w_2\ ) \end{array}$$

When $\beta = 0$, $Otally()$ returns $\{v_1 \oplus_m r, v_2\}$. When $\beta = 1$, $Otally()$ returns $\{v_1, v_2\}$. Since $r \neq 0$, $\mathcal{A}$ can determine β with probability 1.

Note that the attack of Example 4 does not affect the ballot of id_2; we can assume $\mathcal{A}$ does not control the associated device.

2) Concrete attacks: We show how attacks discovered with the formal definition can be exploited in a concrete execution of the voting scheme, allowing to extract information about private votes when they follow a particular distribution. Assume there are two honest voters id_1, id_2 and 4 candidates, represented by elements of $\mathbb{Z}_4$. For illustration, we consider that the votes v_1 (of id_1) and v_2 (of id_2) follow the distribution

$$p_0 = \frac{1}{2} \qquad p_1 = \frac{1}{2} \qquad p_2 = 0 \qquad p_3 = 0$$

where p_i represents the probability of a vote for candidate i. The exploits can also be performed for more general distributions where $p_2, p_3 > 0$. This reflects the realistic situation where some candidates are expected to finish with very low scores. The goal of $\mathcal{A}$ is to find out how one, or both, of the honest voters voted. On an ideal system where $\mathcal{A}$ sees only a permutation of $\{v_1, v_2\}$ as outcome, $\mathcal{A}$ can win with probability $\frac{3}{4}$. Indeed, with probability $\frac{1}{2}$ both honest voters vote for the same candidate and, trivially, the attacker knows their choice, or (again with probability $\frac{1}{2}$) both voters make different choices (either v_1 votes 0 and v_2 votes 1, or v_1 votes 1 and v_2 votes 0), and the attacker has probability $\frac{1}{2}$ to guess their choices correctly.

For the concrete voting schemes that we consider, we derive from Examples 2–4 that $\mathcal{A}$ can win with probability 1, rather than probability $\frac{3}{4}$.

Helios. Following the steps of Example 2 or 3, $\mathcal{A}$ obtains $\{v_1, v_1\}$ as outcome of the election, so it can derive the vote of id_1 with probability 1—even without assuming a particular vote distribution. Note that $\mathcal{A}$ derives no information about the vote of id_2. The attack requires that $\mathcal{A}$ is able to drop and replace the ballot of id_2.

OnePad. Following the steps of Example 4, with $r = 1$, $\mathcal{A}$ obtains

$$\{\rho_1, \rho_2\} = \{v_1 \oplus_4 1, v_2\}$$

as outcome of the election. According to the vote distribution assumption, we have that

$$(v_1, v_2) \in \{0,1\} \times \{0,1\}\ \ \&\ \ p_2 = 0\ \ \&\ \ p_3 = 0.$$

Hence $\mathcal{A}$ can make the following case analysis:

- case $\{\rho_1, \rho_2\} = \{2, \rho\}$ for some $\rho \in \{0,1\}$: as $p_2 = 0$, the vote for 2 must be the modified one, i.e. $v_1 \oplus_4 1 = 2$, and therefore $(v_1, v_2) = (1, \rho)$.
- case $\{\rho_1, \rho_2\} = \{1,1\}$: we have that $(v_1 \oplus_4 1, v_2) = (1,1)$ and hence $(v_1, v_2) = (0,1)$.
- case $\{\rho_1, \rho_2\} = \{0,1\}$: as $\{v_1 \oplus_4 1, v_2\} = \{0,1\}$, we either have that $(v_1, v_2) = (3,1)$ or $(v_1, v_2) = (0,0)$; as $p_3 = 0$ it must be that $(v_1, v_2) = (0,0)$.
- other cases are not possible, as $p_2 = p_3 = 0$.

Note that, with probability 1, $\mathcal{A}$ learns how *both* id_1 and id_2 voted, while it has to control only the device of id_1.

PaV with general permutations. Following the steps of Example 4, with $r = 2$, $\mathcal{A}$ obtains $\{\rho_1, \rho_2\} = \{v, v_2\}$ as outcome of the election, with $v = v_1 \oplus_4 2$. Given the above assumption about the distribution of votes, we have $v_1 \in \{0,1\}$, and therefore $v \in \{2,3\}$, so $\mathcal{A}$ can deduce that $v_2 = \{\rho_1, \rho_2\} \cap \{0,1\}$.

Note that $\mathcal{A}$ learns how id_2 voted, and no information about the vote of id_1, although $\mathcal{A}$ does not control the device of id_2 but the device of id_1.

In conclusion, the attacker controlling the voting device can attack vote privacy in [3], [14], [49] by corrupting a ballot before sending it to election servers. This will determine an algebraic relation in the outcome that can be exploited to derive information about the private vote.

3) Additional remarks on our definition: For our attacker model, there are two types of "attacks" against voting schemes that, by design, are tolerated by our definition: ballot blocking and ballot malleability. A *ballot blocking* attack prevents a proportion of ballots from being cast, in order to reduce the anonymity of the remaining votes in the outcome. For example, if there are two voters and one of them is blocked, this will reveal the vote of the remaining voter. In our definition, a blocked ballot in a real execution of the protocol (i.e. a call to $Ovote(id, \_, \_)$ not followed by a call to $Ocast(id, \_, \_)$) will be mirrored by a blocked ballot in the ideal execution, so ballot blocking alone is not sufficient for distinguishing real from ideal executions.

We make this choice for the following reasons. On one hand, this attack is not realistic, because anyone can publicly observe that a significant proportion of ballots does not reach the bulletin board, and the attack would be countered. On the other hand, there is no voting scheme that can prevent such an attack, whereas there are schemes that still provide a meaningful security guarantee in this setting - we need a definition that can evaluate them.

Ballot malleability happens when the adversary is able to transform a ballot into a different valid ballot, without necessarily affecting the underlying vote, and cast it for the same voter that created the ballot. This source of malleability, e.g. when it changes the ballot but not the vote it contains, does not necessarily lead to an attack. For example, in some schemes ballots rely on rerandomiseable cryptography for encryption, signing, or both, like in the BeleniosRF scheme [13]. Ballot malleability does not necessarily affect vote privacy, and is a priori tolerated by our definition - as long as $\mathcal{A}$ is not able to change the underlying vote, or to cast the rerandomised ballot for a different voter; both of these features do indeed lead to attacks, as shown above.

III. Provable privacy with cryptographic tokens

The attacks of Section II-C show that in order to obtain provable privacy for schemes based on a one time pad we need mechanisms to ensure that the padded vote is not modified before being cast on the bulletin board. One option is to rely on verifiability, which itself relies on several assumptions, some of them technical - e.g. appropriate verification devices and trustworthy bulletin board - some of them social - e.g. voters do check their ballots. Another option, that we explore in the rest of the paper, is to use cryptographic tokens attesting to the tallying authorities that the cast ballots have not been compromised. We first consider such tokens at an abstract level, in terms of the security properties that they provide, and then discuss in section IV what cryptographic primitives can be used for instantiation.

