THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

When Are Three Voters Enough for Privacy Properties?

Citation for published version:
Arapinis, M, Cortier, V & Kremer, S 2016, When Are Three Voters Enough for Privacy Properties? in *Computer Security – ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II.* Lecture Notes in Computer Science, vol. 9879, Springer International Publishing, pp. 241-260, 21st European Symposium on Research in Computer Security, Crete, Greece, 26/09/16. https://doi.org/10.1007/978-3-319-45741-3_13

Digital Object Identifier (DOI):
10.1007/978-3-319-45741-3_13

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Computer Security – ESORICS 2016

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 29. Jul. 2021

When are three voters enough for privacy properties?

Myrto Arapinis[1], Véronique Cortier[2], and Steve Kremer[2]

[1] University of Edinburgh - Edinburgh, UK
[2] LORIA, CNRS & Inria Nancy & Université de Lorraine, France

Abstract. Protocols for secure electronic voting are of increasing societal importance. Proving rigorously their security is more challenging than many other protocols, which aim at authentication or key exchange. One of the reasons is that they need to be secure for an arbitrary number of malicious voters. In this paper we identify a class of voting protocols for which only a small number of agents needs to be considered: if there is an attack on vote privacy then there is also an attack that involves at most 3 voters (2 honest voters and 1 dishonest voter).
In the case where the protocol allows a voter to cast several votes and counts, e.g., only the last one, we also reduce the number of ballots required for an attack to 10, and under some additional hypotheses, 7 ballots. Our results are formalised and proven in a symbolic model based on the applied pi calculus. We illustrate the applicability of our results on several case studies, including different versions of Helios and Prêt-à-Voter, as well as the JCJ protocol. For some of these protocols we can use the ProVerif tool to provide the first formal proofs of privacy for an unbounded number of voters.

1 Introduction

Electronic voting has been adopted in several countries, such as the United States, Estonia, Australia, Norway, Switzerland, and France, to conduct legally binding elections (or at least trials for some of them). Electronic voting systems should ensure the same properties than the traditional paper ballots systems, despite the fact that malicious users may easily intercept ballots and try to forge fake ones. One crucial property is vote privacy: no one should know how a particular voter voted. Symbolic models have been very successful in the analysis of more traditional protocols that aim at confidentiality or authentication. Many decision techniques and several tools have been developed (see [1, 2, 3] to cite only a few) which have been successfully applied to a large number of case studies including widely deployed protocols such as TLS [4]. Vote privacy in symbolic models can be expressed through a rather simple and natural property [5]: an attacker should not be able to distinguish the situation where Alice votes 0 and Bob votes 1 from the situation where the votes are swapped:

$$V_{\mathsf{Alice}}(0) \mid V_{\mathsf{Bob}}(1) \approx V_{\mathsf{Alice}}(1) \mid V_{\mathsf{Bob}}(0)$$

Despite its apparent simplicity, this property is difficult to check for several reasons. Firstly, most existing decision techniques apply to reachability properties (such as authentication and confidentiality) but not to indistinguishability properties. Another major difficulty comes from the fact that e-voting systems involve less standard cryptographic primitives and sometimes even specially designed, ad-hoc primitives (like

for the protocol used in Norway [6]). Typical primitives in e-voting are homomorphic encryption, zero-knowledge proofs, reencryption mixnets, etc. Some techniques and tools [7, 8, 9, 10] for indistinguishability properties have recently been developed to automatically check indistinguishability properties and some of them can handle part of the primitives needed in e-voting. For example, ProVerif and Akiss have both been successfully applied to analyse some voting protocols [5, 10, 11, 12, 13, 14]. However, a third source of difficulty is the fact that voting systems are typically parametrized by the number of voters: both the bulletin board and the tally processes have to process as many ballots as they receive. This is typically modeled by considering processes parametrized by the number of voters. Even though parameterized protocols can be encoded in a formalism such as the applied pi calculus, such encodings are complicated and generally beyond the capabilities of what automated tools support. ProVerif, which to the best of our knowledge is the only tool that supports verification of indistinguishability properties for an unbounded number of sessions (i.e. allowing replication) generally fails to prove vote privacy. One exception is a case study of the Civitas voting system by Backes et al. [11] using ProVerif. The other tools for indistinguishability (e.g. SPEC [8], Akiss [10], and APTE [9]) can only handle a finite number of sessions. So case studies have to consider a finite number of voters [10, 12, 13, 14] unless proofs are conducted by hand [13, 15].

Contributions. Our main contribution is a reduction result for a reasonnably large class of voting protocols. If there is an attack on privacy for n voters, we show that there also exists one that only requires 3 voters: 2 honest voters are necessary to state the privacy property and then 1 dishonest is sufficient to find all existing attacks. This result significantly simplifies security proofs: there is no longer need to consider arbitrarily many voters, even in manual proofs. Moreover, this result allows the use of automated tools for checking equivalence properties and justifies previous proofs conducted for a fixed number of voters (provided at least one dishonest voter was considered).

Several protocols assume voters may revote several times. This is for example the case of Helios or Civitas. Revoting is actually crucial for coercion-resistance in Civitas. When revoting is allowed, this should be reflected in the model by letting the ballot box accept an unbounded number of ballots, and retaining only the valid ones according to the revote policy. This aspect is typically abstracted in any existing formal analysis. We show that we can simplify the analysis by reducing the total number of ballots to 10 for typical revoting policies (e.g. the last vote counts) and typical tally functions. Altogether, our result amounts in a finite model property: if there is an attack on privacy on n voters that may vote arbitrarily, then there is an attack that only requires 3 voters and at most 10 ballots. We can further reduce the number of ballots to 7 for a class of protocols that has *identifiable ballots*, that is ballots that reveal the corresponding public credentials. Of course, only 3 ballots are sufficient when revoting is disallowed.

Our result holds in a rather general setting provided that the e-voting system can be modeled as a process in the applied-pi calculus [16]. Of course, this reduction result cannot hold for *arbitrary* systems. For example, if the tally phase checks that at least 4 ballots are present then at least 4 voters are necessary to conduct an attack. So we model what we think to represent a "reasonable" class of e-voting systems. The process modeling the voter may be an arbitrary process as long as it does not depend on cre-

dentials of other voters and provided voters do not need to interact once the tally phase has started. This corresponds to the "vote and go" property, that is often desirable for practical reasons, but also excludes some protocols such as [17]. Once the vote is casted the authorities proceed as follows.

- The bulletin board (if there is one) performs only public actions such as publishing a received ballot, possibly removing some parts and possibly after some public tests, *i.e.* tests that anyone could do as well. Typical public tests are checks of signature validity, well-formedness of the ballots, or validity of zero-knowledge proofs. Alternatively, we may consider an arbitrary bulletin board in case it is corrupted since it is then part of the adversarial environment.
- Next, a revote policy is applied. We consider two particular revote policies: the policy which selects the last ballot, which is the most common one, and the policy that selects the first one, which encodes the situation where revoting is prohibited.
- Finally, the tally is computed according to some counting function. We consider in particular two very common functions: the multiset and the additive counting functions. The multiset counting function returns the votes in an arbitrary order and corresponds for example to the output of a decryption mixnet. The additive counting function returns the number of votes received by each candidate.