A. The TokPad voting scheme (Fig. 3)

We consider a scenario where the voter has access to two devices. The first device, that we call *browser*, is used for sending the ballot to the bulletin board. The second device, that we call *token*, is used for authenticating the ballot. The goal is to guarantee privacy even if the browser or the token are compromised, but not both. The voter has a pair of credentials (w, k). The credential w is used as a one time pad for masking the vote v, like in the OnePad scheme. We have $p = v \oplus_m w$, where m is the number of candidates. The credential k is used by the token for computing a tag $t = \mathsf{Tok}(p, k)$, which is an authentication code for p. The voting ballot that will be sent by the browser to the bulletin board is the pair (p, t). Ballot validation (algorithm Valid called before Tally) ensures that only ballots with verified tags are tallied. We consider a simple tally procedure where talliers decrypt the ciphertexts $\mathsf{Enc}_{\mathsf{pk}}(w)$ in order to get w and reverse the one-time pad to get v. The proof of privacy, however, holds for any tally procedure whose input/output behavior is the same as the simple tally. Indeed, we can use re-encryption mixes or homomorphic tally in order to distribute trust, as we show in Section IV.

Formally, the syntax and security requirement for the cryptographic tokens (definitions 2 and 3) are similar to message authentication codes [4]. Particular to our application is the number of token instances that the adversary can see before having to forge a token: it is the number of voting attempts using the same token device. We can assume that this number is small for honest voters, and indeed equal to 1 when revoting is not allowed. This improves the security guarantees that can be derived for TokPad, especially for some cryptographic instances that we discuss in Section IV, whose aim is to keep the tag length as short as allowed by security requirements.

Definition 2. A cryptographic token is defined by a parameter space $\mathcal{P}$, a message space $\mathcal{M}$, a key space $\mathcal{K}$, a tag space $\mathcal{T}$ and three algorithms $\mathsf{TKGen} : \mathcal{P} \mapsto \mathcal{K}$, $\mathsf{Tok} : \mathcal{M} \times \mathcal{K} \mapsto \mathcal{T}$ and $\mathsf{Ver} : \mathcal{T} \times \mathcal{M} \times \mathcal{K} \mapsto \{\mathsf{true}, \mathsf{false}\}$ such that, for any $\rho \in \mathcal{P}$ and $k \leftarrow \mathsf{TKGen}(\rho)$, we have $\forall m \in \mathcal{M}.\ \mathsf{Ver}(\mathsf{Tok}(m, k), m, k) = \mathsf{true}$.

Definition 3. Given a cryptographic token $\mathsf{T} = (\mathsf{TKGen}, \mathsf{Tok}, \mathsf{Ver})$ and an adversary $\mathcal{A}$, let $\mathsf{Forge}^{\mathsf{tsec,n}}_{\mathcal{A},\mathsf{T}}(1^\lambda)$ be the event that $\mathsf{Exp}^{\mathsf{tsec,n}}_{\mathcal{A},\mathsf{T}}(1^\lambda)$ from Fig. 4 returns true. We define the advantage of $\mathcal{A}$ making at most n token queries, as

$$\mathsf{Adv}^{\mathrm{tsec,n}}_{\mathcal{A},\mathsf{T}}(1^\lambda) = \Pr\left[\mathsf{Forge}^{\mathsf{tsec,n}}_{\mathcal{A},\mathsf{T}}(1^\lambda)\right].$$

B. Proof of privacy

The following theorem shows that TokPad satisfies vote privacy under certain platform and election assumptions. The case $(\gamma, \mu) = (\bot, \top)$ models the scenario of a malicious browser and honest token: the credential for the one-time pad and the token key are not leaked, while the adversary has full influence over what ballot to cast to the bulletin board. The case $(\gamma, \mu) = (\{2\}, \{2\})$ models the scenario of an honest browser and malicious token: the token key is leaked and the adversary can provide any tag for the ballot, but it does not have the credential for and cannot modify the one-time pad sent to the bulletin board. We use election assumptions $(\Phi\mathsf{v}, \Phi\mathsf{c})$ from Example 1 for Theorem 1 showing privacy of TokPad. Below, we sketch the main ideas and steps of the proof, which is fully formalized and machine-checked [54] in EasyCrypt.

For an encryption scheme Enc and an adversary $\mathcal{A}$, we denote by $\mathsf{Adv}^{\mathrm{ind\text{-}cpa}}_{\mathcal{A},\mathsf{Enc}}(1^\lambda)$ the advantage of $\mathcal{A}$ in the chosen plaintext indistinguishability experiment that defines the security of Enc: it quantifies the success of $\mathcal{A}$ in distinguishing $\mathsf{Enc}(m_0, \mathsf{pk})$ from $\mathsf{Enc}(m_1, \mathsf{pk})$, for any m_0, m_1 chosen by $\mathcal{A}$ [28], [40]. We say that Enc is *ind-cpa secure* if, for any ppt adversary $\mathcal{A}$, we have $\mathsf{Adv}^{\mathrm{ind\text{-}cpa}}_{\mathcal{A},\mathsf{Enc}}(1^\lambda) < \zeta(\lambda)$, for some negligible function ζ.

Theorem 1. *Let $\mathcal{V}$ be the TokPad voting scheme using an encryption scheme* Enc *and a cryptographic token* T. *Assume that*

- $\mathsf{Adv}^{\mathrm{ind\text{-}cpa}}_{\mathcal{A},\mathsf{Enc}}(1^\lambda)$ *is bounded by* ζ_{enc} *for any ppt adversary* $\mathcal{A}$ *;*
- $\mathsf{Adv}^{\mathrm{tsec},1}_{\mathcal{A},\mathsf{T}}(1^\lambda)$ *is bounded by* ζ_{tok} *for any ppt adversary* $\mathcal{A}$.

Then, for any ppt adversary $\mathcal{A}$, any set of eligible voters $\mathcal{I}$ and any number of candidates m:

1) Dishonest browser[1] : *with platform assumptions* $\Psi = (\bot, \top, \Phi\mathsf{v}, \Phi\mathsf{c})$*, we have*

$$\mathsf{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V},\Psi}(\lambda, \mathcal{I}, m) \leq 2 * |\mathcal{I}| * (\ \zeta_{\mathsf{enc}}(\lambda) + \zeta_{\mathsf{tok}}(\lambda)\ )$$

2) Dishonest token[2]: *with platform assumptions* $\Psi = (\{2\}, \{2\}, \Phi\mathsf{v}, \Phi\mathsf{c})$*, we have*

$$\mathsf{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V},\Psi}(\lambda, \mathcal{I}, m) \leq 2 * |\mathcal{I}| * \zeta_{\mathsf{enc}}(\lambda)$$

We sketch the proof for the case of a dishonest browser, the case of a dishonest token being similar, with the difference that ballot integrity does not depend on a cryptographic assumption, but is ensured by the honest browser. Let $\mathcal{A}$ be

[1]Lemma dish_browser in file TokpadProof_DishonestBrowser.ec from [54]
[2] Lemma dish_token in file TokpadProof_DishonestToken.ec from [54]

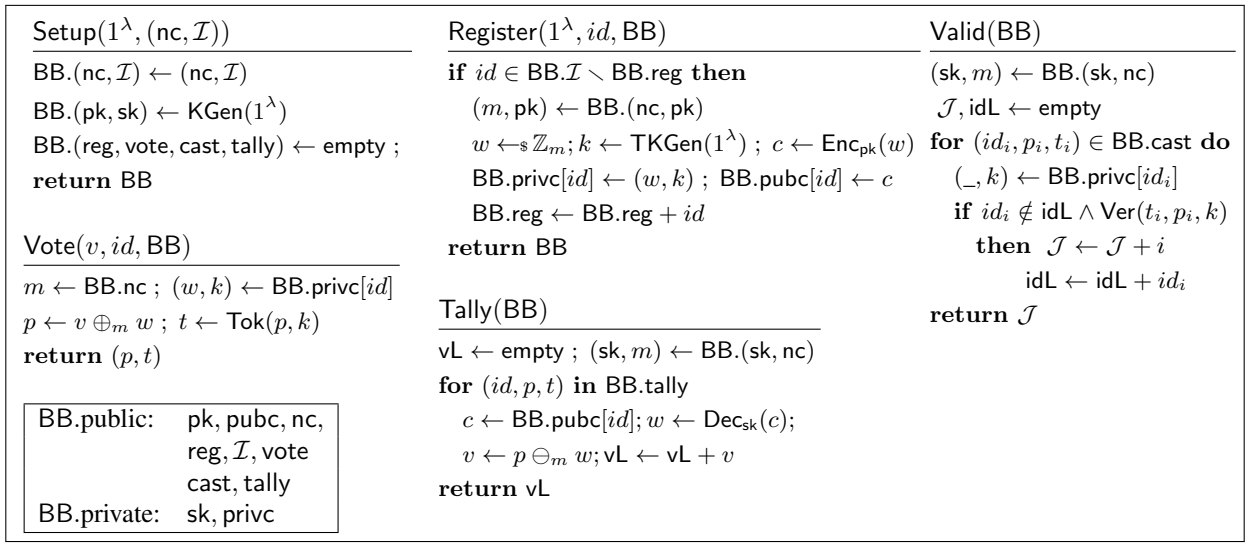

Fig. 3. Algorithms defining the TokPad voting scheme with encryption scheme (KGen, Enc, Dec) and token (TKGen, Tok, Ver)

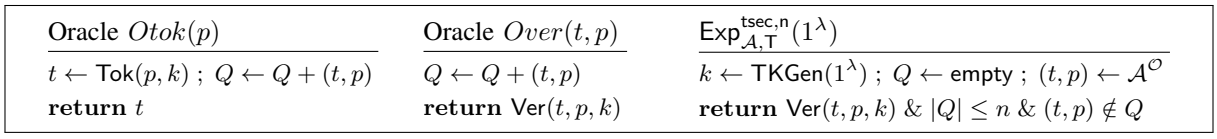

Fig. 4. Token security game with n token queries for adversary $\mathcal{A}$ and oracles $\mathcal{O} = \{Otok, Over\}$

a ppt adversary for the experiment $\text{Exp}^{\text{bpriv},\beta}_{\mathcal{A},\mathcal{V}}(\lambda)$ of Fig. 2. Let BB.cast $= (id_1, b_1), \ldots, (id_n, b_n)$ be the list of ballots present on the bulletin board after the ballot casting phase. We define a partition of these ballots into *honest*, *compromised* and respectively *corrupted*: BB.cast $= \text{BB}_{\text{hon}} \uplus \text{BB}_{\text{comp}} \uplus \text{BB}_{\text{corr}}$, where, for $(id, b) \in \text{BB}_{\text{cast}}$, we have

- $(id, b) \in \text{BB}_{\text{hon}}$ iff $id \notin$ corr and b was created by a call to $Ovote(id, v_0, v_1)$. So we have $b = \text{BB}_{\text{vote}}[id, i]$, where $i = 1$ because revoting is not allowed.
- $(id, b) \in \text{BB}_{\text{comp}}$ iff $id \notin$ corr and b is different from the ballot created with $Ovote(id, v_0, v_1)$. So we have $b \neq \text{BB}_{\text{vote}}[id, 1]$.
- $(id, b) \in \text{BB}_{\text{corr}}$ iff $id \in$ corr.

Denote by $\text{Bad}_{\mathcal{A}}$ the event that one of the ballots compromised by $\mathcal{A}$ passes the validity test, i.e. $\exists j.\ (id_j, b_j) \in \text{BB}_{\text{comp}}\ \&\ j \in \text{Valid(BB)}$. In the first part of the proof, relying on the token security assumption, we bound the probability of the $\text{Bad}_{\mathcal{A}}$ event. This allows us to assume $\neg\text{Bad}_{\mathcal{A}}$ for the second part of the proof, where we reduce vote indistinguishability to the security of the encryption scheme. Formally, consider the advantage of $\mathcal{A}$ conditioned on $\neg\text{Bad}_{\mathcal{A}}$:

$$\text{Adv}^{\text{bpriv}}_{\mathcal{A},\mathcal{V},\Psi,\neg\text{Bad}}(\lambda, \mathcal{I}, m) = \left|\Pr\left[\text{Exp}^{\text{bpriv},\beta}_{\mathcal{A},\mathcal{V},\Psi}(\lambda, \rho) = \beta \mid \beta \leftarrow_\$ \{0,1\}, \neg\text{Bad}_{\mathcal{A}}\right] - \tfrac{1}{2}\right|$$

We have:

Claim 1: $\Pr[\text{Bad}_{\mathcal{A}}] \leq 2 * |\mathcal{I}| * \zeta_{\text{tok}}(\lambda)$

Claim 2: $\text{Adv}^{\text{bpriv}}_{\mathcal{A},\mathcal{V},\Psi,\neg\text{Bad}}(\lambda, \mathcal{I}, m) \leq 2 * |\mathcal{I}| * \zeta_{\text{enc}}(\lambda)$

The desired result follows from

$$\text{Adv}^{\text{bpriv}}_{\mathcal{A},\mathcal{V},\Psi}(\lambda, \mathcal{I}, m) \leq \Pr[\text{Bad}_{\mathcal{A}}] + \text{Adv}^{\text{bpriv}}_{\mathcal{A},\mathcal{V},\Psi,\neg\text{Bad}}(\lambda, \mathcal{I}, m)$$

Proof sketch of Claim 1: Relying on $\mathcal{A}$, we construct an adversary $\mathcal{B}$ in the token security experiment $\text{Exp}^{\text{tsec}}_{\mathcal{B},\text{T}}$. $\mathcal{B}$ samples a voter $id \leftarrow_\$ \mathcal{I}$. If $\text{Bad}_{\mathcal{A}}$ is true, with some probability, as in equation $(\star)$ below, an honest ballot for the sampled id will be compromised by $\mathcal{A}$. This ballot allows $\mathcal{B}$ to break the security of the token for the corresponding key k_{id}. $\mathcal{B}$ implements the oracle calls for $\mathcal{A}$ as in the experiment $\text{Exp}^{\text{bpriv},\beta}_{\mathcal{A},\mathcal{V},\Psi}(\lambda)$, with the following difference:

- the token key k_{id} for id is not chosen by $\mathcal{B}$, but is the secret key in the token security experiment $\text{Exp}^{\text{tsec},n}_{\mathcal{B},\text{T}}$; note that $\mathcal{B}$ does not have this key.
- to create an honest ballot for id, $\mathcal{B}$ needs to compute Tok(p, k_{id}), for some p. For this, it relies on the $Otok(p)$ oracle provided by $\text{Exp}^{\text{tsec},n}_{\mathcal{B},\text{T}}$.

Note that: (i) $\mathcal{B}$ adds at most polynomial time overhead to the complexity of $\mathcal{A}$, thus it is a ppt adversary; (ii) at most one honest ballot is allowed for each voter, thus $\mathcal{B}$ makes at most one token query in the $\text{Exp}^{\text{tsec,n}}_{\mathcal{B},\text{T}}$ security game. Therefore $\mathcal{B}$ is in the class of token adversaries from the assumptions of the theorem and we have $\Pr\left[\text{Forge}^{\text{tsec},1}_{\mathcal{B},\text{T}}(1^\lambda)\right] \leq \zeta_{\text{tok}}(\lambda)$.