We believe that these conditions are general enough to capture many existing e-voting schemes.

Applications. To illustrate the applicability of our result, we re-investigate several existing analyses of e-voting protocols. First, we consider several versions of the Helios protocol [18], both in its mixnet and homomorphic versions. These versions also include the Belenios [19] protocol. We are able to use the ProVerif tool to show privacy for the mixnet versions of these protocols for a bounded number of voters and ballots. Our reduction result allows immediately to conclude that vote privacy also holds for an arbitrary number of voters. The homomorphic version of Helios is out of reach of existing tools due to the presence of associative and commutative symbols. However, our reduction result does apply, which means that the manual proof of Helios conducted in [13] did not need to consider arbitrarily many voters and could be simplified. In case one wishes to adapt this proof to Belenios [19], our reduction result would alleviate the proof. The Prêt-à-Voter [20] protocol (PaV) has been analysed using ProVerif for 2 honest voters [12]. Adding a third, dishonest, voter, we can apply our result and obtain the first proof of vote privacy for an arbitrary number of voters. Unfortunately, ProVerif did not scale up to verify automatically the protocol in presence of a dishonest voter. We were also able to apply our result (and a proof using ProVerif) to a protocol by Moran and Naor and to the JCJ protocol implemented in Civitas (without a ProVerif proof).

Related work. To our knowledge, the only other reduction result applying to voting protocols was proposed by Dreier et al. [21]. Their result states that it is sufficient to prove vote privacy for two honest voters when the protocol is observationally equivalent to a protocol consisting of the parallel composition (not sharing any secret) of a partition of the set of voters. Applicability has however only been shown to examples where this

trivially holds, e.g. [17, 22] as these protocols use completely public tallying mechanisms. In general, proving the required equivalence does not seem easier than proving directly vote secrecy. Moreover, it does not apply to some well known protocols such as Helios since a dishonest voter is needed to mount the vote replay attack [13].

The results of [23,24] show how to reduce the number of agents, in the case of trace properties [23] and equivalence properties [24]. The major difference with our work is that [23, 24] simply reduce the number of *agent identities* while the number of sessions (or processes) remains the same. In contrast, we do not only reduce the number of voter identities but also the number of ballots the ballot box needs to process, yielding a simpler process.

2 Modelling security protocols

As usual in symbolic protocol analysis we model protocol messages as terms. Protocols are modelled in a process calculus, similar to the applied pi calculus [16].

2.1 Messages

We assume an infinite set of *names* $\mathcal{N} = \{a, b, k, n, \ldots\}$ (which are used to represent keys, nonces, ...) and an infinite set of *channels* $\mathcal{C}h = \{c, c_1, ch, ch_1, \ldots\}$ (which are used to represent communication channels). We also consider a set of *variables* $\mathcal{X} = \{x, y, \ldots\}$, and a signature Σ consisting of a finite set of *function symbols*.

Terms are defined as names, variables, and function symbols applied to other terms. In particular, a channel is not a term. Let $\mathsf{N} \subseteq \mathcal{N}$ and $\mathsf{X} \subseteq \mathcal{X}$, the set of terms built from N and X by applying function symbols in Σ is denoted by $\mathcal{T}(\Sigma, \mathsf{N} \cup \mathsf{X})$. We write $fv(t)$ (resp. $fn(t)$) for the set of variables (resp. names) occurring in a term t. A term is *ground* if it does not contain any variable.

Example 1. We model asymmetric encryption, signatures, and pairs by the signature

$$\Sigma_{\mathsf{aenc}} \stackrel{\text{def}}{=} \{\mathsf{aenc}/3, \mathsf{adec}/2, \mathsf{pk}/1, \mathsf{sig}/2, \mathsf{checksig}/2, \mathsf{getmsg}/1, \mathsf{vk}/1, \langle\cdot,\cdot\rangle/2, \pi_1/1, \pi_2/1\}$$

where f/i denotes that f has arity i. Consider term $t \stackrel{\text{def}}{=} \langle \mathsf{pk}(sk), \mathsf{aenc}(\mathsf{pk}(sk), r, m)\rangle$ where $sk, r, m \in \mathcal{N}$. The term t represents a pair consisting of the public key $\mathsf{pk}(sk)$ associated to the private key sk and the encryption of message m with public key $\mathsf{pk}(sk)$ using randomness r. To improve readability, we may sometimes write $\langle t_1, \ldots, t_n\rangle$ instead of $\langle t_1, \langle \ldots \langle t_{n-1}, t_n\rangle \ldots\rangle\rangle$.

We denote by $\ell = [t_1, \ldots, t_n]$ the list of terms $t_1, \ldots, t_n$ and by $t_0 :: \ell$ the list obtained by adding the term t_0 to the head of the list, i.e., $t_0 :: \ell = [t_0, t_1, \ldots, t_n]$. Sometimes we interpret lists as multisets and we write $\ell_1 =^{\#} \ell_2$ for the equality of the multisets corresponding to these lists.

A *substitution* is a partial function from variables to terms. The substitution σ that maps x_i to t_i $(1 \leq i \leq n)$ is denoted $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ and we write $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$ for the domain of σ. We denote by $\emptyset$ the substitution whose

domain is empty. We always suppose that substitutions are acyclic. As usual we extend substitutions to terms and write $t\sigma$ for the application of σ to term t.

To model algebraic properties of cryptographic primitives, we define an *equational theory* by a finite set E of equations $u = v$ with $u, v \in \mathcal{T}(\Sigma, \mathcal{X})$. We define $=_{\mathsf{E}}$ to be the smallest equivalence relation on terms, that contains E and that is closed under application of function symbols and substitutions of terms for variables.

Example 2. Continuing Example 1 we define the equational theory $\mathsf{E}_{\mathsf{aenc}}$ by the following equations.

$$\mathsf{adec}(x_k, \mathsf{aenc}(\mathsf{pk}(x_k), x_r, x_m)) = x_m \qquad \mathsf{checksig}(\mathsf{sig}(x, y), \mathsf{vk}(y)) = \mathsf{ok}$$
$$\pi_i(\langle x_1, x_2 \rangle) = x_i \quad (i \in \{1, 2\}) \qquad \mathsf{getmsg}(\mathsf{sig}(x, y)) = x$$

Then we have that $\mathsf{adec}(sk, \pi_2(t)) =_{\mathsf{E}_{\mathsf{aenc}}} m$.

To illustrate our calculus we consider the Helios e-voting protocol as running example. The Helios protocol relies on zero knowledge proofs. We next specify the equational theory for the particular zero knowledge proofs built by the Helios participants.