For each $id \in \mathcal{I}$, let $\text{Bad}^{id}_{\mathcal{A}}$ be the event that a ballot from the corresponding voter is a witness for $\text{Bad}_{\mathcal{A}}$, i.e. it is compromised and it passes the validity test. We can show that:

$$(\star)\ \Pr[\text{Bad}_{\mathcal{A}}] = \Pr\left[\bigvee_{id \in \mathcal{I}} \text{Bad}^{id}_{\mathcal{A}}\right] \leq |\mathcal{I}| * \Pr\left[\text{Bad}^{id}_{\mathcal{A}} \mid id \leftarrow_\$ \mathcal{I}\right]$$

Furthermore, for $\mathcal{B}$ constructed as above, we can show that:

$$(\star\star) \qquad \Pr\left[\mathsf{Forge}^{\mathsf{tsec},1}_{\mathcal{B},\mathsf{T}}(1^\lambda)\right] = \frac{1}{2} * \Pr\left[\mathsf{Bad}^{id}_{\mathcal{A}} \mid id \leftarrow_{\$} \mathcal{I}\right]$$

From $(\star)$, $(\star\star)$ and $\Pr\left[\mathsf{Forge}^{\mathsf{tsec},1}_{\mathcal{B},\mathsf{T}}(1^\lambda)\right] \leq \zeta_{\mathsf{tok}}(\lambda)$, we deduce Claim 1.

Proof sketch of Claim 2: Let $b_1, \ldots, b_n$ be the set of valid ballots cast by $\mathcal{A}$ for honest voters. Since we have $\neg\mathsf{Bad}_{\mathcal{A}}$, each b_i was constructed by an oracle call $Ovote(id_i, v^i_0, v^i_1)$. Therefore, the outcome of $Otally()$ for these ballots should be $v^1_0, \ldots, v^n_0$, and we can compute this output directly, without using the private credentials $w_1, \ldots, w_n$ for decrypting the one-time pad, by storing internally the votes provided in $Ovote$. We do this in a modified vote privacy experiment that we annotate with bpriv_1. We have

$$(\bullet)\ \mathsf{Adv}^{\mathsf{bpriv}}_{\mathcal{A},\mathcal{V},\Psi,\neg\mathsf{Bad}}(1^\lambda, \mathcal{I}, m) = \mathsf{Adv}^{\mathsf{bpriv}_1}_{\mathcal{A},\mathcal{V},\Psi,\neg\mathsf{Bad}}(1^\lambda, \mathcal{I}, m)$$

Note that the tally outcome for honest voters in bpriv_1 is determined by arguments to $Ovote$ and is independent of the private voter credentials. Next we rely on the security assumption for Enc to also make the public credentials independent of the private credentials. We transform bpriv_1 into bpriv_2 as follows:

- for every private credential $w_i \in \mathbb{Z}_m$, sample a fresh random $x_i \in \mathbb{Z}_m$;
- replace every public credential $\mathsf{Enc}_{\mathsf{pk}}(w_i)$ with the fresh ciphertext $\mathsf{Enc}_{\mathsf{pk}}(x_i)$.

Relying on a hybrid argument and the security assumption on Enc, we can show:

$$(\bullet\bullet) \quad \mathsf{Adv}^{\mathsf{bpriv}_1}_{\mathcal{A},\mathcal{V},\Psi,\neg\mathsf{Bad}}(1^\lambda, \mathcal{I}, m) \leq \mathsf{Adv}^{\mathsf{bpriv}_2}_{\mathcal{A},\mathcal{V},\Psi,\neg\mathsf{Bad}}(1^\lambda, \mathcal{I}, m) + 2 * |\mathcal{I}| * \zeta_{\mathsf{enc}}(\lambda)$$

where the coefficient $|\mathcal{I}|$ comes from the fact that, in the worst case, we may need one hybrid game per voter to make the reduction to ind-cpa security. Let $(w_1 \oplus v^1_\beta, \mathsf{Tok}(w_1 \oplus v^1_\beta, k_1)), \ldots, (w_m \oplus v^m_\beta, \mathsf{Tok}(w_m \oplus v^m_\beta, k_m))$ be the set of ballots created with $Ovote$ in bpriv_2; the ballots $b_1, \ldots, b_n$ that are cast for honest voters are a subset of these ballots. By construction of bpriv_2, each w_i is sampled uniformly and is only used once in the experiment, for the computation of $w_i \oplus v^i_\beta$. Therefore, w_i acts as a one-time pad that hides v^i_β, and we can deduce:

$$\mathsf{Adv}^{\mathsf{bpriv}_2}_{\mathcal{A},\mathcal{V},\Psi,\neg\mathsf{Bad}}(1^\lambda, \mathcal{I}, m) = 0$$

Together with $(\bullet)$ and $(\bullet\bullet)$, we conclude Claim 2.

IV. Voting system from TokPad

An instance of TokPad is determined by the choice of the encryption scheme for credentials and of cryptographic tokens for ballot authentication. Together with the number of eligible voters, they will determine the security guarantees that can be derived from Theorem 1. For tokens, we review two standard constructions that can be instantiated with any choice for the size of the tag space. A larger size provides better security, and it can be used when communication channels can be established between token and browser. A smaller size can also be used when the only available channel is the voter.

In addition to ind-cpa security, in order to ensure vote privacy even when some talliers are dishonest, the encryption scheme needs to allow:

- ballot anonymization by *re-encryption mixnets* or by *homomorphic aggregation* of ballots.
- *distributed decryption* by a shared secret key.

We can use classic voting-friendly schemes like ElGamal or Paillier to support re-encryption mixnets and distributed decryption. In order to support homomorphic aggregation of ballots we propose a technique that allows to derive, given one-time pads and credentials encrypted with an additively homomorphic scheme, ciphertexts containing an appropriate encoding of votes.

A. Cryptographic primitives for tokens

PRP-based mac [4], [27]. Assume $\mathcal{F} : \mathcal{T} \times \mathcal{K} \mapsto \mathcal{T}$ is a function from a pseudorandom permutation family with key generation algorithm $\mathcal{G}$. Then we let:

- the message and tag space for the token be equal to $\mathcal{T}$
- TKGen be defined by $\mathcal{G}$
- $\mathsf{Tok}(m, k)$ be defined by $\mathcal{F}(m, k)$
- $\mathsf{Ver}(t, m, k)$ be true iff $\mathcal{F}(m, k) = t$.

Seeking constructions for $\mathcal{F}$ that allow flexibility in choosing the size of tags from the space $\mathcal{T}$, we can rely on format-preserving encryption [5], [10] to choose any suitable domain for $\mathcal{T}$:

Lemma 1 ([5], [10])**.** *For any finite domain $\mathcal{T}$, there is a key space $\mathcal{K}$ and a pseudorandom permutation family with functions $\mathcal{F} : \mathcal{T} \times \mathcal{K} \mapsto \mathcal{T}$.*

Thus, the probability of distinguishing any such function $\mathcal{F}$ from a random permutation is bounded by a negligible function ζ, for any ppt adversary. The following lemma shows that pseudorandomness of $\mathcal{F}$ implies security for the associated token:

Lemma 2 ([4], [27])**.** *For any ppt adversary $\mathcal{A}$ against a prp-based token T, we have*

$$\Pr\left[\mathsf{Forge}^{\mathsf{tsec,n}}_{\mathcal{A},\mathsf{T}}(\lambda)\right] \leq \zeta(\lambda) + \frac{1}{(|\mathcal{T}| - n)}$$

where ζ bounds the probability for a ppt adversary to distinguish the underlying prp from a random permutation.

The second term in the sum represents the probability for $\mathcal{A}$ to guess a valid tag from the remaining options after having made n token queries.

Hash-based mac [12], [33], [55]. For a prime q and any $a, b, m \in \mathbb{Z}_q$, define the function $h_{a,b}(m) = a \cdot m + b \mod q$. Then, using $h_{a,b}(\cdot)$ in the same way as $\mathcal{F}(\cdot, k)$ above, we can define a token with message and tag space $\mathbb{Z}_q$ and key space $\mathbb{Z}_q \times \mathbb{Z}_q$.

Lemma 3 ([55])**.** *For any adversary $\mathcal{A}$ against a hash-based token T with tag space $\mathcal{T}$, we have*

$$\Pr\left[\mathsf{Forge}^{\mathsf{tsec},1}_{\mathcal{A},\mathsf{T}}(1^\lambda)\right] \leq \frac{1}{|\mathcal{T}|-1}.$$

The advantage of a hash-based mac is that it is a simple algebraic operation and can be easily verified. The disadvantage is that it can be used at most once, but this is sufficient for TokPad.

From Theorem 1 (for the case of a dishonest browser) and Lemma 2 (resp. Lemma 3), we derive the following Corollary, where Lemma 1 allows us to choose any suitable value for the size of the tag space in the case of a prp-based token:

Corollary 1. *The TokPad scheme using an ind-cpa secure encryption scheme and a (prp, resp. hash)-based token with tag space $\mathcal{T}$ satisfies privacy with bound $2{*}n_v{*}(\zeta(\lambda)+\frac{1}{|\mathcal{T}|-1})$, where n_v is the number of eligible voters and ζ is a negligible function.*

Assuming, for example, that token tags are 8 letter words built from 64 alphanumeric characters, we obtain $|\mathcal{T}| = 64^8 = 2^{48}$. If there are a maximum of 100 million voters, so $n_v < 2^{27}$, we derive from Corollary 1 that TokPad satisfies privacy with bound $\zeta(\lambda)+\frac{1}{2^{20}-1} < 0.001\cdot 10^{-4}$. If we have 1 million voters, so $n_v < 2^{20}$, and we use 6 letter words, so $|\mathcal{T}| = 2^{36}$, the bound is smaller than $0.002\cdot 10^{-2}$. Note that these bounds, in particular the coefficient n_v, are for the worst case where the adversary attacks privacy by attempting to massively corrupt tokens. In that case, from the token security assumption, a large proportion of the corresponding corrupted tags would be invalid. That can be considered enough corruption evidence in order to cancel the election, as for coercion evidence in [30].

B. Encryption scheme and homomorphic tally

To ensure privacy in case of dishonest talliers voting schemes rely on distribution of the secret key [17], [22], [46], combined with mixnets [1], [15], [38], [49] or homomorphic tallying [18], [19], [22], [36] to anonymize ballots. ElGamal [25] or Paillier [45] are examples of schemes typically used in this context, that we can also adopt in TokPad. Homomorphic tallying, however, requires a special encoding of encrypted votes in order to exploit the additive homomorphism of the encryption scheme. The initial ballot data in TokPad is a one-time pad of votes, hidden from voting devices, so we need an additional transformation to compute the necessary vote encoding. We propose a solution that can be applied to any homomorphic encryption scheme, that allows us to compute the vote encoding from available public data.

Consider Enc_0 to be an additively homomorphic ind-cpa secure encryption scheme, e.g. exponential ElGamal [25] or Paillier [45], with message space $\mathbb{Z}_q$, for a prime q, key space denoted by $\mathcal{K}_0$ and ciphertext space denoted by $\mathcal{C}_0$. Note that, from $z \in \mathbb{Z}_q$ and $c = \mathsf{Enc}'_{\mathsf{pk}}(x)$, we can compute $\mathsf{Enc}'_{\mathsf{pk}}(x*z)$. Denote by c^z this operation, and by $c_1 \star c_2$ the operation that allows to obtain $\mathsf{Enc}'_{\mathsf{pk}}(x_1{+}x_2)$ from $c_1 = \mathsf{Enc}'_{\mathsf{pk}}(x_1)$ and $c_2 = \mathsf{Enc}'_{\mathsf{pk}}(x_2)$. Assume there are m candidates, represented by elements of $\mathbb{Z}_m$. Let L be a number greater than the number of eligible voters. For each $\ell \in \mathbb{Z}_m$, a vote for ℓ will be encoded by an element in $\{L^\ell, L^{\ell-m}\}$. All algebraic operations in the following are implicitly modulo q.

Vote encoding. For any $\alpha, \ell, r \in \mathbb{Z}_q$, we let

$$\begin{array}{rcl} \mathsf{Encode}_L(\alpha,\ell) & = & \alpha L^\ell \\ \mathsf{Decode}_L(r,\ell) & = & (r \bmod L^{\ell+1} - r \bmod L^\ell)/L^\ell \end{array}$$

A vote for v is usually encoded as $\mathsf{Encode}_L(1, v)$. To achieve our transformation, we will consider $\mathsf{Encode}_L(1, v-m)$ to be an encoding of v as well. Then, α votes for v are represented by $\mathsf{Encode}_L(\alpha_1, v) + \mathsf{Encode}_L(\alpha_2, v-m)$, with $\alpha = \alpha_1 + \alpha_2$. The following lemma shows how additive operations on encodings translate to respective operations on the underlying votes.

Lemma 4. *Let $\alpha_1, \ldots, \alpha_k, \ell_1, \ldots, \ell_k, L \in \mathbb{Z}_q$ be such that $\alpha_1 + \ldots + \alpha_k < L$ and $\ell_1 < \ldots < \ell_k$. Then, for any $j \in \{1, \ldots, k\}$, we have*

$$\begin{array}{rcl} \mathsf{Encode}_L(\alpha_1,\ell) + \mathsf{Encode}_L(\alpha_2,\ell) & = & \mathsf{Encode}_L(\alpha_1+\alpha_2,\ell) \\ \mathsf{Decode}_L(\Sigma_{i=1}^k \mathsf{Encode}_L(\alpha_i,i),\ell_j) & = & \alpha_j \end{array}$$

Proof. For any α_1, α_2, we have directly from definition $\mathsf{Encode}_L(\alpha_1,\ell) + \mathsf{Encode}_L(\alpha_2,\ell) = \alpha_1 L^\ell + \alpha_2 L^\ell = \mathsf{Encode}_L(\alpha_1+\alpha_2,\ell)$.