Example 3. The Helios zero knowledge proofs can be modelled by the signature

$$\Sigma_{\mathsf{zkp}} \stackrel{\text{def}}{=} \{\mathsf{zkp}_{\mathsf{E}}/3, \mathsf{checkzkp}_{\mathsf{E}}/2, \mathsf{okzkp}_{\mathsf{E}}/0\} \cup \{\mathsf{zkp}^m_{\mathsf{DM}}/3, \mathsf{checkzkp}^m_{\mathsf{DM}}/3, \mathsf{okzkp}^m_{\mathsf{DM}}/0\}_{m \in \mathbb{N}}$$

In case of homomorphic tally, the voters should also prove that their vote is valid, which can be modeled in a similar way. When submitting an encrypted vote, voters are required to prove that the encryption is well-formed, that is to say, that they know the corresponding plaintext and randomness. This is reflected by the following equation.

$$\mathsf{checkzkp}_{\mathsf{E}}(\mathsf{zkp}_{\mathsf{E}}(xr, xv, \mathsf{aenc}(xpk, xr, xv)), \mathsf{aenc}(xpk, xr, xv)) = \mathsf{okzkp}_{\mathsf{E}}.$$

In the decryption mixnets-based variant of the Helios protocol, the talliers output a zero knowledge proof of correct mix and decryption. Such a proof establishes that the output of the decryption mixnet is indeed a permutation of the content of the encrypted ballots received as input. This is captured by the following infinite set of equations. For all $m \in \mathbb{N}$, and all $\{i_1, \ldots, i_m\} = \{1, \ldots, m\}$,

$$\mathsf{checkzkp}^m_{\mathsf{DM}}(\mathsf{zkp}^m_{\mathsf{DM}}(xk, xciph, xplain), xciph, xplain) = \mathsf{okzkp}^m_{\mathsf{DM}}$$

with $xciph = (\mathsf{aenc}(\mathsf{pub}(xk), xr_1, xv_1), \ldots, \mathsf{aenc}(\mathsf{pub}(xk), xr_m, xv_m))$ and $xplain = (xv_{i_1}, \ldots, xv_{i_m})$.

In all the examples of this section, we will consider the signature $\Sigma = \Sigma_{\mathsf{aenc}} \cup \Sigma_{\mathsf{zkp}}$ and the equational theory $\mathsf{E} = \mathsf{E}_{\mathsf{aenc}} \cup \mathsf{E}_{\mathsf{zkp}}$.

We say that a symbol $+$ is associative and commutative (AC in short) w.r.t. an equational theory E if E contains the two equations:

$$x + y = y + x \qquad x + (y + z) = (x + y) + z$$

2.2 Processes

We model protocols using a process calculus. Our *plain processes* are similar to plain processes in applied pi calculus [16] and are defined through the grammar given in Figure 1 where c is a channel, t, t_1, t_2 are terms, x is a variable, n is either a name or a channel, and $i \in \mathbb{N}$ is an integer. The terms t, t_1, t_2 may contain variables.

The process 0 does nothing. $P \mid Q$ behaves as the parallel execution of processes P and Q. $\nu n.P$ restricts the scope of n. When n is a name, it typically represents a freshly generated, secret value, *e.g.*, a key or a nonce, in P. When n is a channel, it declares a private channel, that cannot be accessed by the adversary. Replication $!P$ behaves as an unbounded number of copies of P. The conditional $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$ behaves as P if t_1 and t_2 are equal in the equational theory and as Q otherwise. The process $c(x).P$ inputs a message t on channel c, binds it to x and then behaves as P where x has been replaced by t. $\overline{c}\langle t\rangle.Q$ outputs message t on channel c before behaving as Q. Our calculus also introduces a *phase* instruction, in the spirit of [24, 25], denoted $i : P$. We denote by $Phase(P)$ the set of phases that appears in P, that is the set of j such that $j : Q$ occurs in P. By a slight abuse of notations, we write $Phase(P) < Phase(Q)$ if any phase in $Phase(P)$ is smaller than any phase in $Phase(Q)$.

$$
\begin{aligned}
P, Q := \ & 0 \\
& P \mid Q \\
& \nu n.P \\
& !P \\
& \mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q \\
& c(x).P \\
& \overline{c}\langle t\rangle.Q \\
& i : P
\end{aligned}
$$

Fig. 1. Syntax of plain processes

As usual, names and variables have scopes, which are delimited by restrictions and inputs. We write $fv(P)$, $bv(P)$, $fn(P)$ and $bn(P)$ for the sets of free and bound variables, and free and bound names of a plain process P respectively.

Example 4. A voter in Helios proceeds as follows. She computes her ballot by encrypting her vote with the public key $\mathsf{pk}(skE)$ of the election. The corresponding secret key is shared among several election authorities, which is not modeled here. Then she casts her ballot together with her identity and a zero knowledge proof through an authenticated channel. All this information will be published on a public bulletin board. The process $V(\mathsf{pk}(skE), cred, id, v)$ models the actions of a voter with identity id and credential $cred$ casting a ballot for candidate v:

$$V(\mathsf{pk}(skE), cred, id, v) \stackrel{\text{def}}{=} \nu r.\ \overline{bb}\langle\langle id, \mathsf{sig}(bal, cred), prf\rangle\rangle$$

where $bal = \mathsf{aenc}(\mathsf{pk}(skE), r, v)$ and $prf = \mathsf{zkp}_{\mathsf{E}}(r, v, bal)$. The authenticated channel is modelled by a signature although Helios relies on a login/password mechanism.

Extended processes keep track of additional information during an execution: the names that have been bound, the currently active processes that are running in parallel, the history of messages that were output by the process and the current phase.

Definition 1 (Extended process). *An* extended process *is a tuple* $(\mathcal{E}; \mathcal{P}; \Phi; i)$ *where:*

- *$\mathcal{E}$ is a set of names and channels that are restricted in $\mathcal{P}$ and Φ;*
- *$\mathcal{P}$ is a multiset of* plain processes *with $fv(\mathcal{P}) = \emptyset$;*
- *$\Phi = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$ is a ground substitution where $u_1, \ldots, u_n$ represent the messages previously output to the environment.*
- *i is an integer denoting the current phase.*

Example 5. The following extended process models two honest Helios voters id_A and id_B ready to cast their ballots v_A and v_B respectively in a first phase, and the Helios tallying authorities Tal ready to tally the cast ballots in a second phase

$$\mathsf{Helios}(v_A, v_B) \stackrel{\text{def}}{=} (\mathcal{E}_0, 1 : V_A \mid 1 : V_B \mid 2 : Tal, \emptyset, 1)$$

where $\mathcal{E}_0$ is a set of names with $cred_A, cred_B \in \mathcal{E}_0$,

$$V_A \stackrel{\text{def}}{=} V(\mathsf{pk}(skE), cred_A, id_A, v_A) \text{ and } V_B \stackrel{\text{def}}{=} V(\mathsf{pk}(skE), cred_B, id_B, v_B)$$

model the two honest voters where V is defined in Example 4, and

$$Tal \stackrel{\text{def}}{=} bb(xb_A).bb(xb_B).T$$

for some process T modelling the tallying authorities.

Given $A = (\mathcal{E}; \mathcal{P}; \Phi; i)$, we define the set of free and bound names of A as $fn(A) = (fn(\mathcal{P}) \cup fn(\Phi)) \smallsetminus \mathcal{E}$, and $bn(A) = bn(\mathcal{P}) \cup \mathcal{E}$. Similarly free and bound variables are defined as $fv(A) = (fv(\mathcal{P}) \cup \mathrm{dom}(\Phi))$, and $bv(A) = bv(\mathcal{P})$. An extended process A is closed if $fv(A) = \mathrm{dom}(\Phi)$.

The operational semantics of our calculus is defined by a labelled transition system which allows to reason about processes that interact with their environment. The transition relation $A \xrightarrow{\ell} B$ relates two ground extended processes A and B and is decorated by a label ℓ, which is either an input ($c(M)$), an output ($\nu x.\overline{c}\langle x\rangle$), or a silent action ($\tau$). Silent actions are standard, while visible input and output actions are interactions with the adversary on public channels. An output label $\nu x.\overline{c}\langle x\rangle$ reflects that messages are output "by reference": the label contains the variable added to $\mathrm{dom}(\Phi)$ which maps to the ground message that was output. The input label $c(M)$ contains the term M used by the adversary to compute the message: M may be constructed from previous outputs (adressed through variables in $\mathrm{dom}(\Phi)$), but is not allowed to use private names. The transition relation is formally defined in the companion technical report [26].

Notations. Given a set $\mathcal{S}$ we denote by $\mathcal{S}^*$ the set of all finite *sequences* of elements in $\mathcal{S}$. We may also write $\tilde{u}$ for the finite sequence $u_1, \ldots, u_n$. Let $\mathcal{A}$ be the alphabet of actions (in our case this alphabet is infinite and contains the special symbol τ). For every $w \in \mathcal{A}^*$, the relation $\xrightarrow{w}$ on processes is defined in the usual way, *i.e.*, we write $A \xrightarrow{w} A'$ when $w = \ell_1\ell_2 \cdots \ell_n$ and $A \xrightarrow{\ell_1} A_1 \xrightarrow{\ell_2} \ldots \xrightarrow{\ell_n} A'$. For $s \in (\mathcal{A} \smallsetminus \{\tau\})^*$, the relation $\stackrel{s}{\Rightarrow}$ on processes is defined by: $A \stackrel{s}{\Rightarrow} B$ if, and only if there exists $w \in \mathcal{A}^*$ such that $A \xrightarrow{w} B$ and s is obtained by erasing all occurrences of τ from w.

Example 6. Continuing our running example we illustrate the operational semantics by the following transitions

$$Helios(v_A, v_B) \xrightarrow{\nu y_A.\overline{bb}\langle y_A\rangle} \xrightarrow{\nu y_B.\overline{bb}\langle y_B\rangle} \xrightarrow{\mathsf{phase\ 2}} (\mathcal{E}; T; \Phi; 2) \quad \text{where}$$

- $\mathcal{E} = \mathcal{E}_0 \cup \{r_A, r_B\}$,
- $\Phi = \{y_A \mapsto \langle id_A, \mathsf{sig}(bal_A, cred_A), prf_A\rangle, y_B \mapsto \langle id_B, \mathsf{sig}(bal_B, cred_B), prf_B\rangle\}$ where $bal_C = \mathsf{aenc}(\mathsf{pk}(skE), r_C, v_C)$ and $prf_C = \mathsf{zkp_E}(r_C, v_C, bal_C)$ for $C \in \{A, B\}$.

A *frame* $\varphi = \nu\mathcal{E}.\Phi$ consists of a set of names $\mathcal{E}$ and a substitution $\Phi = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$. The names $\mathcal{E}$ are bound in φ and can be α-converted. Moreover names can be added (or removed) to (from) $\mathcal{E}$ as long as they do not appear in Φ. We write $\varphi =_\alpha \varphi'$ when frames φ and φ' are equal up to α-conversion and addition/removal of unused names. In this way two frames can always be rewritten to have the same set of bound names. When $A = (\mathcal{E}; \mathcal{P}; \Phi; i)$ is an extended process, we define $\phi(A) \stackrel{\text{def}}{=} \nu\mathcal{E}.\Phi$.

Given a frame $\varphi = \nu\mathcal{E}.\Phi$ an attacker can construct new terms building on the terms exposed by φ. For this the attacker applies a *recipe* on the frame. A recipe R for a frame φ is any term such that $fn(R) \cap \mathcal{E} = \emptyset$ and $fv(R) \subseteq \mathrm{dom}(\Phi)$. An attacker is unable to distinguish two sequences of messages if he cannot construct a test that distinguishes them. This notion is formally captured by *static equivalence* [16] of frames.

Definition 2 (Static equivalence). *Two frames $\varphi_1 =_\alpha \nu\mathcal{E}.\Phi_1$ and $\varphi_2 =_\alpha \nu\mathcal{E}.\Phi_2$ are statically equivalent, noted $\varphi_1 \sim \varphi_2$ when $\mathrm{dom}(\Phi_1) = \mathrm{dom}(\Phi_2)$, and for all recipes M and N of φ_1 we have that $M\Phi_1 =_\mathsf{E} N\Phi_1$ iff $M\Phi_2 =_\mathsf{E} N\Phi_2$.*

Note that in the above definition the frames φ_1 and φ_2 have the same set of recipes as they bind the same names $\mathcal{E}$ and their substitutions have the same domain.

Example 7. Let Φ be the substitution of Example 6 and

$$\Phi' = \{y_A \mapsto \langle id_A, \mathsf{sig}(bal'_A, cred_A), prf'_A\rangle, y_B \mapsto \langle id_B, \mathsf{sig}(bal'_B, cred_B), prf'_B\rangle\}$$

where $bal'_C = \mathsf{aenc}(\mathsf{pk}(skE), r_C, v_D)$ and $prf'_C = \mathsf{zkp_E}(r_C, v_D, bal'_C)$ for $C, D \in \{A, B\}$ with $C \neq D$. Since $\mathsf{adec}(skE, \pi_1(\pi_1(\mathsf{getmsg}(y_A))))\Phi =_\mathsf{E} v_A$, but $\mathsf{adec}(skE, \pi_1(\pi_1(\mathsf{getmsg}(y_A))))\Phi' \neq_\mathsf{E} v_A$, we have that

$$\nu skE.\nu r_A.\nu r_B.\Phi \sim_\mathsf{E} \nu skE.\nu r_A.\nu r_B.\Phi' \text{ while } \nu r_A.\nu r_B.\Phi \not\sim_\mathsf{E} \nu r_A.\nu r_B.\Phi'$$

Indeed, an attacker may distinguish between these two frames as soon as he has the secret key skE, by simply decrypting the ballots.