Let $r = \Sigma_{i=1}^k \mathsf{Encode}_L(\alpha_i, i) = \alpha_1 L^{\ell_1} + \ldots + \alpha_k L^{\ell_k}$ and consider any $j \in \{1, \ldots, k\}$. From the assumptions of the Lemma, we can deduce that

$$\begin{array}{rcl} \alpha_1 L^{\ell_1} + \ldots + \alpha_j L^{\ell_j} & < & L^{\ell_j+1} \\ \alpha_1 L^{\ell_1} + \ldots + \alpha_{j-1} L^{\ell_{j-1}} & < & L^{\ell_j} \end{array}$$

Therefore, for any $j \in \{1, \ldots, k\}$, we have

$$\begin{array}{rcl} r \mod L^{\ell_j+1} & = & \alpha_1 L^{\ell_1} + \ldots + \alpha_{j-1} L^{\ell_{j-1}} + \alpha_j L^{\ell_j} \\ r \mod L^{\ell_j} & = & \alpha_1 L^{\ell_1} + \ldots + \alpha_{j-1} L^{\ell_{j-1}} \end{array}$$

So we can conclude that, for any $j \in \{1, \ldots, k\}$, we have $\mathsf{Decode}_L(r, \ell_j) = \alpha_j$. □

For tallying, we will apply Lemma 4, where

- $(\ell_1, \ldots, \ell_k)$ will be the levels $(0, \ldots, m-1, 0-m, \ldots, (m-1)-m)$ at which votes are encoded: each vote v is encoded either at level v or at level $v-m$ (modulo q).
- $\alpha_1, \ldots, \alpha_k$ will be the number of votes encoded at each level $\ell_1, \ldots, \ell_k$.
- q will be the order of the additive group $\mathbb{Z}_q$, that underlies Enc'

Since L is chosen to be greater than the number of eligible voters (and there is at most one vote per voter), we will have $\alpha_1 + \ldots + \alpha_k < L$. In order to have $\ell_1 < \ldots < \ell_k$, it is sufficient to choose q such that $2m < q$ (typically q is a large prime easily satisfying this constraint).

The goal in the following is to come up with an encryption scheme for credentials w in TokPad that allows to publicly obtain encrypted encodings of votes v from the shifted vote $v \oplus_m w$.

Encryption scheme. We define the encryption scheme Enc with message space $\mathbb{Z}_m$, key space $\mathcal{K}_0$ and ciphertext space $\mathbb{Z}_q \times \mathcal{C}_0$ as follows

$$\mathsf{Enc}_{\mathsf{pk}}(x) \quad = \quad (L^{-x} \cdot r \bmod q \ , \ \mathsf{Enc}'_{\mathsf{pk}}(r^{-1})) \ , \quad r \leftarrow_{\$} \mathbb{Z}_q^*$$

Decryption can be performed by first decrypting the inner ciphertext, removing r from the first component, and computing the discrete logarithm of L^{-x}, since x comes from a small set $\mathbb{Z}_m$. However, we do not perform decryption of credentials directly, but, from $v \oplus_m w$ and $\mathsf{Enc}_{\mathsf{pk}}(w)$ we compute $\mathsf{Enc}'_{\mathsf{pk}}(\mathsf{Encode}_L(1, v))$. Tallying can then be performed relying on the homomorphic properties of $\mathsf{Enc}'/\mathsf{Dec}'$ and $\mathsf{Encode}/\mathsf{Decode}$ (Lemma 4).

Encryption switching. For $a \in \mathbb{Z}_m$ and $c = (c_1, c_2)$ a ciphertext constructed with Enc, we let $\mathsf{Switch}(a, c) = c_2^{a \cdot c_1}$.

Lemma 5. *For any $m \in \mathbb{N}$, any $v, w \in \mathbb{Z}_m$, we have*

$$\mathsf{Switch}(v \oplus_m w, \mathsf{Enc}_{\mathsf{pk}}(w)) = \mathsf{Enc}'(\mathsf{Encode}_L(1, v'))$$

for some $v' \in \{v, v - m\}$.

Proof. Let $\mathsf{Enc}_{\mathsf{pk}}(w) = (c_1, c_2)$. From definitions, we have $\mathsf{Switch}(v \oplus_m w, c) = c_2^{L^{v \oplus_m w} \cdot c_1} = \mathsf{Enc}'_{\mathsf{pk}}(r^{-1})^{L^{v \oplus_m w} \cdot L^{-w} \cdot r} = \mathsf{Enc}'_{\mathsf{pk}}(L^{v \oplus_m w - w}) = \mathsf{Enc}'_{\mathsf{pk}}(\mathsf{Encode}_L(1, v \oplus_m w - w))$, for some $r \in \mathbb{Z}_q$. Since $v, w \in \mathbb{Z}_m$, we deduce $v \oplus_m w - w \in \{v, v - m\}$, and we can conclude. □

As consequence of ind-cpa security for Enc', we have:

Lemma 6. Enc *is ind-cpa secure.*

Proof sketch. By contradiction, assume $\mathcal{A}$ is a successful adversary against Enc. We construct an adversary $\mathcal{A}'$ against Enc', by simulating the ind-cpa game for $\mathcal{A}$. Assume $\mathcal{A}$ requests a challenge ciphertext for a pair $(m_0, m_1) \in \mathbb{Z}_m^2$, in order to distinguish $\mathsf{Enc}_{\mathsf{pk}}(m_0)$ from $\mathsf{Enc}_{\mathsf{pk}}(m_1)$. Let $r_0 \leftarrow_{\$} \mathbb{Z}_q$ and let r_1 be such that $L^{-m_1} = L^{-m_0} \cdot r_0 \cdot r_1$. The pair $(r_0^{-1}, r_1^{-1}) \in \mathbb{Z}_q^2$ is sent by $\mathcal{A}'$ to its ind-cpa challenger; it obtains $\mathsf{Enc}'_{\mathsf{pk}}(r_\beta^{-1})$ in return; it returns $\mathsf{Enc}_{\mathsf{pk}}(m_\beta) = [\ L^{-m_0} \cdot r_0, \mathsf{Enc}'_{\mathsf{pk}}(r_\beta^{-1})\ ]$ to $\mathcal{A}$; it obtains β' from $\mathcal{A}$ and outputs β' as its own guess. Then the advantage of $\mathcal{A}'$ in distinguishing $\mathsf{Enc}'_{\mathsf{pk}}(r_0^{-1})$ from $\mathsf{Enc}'_{\mathsf{pk}}(r_1^{-1})$ is the same as the advantage of $\mathcal{A}$ in distinguishing $\mathsf{Enc}_{\mathsf{pk}}(m_0)$ from $\mathsf{Enc}_{\mathsf{pk}}(m_1)$, so we contradict the ind-cpa security of Enc'.

Corollary 2 (Homomorphic tally)**.** *For any* BB *obtained by interacting with oracles $\mathcal{O}$ in Figure 3 and any parameters L, q chosen as above, we have* $\mathsf{Result}(\mathsf{Tally}(\mathsf{BB})) = \mathsf{Tally}_{\mathsf{hom}}(\mathsf{BB})$*, where* Result *gathers the votes for each candidate in a vector of dimension m and* $\mathsf{Tally}_{\mathsf{hom}}(\mathsf{BB})$ *is defined by:*

$\mathsf{sk} \leftarrow \mathsf{BB.sk}$; $m \leftarrow \mathsf{BB.nc}$; $c_R \leftarrow \mathsf{Enc}'_{\mathsf{pk}}(0)$
for (id, p, t) **in** $\mathsf{BB.tally}$
  $c \leftarrow \mathsf{BB.pubc}[id]$; $c' \leftarrow \mathsf{Switch}(p, c)$; $c_R \leftarrow c_R \star c'$;
$R \leftarrow \mathsf{Dec}'_{\mathsf{sk}}(c_R)$;
for $i = 0 \ldots m - 1$
  $r_i \leftarrow \mathsf{Decode}_L(R, i) + \mathsf{Decode}_L(R, i - m)$
return $(r_0, \ldots, r_{m-1})$

From ind-cpa security of Enc, Theorem 1 (for the case of a dishonest browser), results in subsection IV-A and Corollary 2, we deduce:

Corollary 3. *The TokPad scheme with* $\mathsf{Tally}_{\mathsf{hom}}$ *used as tally function,* Enc *defined as above, and a (prp or hash)-based token, satisfies privacy with bound $2 * n_v * (\zeta(1^\lambda) + \frac{1}{|\mathcal{T}|-1})$, where n_v is the number of voters, $\mathcal{T}$ is the tag space and ζ a negligible function.*

C. Deployment, verifiability and open questions

We sketch possible deployment choices as well as verifiability steps that should be considered in order to ensure vote integrity in TokPad. In-depth study of both is left for future work. Recall that *end-to-end verifiability* can be split into *individual verifiability*, ensuring that ballots are cast as intended, and *universal verifiability*, ensuring that ballots are tallied as cast [2], [42].