Given two extended processes A_1 and A_2, we often write $A_1 \sim A_2$ for $\phi(A_1) \sim \phi(A_2)$. Given an extended process A we define its set of traces as

$$\mathsf{traces}(A) \stackrel{\text{def}}{=} \{(tr, B) \mid A \stackrel{tr}{\Rightarrow} B\}$$

We can now define what it means for an attacker to be unable to *distinguish* two processes even if he is allowed to actively interact with them. This notion of indistinguishability is naturally modelled by *trace equivalence*.

Definition 3 (Trace equivalence). *Let A and B be two closed extended processes. A is trace included in B, written $A \sqsubseteq B$, if for every trace $(tr, A') \in \mathsf{traces}(A)$ there exists B' such that $(tr, B') \in \mathsf{traces}(B)$ and $A' \sim B'$. A and B are* trace equivalent*, denoted $A \approx B$, if $A \sqsubseteq B$ and $B \sqsubseteq A$.*

Intuitively, as the sequence of visible actions in the labels encode the adversary's actions the definition requires that for the same interaction with the adversary the protocols produce indistinguishable outputs.

3 Modelling e-voting protocols

In this section we explain how we formally model e-voting protocols and state the assumptions needed for our results.

Since many e-voting protocols use zero-knowledge proofs, we consider a signature Σ with $\mathsf{zkp}, \mathsf{checkzkp}, \mathsf{okzkp} \in \Sigma$ and we assume an equational theory that can be described by an AC-convergent (possibly infinite) rewrite theory such that the only rules in which zkp, checkzkp, and okzkp occur, are of the form:

$$\mathsf{checkzkp}(\mathsf{zkp}(U_1, \ldots, U_m), V_1, \ldots, V_n) \to \mathsf{okzkp}$$

where $\mathsf{zkp}, \mathsf{checkzkp}, \mathsf{okzkp}$ do not occur in the U_i, V_j. Since the terms U_i, V_j are left unspecified, this captures most existing zero-knowledge proofs. In particular, it covers the zero-knowledge proofs considered in Example 3.
A voting protocol is a family of processes $\{\Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})\}_{n_h,n_d,m\in\mathbb{N}}$ where

- n_h and n_d are the number of honest and dishonest voters respectively;
- $\mathcal{C}r^h_{n_h}$ (*resp.* $\mathcal{C}r^d_{n_d}$) is the set of n_h (*resp.* n_d) voting credentials which determines the set of honest eligible voters (*resp.* dishonest eligible voters), such that $\mathcal{C}r^h_{n_h} \cap \mathcal{C}r^d_{n_d} = \emptyset$. Each credential $\tilde{cr} \in \mathcal{C}r^h_{n_h} \cup \mathcal{C}r^d_{n_d}$ is a sequence of terms;
- m is the number of ballots accepted during the tally;
- $\mathcal{K}_{\mathsf{pv}}$ (*resp.* $\mathcal{K}_{\mathsf{pb}}$) is the set of all private (*resp.* public) material.

As usual it is sufficient to consider voting processes that model only the honest voters and the tally (the dishonest voters are left unspecified as part of the environment, and their credentials are public). We may assume w.l.o.g. that the tally process starts with a fresh phase and first reads the ballots on the board. Formally, we assume that voting processes are of the form:

$$\begin{aligned}\Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}) \stackrel{\text{def}}{=} {} & V(\tilde{cr}_1) \mid V(\tilde{cr}_2) \mid \cdots \mid V(\tilde{cr}_{n_h}) \mid \\ & \mathsf{tall} : bb(x_1).\ \ldots\ .bb(x_m).T^{n,m}(\mathcal{C}r_n, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})\end{aligned}$$

where $\mathcal{C}r_n = \mathcal{C}r^h_{n_h} \cup \mathcal{C}r^d_{n_d}$, and for all $i \in \{1, \ldots, n_h\}$, $\tilde{cr}_i \in \mathcal{C}r^h_{n_h}$. Furthermore, we require that $Phase(V) < \mathsf{tall}$, $Phase(T^{n,m}) = \emptyset$ and $T^{n,m}(\mathcal{C}r_n, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})$ contains at most one output which is performed on the channel tal. We note that from the above structure of a voting process it follows that all traces are prefixes of traces of the form

$$tr' \cdot \mathsf{phase\ tall} \cdot bb(RB_1) \ldots bb(RB_m) \cdot \nu y.\overline{tal}\langle y\rangle.$$

$V(\tilde{cr})$ models an honest voter, whose credentials are $\tilde{cr}$. $T^{n,m}(\mathcal{C}r_n, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})$ is the remainder of the tallier process. It is parameterised by the number m of ballots it accepts and the number n of eligible voters. We require that $V(\tilde{cr})$ be independent of n and m and does not use any other credentials, i.e. $fn(V(\tilde{cr})) \cap \mathcal{C}r_n \subseteq \{\tilde{cr}\}$. These are the only restrictions on the voter process and we believe them to be reasonable and natural.

An e-voting protocol proceeds in two phases: vote casting and tallying. During the vote phase all voters simply cast their ballots. The tally phase proceeds as follows. First

m ballots are input. Then a *public test* is applied to these ballots to carry out a first validity check, e.g. verify some zero knowledge proofs ensuring that the ballots are well formed. Next, the *revote policy* is applied to remove votes cast by a same voter, e.g., keep only the last one. Finally, the process performs the tally and outputs the result.

3.1 Public tests

As explained above, the ballot box may apply public tests to the casted ballots. Public tests are Boolean combinations over atomic formulas of the form $M = N$ where $M, N \in \mathcal{T}(\Sigma, \mathsf{X})$, *i.e.* they do not contain any names. An atomic formula is satisfied when $M =_{\mathsf{E}} N$ and we lift satisfaction to tests as expected.

We assume a family of tests $\{\mathsf{Test}^m\}_{m\in\mathbb{N}}$ where m is the number of casted ballots that are tested and Test^m contains m distinguished variables $x_1, \ldots, x_m$ to be substituted by the ballots. We write $\mathsf{Test}^m([B_1, \ldots, B_m]) = \top$ when the test $\mathsf{Test}^m\{x_1 \mapsto B_1, \ldots, x_m \mapsto B_m\}$ is satisfied. Finally we say that a *test is voting-friendly* whenever satisfaction is preserved on sublists of ballots, that is $\mathsf{Test}^m([B_1, \ldots, B_m]) = \top$ implies $\mathsf{Test}^h([B_{i_1}, \ldots, B_{i_h}]) = \top$ for any $1 \leq i_1 < \cdots < i_h \leq m$.

We believe this condition to be natural. It discards contrived tests that would accept a ballot only if another ballot is present. Conversely, we may consider tests that discard lists with duplicate ballots.

Example 8. The public test applied by the tallying authorities in the Helios protocol consists of two parts. First, a local test that checks the zero knowledge proofs of each submitted ballot, and second, a global test that checks that encrypted votes are pairwise distinct. This is to avoid the replay attack mentioned in [13]. Such checks are formally reflected by the family of tests $\{\mathsf{Test}^m\}_{m\in\mathbb{N}}$ with

$$Test^m([B_1, \ldots, B_m]) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{i=m} \mathsf{lTest}(B_i) \ \bigwedge_{i,j\in\{1,\ldots,m\}}^{i\neq j} \mathsf{gTest}(B_i, B_j)$$

$$\mathsf{lTest}(B) \stackrel{\text{def}}{=} \begin{cases} \top \text{ if } B = \langle id, bal, prf\rangle \text{ and } \mathsf{checkzkp_E}(\mathsf{getmsg}(bal), prf) =_{\mathsf{E}} \mathsf{okzkp_E} \\ \bot \text{ otherwise} \end{cases}$$

$$\mathsf{gTest}(B, B') \stackrel{\text{def}}{=} \begin{cases} \top \text{ if } B = \langle id, bal, prf\rangle \text{ and } B' = \langle id', bal', prf'\rangle \\ \quad \text{and } \mathsf{getmsg}(bal) \neq \mathsf{getmsg}(bal') \\ \bot \text{ otherwise} \end{cases}$$

3.2 Revote policies

Many e-voting protocols offer voters the possibility to cast several votes, keeping eventually only one vote per voter, e.g. the last submitted ballot. Which vote is kept depends on the particular policy. Re-voting intends to guarantee some protection against coercion. We formalize the notion of policy as a function $\mathsf{Policy}^{n,m}$ which takes a list of m terms (intuitively, the vote and credential) and a set of n credentials (honest and dishonest) and returns the sublist of selected terms to be tallied. A protocol will depend on a family of such policy functions $\{\mathsf{Policy}^{n,m}\}_{n,m\in\mathbb{N}}$. We consider two particular, but standard revote policies. The most usual one selects the last cast vote:

$$\mathsf{Policy}^{n,m}_{\mathsf{last}}([V_1, \ldots, V_m], \mathcal{C}r_n) \stackrel{\text{def}}{=} [V_{i_1}, \ldots, V_{i_k}]$$

where each $V_{i_j} = (v, \tilde{cr})$ is the last occurence of the credential $\tilde{cr} \in \mathcal{C}r_n$ in the list $[V_1, \ldots, V_m]$. We also consider the policy which only keeps the first vote of each voter:

$$\mathsf{Policy}^{n,m}_{\mathsf{first}}([V_1, \ldots, V_m], \mathcal{C}r_n) \stackrel{\text{def}}{=} [V_{i_1}, \ldots, V_{i_k}]$$

where each $V_{i_j} = (v, \tilde{cr})$ is the first occurence of the credential $\tilde{cr} \in \mathcal{C}r_n$ in the list $[V_1, \ldots, V_m]$. Such a policy typically models the norevote policy (a voter cannot revote).

3.3 Extracting ballots and counting votes

A voting protocol should tally the ballots "as expected". Formally, what is expected can be formalized through an *extract* and a *counting* function.

Given a ballot B, and two sets of terms $\mathcal{K}_{\mathsf{pb}}$ and $\mathcal{K}_{\mathsf{pv}}$ representing the public and private material, the extraction function $\mathsf{Extract}$ returns the corresponding vote and credential, or $\perp$ when a ballot is not well formed., i.e., $\mathsf{Extract}(B, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}) \in (\mathcal{V} \times \mathcal{C}r_n) \cup \{\perp\}$. Moreover, we lift the extract function to lists of m ballots by applying the function pointwise, i.e., $\mathsf{Extract}^m([B_1, \ldots, B_m], \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}) \stackrel{\text{def}}{=}$

$$[\mathsf{Extract}(B_1, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}), \ldots, \mathsf{Extract}(B_m, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})]$$

Similar extract functions have been introduced in [27] to define ballot privacy.

Example 9. The $\mathsf{Extract}$ function for the Helios protocol decrypts the encrypted vote and associates it with the signature associated to the ballot:

$$\mathsf{Extract}(B, \{skE\}, \{\mathsf{pk}(skE)\}) \stackrel{\text{def}}{=}$$
$$\begin{cases} (v, (id, cred)) & \text{if } B = \langle id, bal, prf \rangle \text{ and } bal =_{\mathsf{E}} \mathsf{sig}(\mathsf{aenc}(\mathsf{pk}(skE), r, v), cred) \\ \perp & \text{otherwise} \end{cases}$$

Similarly the counting function defines how the protocol is supposed to tally the votes. The function Count^ℓ takes as input a list of ℓ pairs $(v, cr) \in \mathcal{V} \times \mathcal{C}r$ and returns a list of terms as the election result.

Definition 4. *Let* $\{\mathsf{Count}^\ell\}_{\ell \in \mathbb{N}}$ *be a family of counting functions.* $\{\mathsf{Count}^\ell\}_{\ell \in \mathbb{N}}$ *is* voting-friendly *if for all* m, n *and lists of terms* W_1 *of size* m*,* W_2 *of size* n *we have that*

1. *if* $W_1 =^{\#} W_2$ *then* $\mathsf{Count}^m(W_1) =^{\#} \mathsf{Count}^n(W_2)$*;*
2. *if* $\mathsf{Count}^m(W_1) =^{\#} \mathsf{Count}^n(W_2)$
   then $\mathsf{Count}^{m+1}((v_1, cr_1) :: W_1) =^{\#} \mathsf{Count}^{n+1}((v_2, cr_2) :: W_2)$ *iff* $v_1 = v_2$

The first assumption requires that the result does not depend on the order in which votes are provided (intuitively, only valid votes are kept at this stage). We believe this property to be natural and it excludes contrived counting functions that would, e.g., only keep votes at even positions. The second assumption states that we may count "step by step". This is more restrictive since it excludes the majority function, i.e., the function that only outputs the name of the candidate that received most votes. But, it captures the most common result functions, namely the multiset and the additive counting functions.

Example 10. The multiset counting function typically arises in mixnet based tallies, which simply output the list of votes (intuitively once votes have been shuffled).

$$\mathsf{Count}^1_{\mathsf{Mix}}([V_1]) \stackrel{\text{def}}{=} [v] \text{ and } \mathsf{Count}^m_{\mathsf{Mix}}([V_1,\ldots,V_m]) \stackrel{\text{def}}{=} v :: Count^{m-1}_{\mathsf{Mix}}([V_2,\ldots,V_m])$$

where $V_1 = (v, \tilde{cr})$ and $m > 1$. The additive counting function can be defined similarly. For simplicity consider a binary vote, where we just want to count the number of 1's:

$$\mathsf{Count}^1_{\mathsf{HE}}([V_1]) \stackrel{\text{def}}{=} v \text{ and } \mathsf{Count}^m_{\mathsf{HE}}([V_1,\ldots,V_m]) \stackrel{\text{def}}{=} v + Count^{m-1}_{\mathsf{HE}}([V_2,\ldots,V_m])$$

where $V_1 = (v, \tilde{cr})$, $m > 1$ and $+$ is an AC symbol. Both functions are voting-friendly.