To get individual verifiability we need: *browser verification*, to ensure the ballot (p, t) seen in the browser by the voter is the same as the one cast on the bulletin board; *token verification*, to ensure that the tag value t provided by the token is indeed $\mathsf{Tok}(p, k)$; *credential verification*, to ensure that the private credentials (w, k) received by the voter correspond to the ones used for tally and ballot validation. Browser verification could be performed relying on a separate device, e.g. the token, to check the state of the bulletin board. If the token function is a simple algebraic operation, e.g. the hash-based token presented in subsection IV-A, token verification could be done directly by the voter. Otherwise, it is a more general open problem. An alternative could be to use a signing scheme, where the public key could be used for token verification. The challenge then would be to transfer the signature from the token to the browser, as it would be too long to be copied by the voter. To verify the credential w, we can adapt the Benaloh challenge approach [6]. The voter could challenge the system to verifiably decrypt the public credential on the bulletin board. The voter would then need to obtain another credential, created dynamically or beforehand. We can use a similar approach to verify the key k, but a fresh key would need to be loaded on the token device if the voter has chosen to challenge it.

For universal verifiability, we need to consider *ballot validation* and *ballot tally*. For each voter id, a public counterpart of the corresponding token key k_{id} would allow talliers to output zero-knowledge proofs attesting that ballots are correctly validated. For tally verifiability, we can rely on standard zero-knowledge proofs. Furthermore, most operations of our homomorphic tally are based only on public data, thus minimizing the number of required proofs. We also need to ensure that the additional information revealed for verifiability does not compromise privacy. Formal proofs are future work, but we expect strategies similar to those used to prove privacy of Helios [16] to work.

One may also be interested in a stronger notion of privacy, receipt-freeness, where the adversary cannot learn the contents of a ballot even with cooperation from the corresponding voter.

In TokPad, for any votes v, v' and credential w, there exists w' such that $v \oplus_m w = v' \oplus_m w'$. Therefore, even if the voter reveals v and w to the adversary, there is no way of verifying that the ballot does encode a vote for v. This holds under the assumption that voters cannot verify their credentials; otherwise, voters could delegate verification abilities to the adversary, who could then be convinced that w is the correct credential. This creates an additional challenge if we want to achieve both verifiability and receipt-freeness.

A foremost deployment issue, and open question, is secure generation and distribution of the private credential w. Like in other systems [3], [14], [35], we can rely on an out of band channel or a separate trusted device. The browser should be deployed on a device that can communicate with the bulletin board. The token could be deployed on a restricted device, like hardware tokens typically used for banking, or on a general purpose device like a mobile phone. In order to control access to the bulletin board, to authenticate voters and to load the secret key on the token device, we can have on top of TokPad an infrastructure based on passwords and signatures like in Helios and Belenios [1], [13].

There are a few open questions related to the security and usability of secret keys. The tally key sk can be distributed among a set of talliers, if we assume the underlying homomorphic scheme allows it. For the token key, we need a distributed token verification algorithm, and secure distributed protocols for generating the key and loading it on the token device. Estimating the (in)security of reusing the same token key across several elections or protocols is another problem that would arise from practice. If the election allows several voter choices, several different credentials could be used to mask each choice. All of them could possibly be authenticated by a single token tag. Extending TokPad for this scenario and other, more expressive, election settings is another direction for future work.

V. Related work

Voting schemes for untrusted devices. In addition to schemes based on a one-time pad [3], [14], [49], already reviewed in subsection II-A, we have:

Pretty Good Democracy [34] relies on paper sheets that contain codes for each candidate and a confirmation code that voters will receive back from the system in order to get assurance that their ballot reached the bulletin board. Tallying relies on encrypted tables, plaintext equivalence tests and re-encryption mixes in order to decrypt the corresponding confirmation code for the received ballot and transform the encrypted codes into corresponding encrypted votes.

The *Norwegian voting protocol [26], [35]* considers confirmation codes that are different for each candidate (this improves verifiability, perhaps at the price of receipt-freeness), and it proposes more efficient ways of computing the codes: [35] is based on exponentiation and secret sharing of private exponents; [26] is based on strong proxy oblivious transfer. The drawback is that the voting platform learns the vote; only integrity is ensured when the machine is untrusted.

Du-vote [31] is a remote voting scheme that aims to achieve privacy and verifiability on untrusted voting platforms. It relies on dynamic electronic codes, hardware tokens and orchestration of probabilistic checks and zero-knowledge proofs to ensure that no party can cheat. Furthermore, it crops ciphertexts to introduce the voter in the loop without loosing usability. Still, the scheme is rather complex and it's security has not been formally proved. Indeed, [41] shows that under certain assumptions there are attacks against both privacy and verifiability.

Models and proofs for vote privacy. Our result is the first machine-checked proof of privacy in presence of dishonest voting platforms. There are lines of both symbolic [21], [23] and computational [7]–[9], [16], [52] models for vote privacy assuming an honest voting platform. The formalism that is closest to ours is the one of [16]: they have a game-based model of privacy and a complete EasyCrypt proof that privacy holds for many variants of Helios.

We show that privacy in OnePad is violated when voters do not verify their ballots. A similar problem is studied in [20], showing that the classical privacy definition of [7] is violated in absence of individual verifiability [18], [43]. To counter this problem (and also the problem of a dishonest voting device), [20] restricts the privacy definition so that privacy guarantees hold only when all voters have verified their ballot. Our definition covers more general settings, where privacy can be studied and guaranteed independently from verifiability.

To capture the case of a dishonest voting platform, the ballot secrecy definition of [51] allows the adversary to control the bulletin board, similarly to our $Ocast$ oracle. Our definition allows however more flexibility by taking platform assumptions into account. The indistinguishability experiment of [51] does not follow the real versus ideal world approach that we do. Instead, following the style of [7], it requires indistinguishability of two different executions of the scheme - left versus right - that differ following adversary's inputs to $Ovote$ and the *challenge ballots* it obtains as output. This type of definition requires restrictions on the considered executions, in order to ensure that the left and right worlds cannot be trivially distinguished from the outcome of the election. Specifically, [51] requires that the tallied bulletin board BB be balanced, i.e. for any vote v, the number of distinct challenge ballots contained in BB that encode a vote for v should be the same in the left execution as in the right execution. Because of this restriction, [51] fails to capture the attack of [47] against Helios (we do capture it in Example 3 of Section II-C). The reason is that the attack requires two different challenge ballots b, b' created by two calls to $Ovote(id, v_0, v_1)$ to be present on BB, which would then not satisfy the balanced condition. On the other hand, for certain tallying methods (as noted in [8] for left versus right indistinguishability notions [7]) and certain schemes, the definition of [51] is too strong, resulting in trivial attacks against the definition, that do not imply real attacks against the scheme. For example, if the scheme allows a (challenge) ballot b to be transformed into a different valid ballot b', the definition would be violated,

because b' would not be covered by the restriction to a balanced BB. This would prevent, on one hand, to capture meaningful attacks on OnePad and, on the other hand, to prove privacy for schemes with malleable ballots, such as TokPad with rerandomizable tokens, or the BeleniosRF scheme [13], which relies on rerandomizable ciphertexts to achieve privacy and receipt-freeness.