3.4 Properties

When verifying security properties of e-voting protocols it is common to only consider processes whose runs satisfy a particular property. For instance, vote secrecy is typically expressed as the indistinguishability of two processes modelling the situations where two honest voters swap their votes. We need however to ensure that these two honest voters have indeed cast their votes successfully to avoid trivial attacks. Indeed, in a run where the attacker blocks one of these voters, but not the other, the election result will be different and the two processes would be distinguished. Therefore when checking vote secrecy one typically adds a check that guarantees that the two honest votes are counted. We simply require that a *check* $\mathsf{check}([b_1,\ldots,b_m])$ applied to a list ballots $[b_1,\ldots,b_m]$ satisfies the two following requirements:

- If $\mathsf{check}([b_1,\ldots,b_m])$ holds then we can identify two (intuitively honest) ballots b_{i_1}, b_{i_2} such that check holds for any sublist containing b_{i_1} and b_{i_2}.
- If $\mathsf{check}([b_1,\ldots,b_m])$ does not hold then it does not hold either for any sublist of these ballots or if some ballots are replaced by invalid ones (that is replaced by $\perp$).

How such a check is implemented is left unspecified, it could be by listening to private channels, successively checking signatures, etc.

3.5 E-voting processes

As often when considering trace equivalence (e.g. [10, 24]), we assume processes to be deterministic. More precisely, we require the vote phase to be determinate: if the same sequence of labels leads to two different processes then the two resulting frames have to be statically equivalent. This typically holds for standard voting processes since the voter's behaviour is deterministic. For the tallying phase we slightly relax this notion and require what we call *almost determinate*. This relaxed notion only requires that there exists an output of a tally (among all possible outputs, as the particular tally may be chosen non-deterministically) that ensures static equivalence. This allows us to capture some non-deterministic behaviors such as mixnet tally.

Definition 5. *An e-voting protocol* $\{\Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})\}_{n_h,n_d,m\in\mathbb{N}}$ *is* almost determinate *if for any set of names* $\mathcal{E}_0$*, any initial attacker knowledge* Φ_0*, any*

$m, n_h, n_d \in \mathbb{N}$, *and any traces* $(tr, A_1), (tr, A_2) \in \mathsf{traces}(\mathcal{E}_0, \Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}), \Phi_0, 0)$ *we have that*

$$\forall A_1'.\ A_1 \xrightarrow{\nu x.\overline{tal}\langle x\rangle} A_1' \ \Rightarrow \exists A_2'.\ A_2 \xrightarrow{\nu x.\overline{tal}\langle x\rangle} A_2' \text{ and } A_1' \sim A_2'$$

We can now put all the pieces together and link e-voting protocols to the notions of public tests, revote policies, extraction and counting functions and properties.

Definition 6. *An e-voting protocol* $\{\Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})\}_{n_h,n_d,m\in\mathbb{N}}$ *is* voting friendly *w.r.t.* check, $\{\mathsf{Test}^m\}_{m\in\mathbb{N}}$, $\{\mathsf{Policy}^{n,m}\}_{n,m\in\mathbb{N}}$, Extract, $\{\mathsf{Count}^\ell\}_{\ell\in\mathbb{N}}$ *if it is almost determinate, if* $\{\mathsf{Test}^m\}_{m\in\mathbb{N}}$, $\{\mathsf{Policy}^{n,m}\}_{n,m\in\mathbb{N}}$, Extract, *are voting-friendly, and if for any set of names* $\mathcal{E}_0$, *any initial attacker knowledge* Φ_0, *any* m, n_h, n_d, *and any trace* $(tr'\cdot\nu x.\mathsf{phase}\ \mathsf{tall}.bb(RB_1)\ldots bb(RB_m), A_1)$ *of* $(\mathcal{E}_0, \Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}), \Phi_0, 0)$, *the resulting list of ballots* $BB = [B_1, \ldots, B_m]$ *(where* $B_i = RB_i\phi(A_1)$*) satisfies the following properties.*

1) *The tally is successful (that is* $(\nu y.\overline{tal}\langle y\rangle, A_2) \in \mathsf{traces}(A_1)$*) if and only if* BB *passes the test and the check (*$\mathsf{Test}^m(BB) = \top$ *and* $\mathsf{check}(BB) = \top$*)*

2) *Whenever the tally produces an output (that is* $(\nu y.\overline{tal}\langle y\rangle, A_2) \in \mathsf{traces}(A_1)$*) then it outputs a triple* $y\phi(A_2) = \langle res, nvotes, zkp\rangle$ *where*

- res *is the result computed by counting the votes once the extraction function and the revote policy have been applied on the bulletin board;*
- $nvotes$ *is the number of votes that has been counted;*
- zkp *is a (valid) zero-knowledge proof that would not be valid for any other list of ballots different from* BB*;*
- *either* res *is the only result the tally can produce from* BB *(typically in the homomorphic case) or the tally can produce any permutation of it (typically in the mixnet case).*

A fully formal definition can be found in the companion technical report [26]. We believe most existing protocols satisfy these requirements.

For many protocols ballots can be associated to the public credentials that were used to cast them. This is the case for Helios and some of its variants where ballots either contain the voter identity (in the original Helios) or are signed using private credentials (in the Belenios system). As we will see in the next section we can get tighter bounds for this class of protocols. Formally we define protocols with identifiable ballots as follows.

Definition 7. *An e-voting protocol* $\{\Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})\}_{n_h,n_d,m\in\mathbb{N}}$ *has* identifiable ballots *if for all* $n_h, n_d, m \in \mathbb{N}$, *for any trace*

$$(tr'\cdot\nu x.\mathsf{phase}\ \mathsf{tall}.bb(RB_1)\ldots bb(RB_m)\cdot\nu y.\overline{tal}\langle y\rangle, A)$$

of $\Pi^{n_h,n_d,m}(\mathcal{C}r^h_{n_h}, \mathcal{C}r^d_{n_d}, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})$) *there exists a recipe* R *and a variable* x *such that*

$$\forall 1 \le i \le m.\ \textit{if}\ \mathsf{Extract}([RB_i\phi(A)], \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}) = (V, \tilde{cr})\ \textit{then}\ R_i\phi(A) = \mathsf{pub}(\tilde{cr})$$

where $R_i = R\{x \mapsto RB_i\}$.