VI. Conclusion

Our definition specifies ideal properties of privacy-preserving voting schemes on untrusted devices. On one hand, it poses the strong requirement that honest votes should not be modified before tally: this is motivated by our attacks in subsection II-C, showing that private votes can be extracted from a compromised tally. On the other hand, the specification does not prevent ballot dropping: this is motivated by the observation that no current scheme prevents this, and that methods external to the scheme could be used to detect and counteract such attacks. These and other features of our definition can be adjusted to account for new schemes and voting contexts, e.g. when there is an assumption on the vote distribution that counteracts our attacks, or when ballot delivery can be ensured with high probability.

Our attacks exploit algebraic properties resulting from masking votes in a cyclic group. Other ways of masking votes, for example based on precomputed codes [34], may be less vulnerable to such attacks, but they rely on additional infrastructure and trust assumptions. To be a scheme usable in practice, TokPad has to address verifiability and usability questions of section V. It is, at the moment, a first step towards a voter-friendly, provably-secure, verifiable voting scheme with minimal trust assumptions on untrusted platforms.

Acknowledgment

The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC).

References

[1] Ben Adida. Helios: Web-based open-audit voting. In *17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348, 2008.

[2] Ben Adida and C. Andrew Neff. Ballot casting assurance. In *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06*, 2006.

[3] Michael Backes, Martin Gagné, and Malte Skoruppa. Using mobile device communication to strengthen e-voting protocols. In Sadeghi and Foresti [50], pages 237–242.

[4] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[5] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. Format-preserving encryption. In *Selected Areas in Cryptography, SAC 2009*, pages 295–312, 2009.

[6] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07*, 2007.

[7] Josh Daniel Cohen Benaloh. *Verifiable Secret-ballot Elections*. PhD thesis, Yale University, New Haven, CT, USA, 1987. AAI8809191.

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

[9] David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In *Advances in Cryptology - ASIACRYPT 2012*, pages 626–643, 2012.

[10] John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In *Topics in Cryptology - CT-RSA*, pages 114–130, 2002.

[11] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve Schneider, Sriramkrishnan Srinivasan, Vanessa Teague, Roland Wen, and Zhe Xia. A supervised verifiable voting protocol for the victorian electoral commission. In Manuel J. Kripp, Melanie Volkamer, and Rüdiger Grimm, editors, *5th International Conference on Electronic Voting 2012, (EVOTE 2012), Bregenz, Austria*, volume 205 of *LNI*, pages 81–94. GI, 2012.

[12] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[13] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF: A non-interactive receipt-free electronic voting scheme. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1614–1625, 2016.

[14] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, pages 118–139, 2005.

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

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

[17] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed ElGamal à la Pedersen: Application to Helios. In Sadeghi and Foresti [50], pages 131–142.

[18] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for Helios under weaker trust assumptions. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, pages 327–344, 2014.

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

[20] Véronique Cortier and Joseph Lallemand. Voting: You can't have privacy without individual verifiability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 53–66. ACM, 2018.

[21] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[22] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *Transactions on Emerging Telecommunications Technologies*, 8(5):481–490, 1997.

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

[24] Easycrypt: Computer-aided cryptographic proofs. https://www.easycrypt.info/.

[25] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 10–18, 1984.

[26] Kristian Gjøsteen. The norwegian internet voting protocol. In *E-Voting and Identity - VoteID 2011*, volume 7187 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[27] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Proceedings of CRYPTO*

84 on Advances in Cryptology, pages 276–288, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[28] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270 – 299, 1984.

[29] Laurent Grégoire. Comment mon ordinateur a voté à ma place, 2012. http://tnerual.eriogerg.free.fr/autovote.pdf.

[30] Gurchetan S. Grewal, Mark Dermot Ryan, Sergiu Bursuc, and Peter Y. A. Ryan. Caveat coercitor: Coercion-evidence in electronic voting. In *IEEE Symposium on Security and Privacy, SP 2013*, pages 367–381. IEEE Computer Society, 2013.

[31] Gurchetan S. Grewal, Mark Dermot Ryan, Liqun Chen, and Michael R. Clarkson. Du-vote: Remote electronic voting with untrusted computers. In *IEEE Computer Security Foundations Symposium, CSF 2015*, pages 155–169. IEEE Computer Society, 2015.

[32] Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors. *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece*, volume 6345 of *Lecture Notes in Computer Science*. Springer, 2010.

[33] Shai Halevi and Hugo Krawczyk. MMH: software message authentication in the gbit/second rates. In Eli Biham, editor, *Fast Software Encryption, FSE '97, Haifa, Israel*, volume 1267 of *Lecture Notes in Computer Science*, pages 172–189. Springer, 1997.

[34] James Heather, Peter Y. A. Ryan, and Vanessa Teague. Pretty good democracy for more expressive voting schemes. In Gritzalis et al. [32], pages 405–423.

[35] Sven Heiberg, Helger Lipmaa, and Filip van Laenen. On e-vote integrity in the case of malicious voter computers. In Gritzalis et al. [32], pages 373–388.

[36] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology - EUROCRYPT 2000*, pages 539–556. Springer, 2000.

[37] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pages 339–353, 2002.

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

[39] Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. Usability analysis of helios - an open source verifiable remote electronic voting system. In *Proc. Electronic Voting Technology Workshop / Workshop on Trustworthy Elections (EVT/WOTE'11)*, 2011.

[40] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[41] Steve Kremer and Peter B. Rønne. To du or not to du: A security analysis of du-vote. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 473–486. IEEE, 2016.

[42] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Gritzalis et al. [32], pages 389–404.

[43] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010.

[44] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *CCS 2001, ACM Conference on Computer and Communications Security*, pages 116–125, 2001.

[45] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[46] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology - EUROCRYPT '91*, pages 522–526, 1991.

[47] Peter B. Rønne. Private communication.

[48] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *IEEE Trans. Information Forensics and Security*, 4(4):662–673, 2009.

[49] Peter Y. A. Ryan and Steve A. Schneider. Prêt à voter with re-encryption mixes. In *ESORICS 2006, 11th European Symposium on Research in Computer Security*, pages 313–326, 2006.

[50] Ahmad-Reza Sadeghi and Sara Foresti, editors. *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany*. ACM, 2013.

[51] Ben Smyth. A foundation for secret, verifiable elections. *IACR Cryptology ePrint Archive*, 2018:225, 2018.

[52] Ben Smyth and David Bernhard. Ballot secrecy and ballot independence coincide. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 463–480, 2013.

[53] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security analysis of the estonian internet voting system. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 703–715. ACM, 2014.

[54] *Easycrypt code for privacy of TokPad*. https://gitlab.com/ec-evoting/tokpad.

[55] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.