4 Main results

Throughout the section we consider two voting protocols

$$\{\Pi_i^{n_h,n_d,m}(\mathcal{C}r_{n_h}^h, \mathcal{C}r_{n_d}^d, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}})\}_{n_h,n_d,m\in\mathbb{N}}$$

for $1 \leq i \leq 2$ which are voting-friendly for check_i, $\{\mathsf{Test}^m\}_{m\in\mathbb{N}}$, $\{\mathsf{Policy}^{n,m}\}_{n,m\in\mathbb{N}}$, $\mathsf{Extract}_i^m$, $\{\mathsf{Count}_i^\ell\}_{\ell\in\mathbb{N}}$. Note that we assume the same public test for both protocols. Moreover we assume that $n_h \geq 2$ and $m \geq n_h + n_d$.

Let $\mathcal{E}_0$ be a set of names, and Φ_0 a ground substitution representing the initial attacker knowledge. $\{A_0^{n_h,n_d,m}\}_{n_h,n_d,m\in\mathbb{N}}$ and $\{B_0^{n_h,n_d,m}\}_{n_h,n_d,m\in\mathbb{N}}$ are two families of extended processes defined as follows

$$A_0^{n_h,n_d,m} \stackrel{\text{def}}{=} (\mathcal{E}_0 \cup \mathcal{C}r_{n_h}^h, \Pi_1^{n_h,n_d,m}(\mathcal{C}r_{n_h}^h, \mathcal{C}r_{n_d}^d, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}), \Phi_0, 0)\ \forall n_h, n_d, m \in \mathbb{N}$$
$$B_0^{n_h,n_d,m} \stackrel{\text{def}}{=} (\mathcal{E}_0 \cup \mathcal{C}r_{n_h}^h, \Pi_2^{n_h,n_d,m}(\mathcal{C}r_{n_h}^h, \mathcal{C}r_{n_d}^d, \mathcal{K}_{\mathsf{pv}}, \mathcal{K}_{\mathsf{pb}}), \Phi_0, 0)\ \forall n_h, n_d, m \in \mathbb{N}$$

Our reduction results apply to equivalences of the form $A_0^{n_h,n_d,m} \approx B_0^{n_h,n_d,m}$ for all m, n_h, n_d. Vote privacy is typically modelled in this way [5]. The proofs of the results presented in this section could not be included due to lack of space, but are available in the technical report [26].

Our first result states that attacks on such equivalences require at most 3 voters.

Proposition 1. *If $A_0^{k_h,k_d,\ell} \not\approx B_0^{k_h,k_d,\ell}$ then $A_0^{2,k'_d,\ell} \not\approx B_0^{2,k'_d,\ell}$ for $k'_d = 0$ or $k'_d = 1$.*

Note that this case does not yet bound the number of ballots to be considered. In particular, when re-voting is allowed the attacker may a priori need to submit several ballots in order to distinguish the two processes. In other words, the ballot box is still parameterized by the number of ballots to be received. However, whenever we assume that Π_1 and Π_2 do not allow voters to revote, we can deduce immediately that 3 ballots suffice to capture any attack. More formally, we encode this situation by letting $k = \ell$ and considering the re-vote policy that only keeps the first vote of each voter.

Theorem 1. *If $\{\mathsf{Policy}^{n,m}\}_{n,m\in\mathbb{N}} = \{\mathsf{Policy}_{\mathsf{first}}^{n,m}\}_{n,m\in\mathbb{N}}$ and $A_0^{k_h,k_d,k} \not\approx B_0^{k_h,k_d,k}$ where $k = k_h + k_d$, then $A_0^{2,k'_d,k'} \not\approx B_0^{2,k'_d,k'}$ for $k'_d = 0$ or $k'_d = 1$ and $k' = 2 + k'_d$.*

Intuitively, the case where $k'_d = 0$ corresponds to the case where an attacker can distinguish the processes playing only with two honest voters. This case for instance arises when analyzing a naive protocol where each voter simply signs his vote, hence offering no anonymity at all. The case where $k'_d = 1$ corresponds to the case where the attacker computes a vote which depends on the honest votes. The above results state that an attacker does not need more then one ballot in that case. An example of such an attack is the vote copy attack on Helios described in [13]. We could actually encode any attack with 2 voters into an attack with 3 voters by letting the adversary play like a useless, honest, voter. This would require however to formalize the fact that the attacker may always simulate an honest voter, that is, the voting process.

We now consider the case where re-voting is allowed. In this case we can bound the number of ballots that need to be considered to $4 + 2k$ (for k number of voters in total).

Proposition 2. *If* $A_0^{k_h,k_d,\ell} \not\approx B_0^{k_h,k_d,\ell}$*, then there exists* $\ell_{min} \leq 4 + 2k$ *such that* $A_0^{k_h,k_d,\ell_{min}} \not\approx B_0^{k_h,k_d,\ell_{min}}$ *where* $k = k_h + k_d$*.*

Combining the reductions on the number of voters and the number of ballots we obtain the following theorem.

Theorem 2. *If* $A_0^{k_h,k_d,\ell} \not\approx B_0^{k_h,k_d,\ell}$*, then there exists* $k'_d \in \{0,1\},\ \ell_{min} \leq 4+2k$ *such that* $A_0^{2,k'_d,\ell_{min}} \not\approx B_0^{2,k'_d,\ell_{min}}$ *where* $k = 2 + k'_d$*.*

This is an immediate consequence of Propositions 1 and 2 and yields a bound of 4+2×3=10. When protocols have identifying ballots (Definition 7) we can tighten our reduction of the number of ballots: we only need to consider $4 + k$ ballots.

Corollary 1. *If* Π_1 *and* Π_2 *have identifying ballots and* $A_0^{k_h,k_d,\ell} \not\approx B_0^{k_h,k_d,\ell}$*, then* $\exists \ell_{min} \leq 4 + k.\ A_0^{k_h,k_d,\ell_{min}} \not\approx B_0^{k_h,k_d,\ell_{min}}$ *where* $k = k_h + k_d$*.*

This is a corollary of the proof of Proposition 2. With identifiable ballots, we know that the ballots selected by the revoting policy on the left and on the right hand-side are the same. Again, we combine this result with the reduction on the number of voters.

Theorem 3. *If* Π_1 *and* Π_2 *have identifying ballots and* $A_0^{k_h,k_d,\ell} \not\approx B_0^{k_h,k_d,\ell}$ *then* $\exists k'_d \in \{0,1\},\ \ell_{min} \leq 4 + k$ *such that* $A_0^{2,k'_d,\ell_{min}} \not\approx B_0^{2,k'_d,\ell_{min}}$ *where* $k = 2 + k'_d$*.*

This follows from Corollary 1 and Proposition 1 and yields a bound of 4+3=7 ballots.

5 Case studies

We apply our results on several case studies: several versions of Helios [18, 19, 28] and Prêt-à-Voter [20], as well as the JCJ protocol [29] implemented in the Civitas system [30]. For some of these protocols we show that the ProVerif verification tool [1] can be used to perform a security proof that, thanks to our results, is valid for an arbitrary number of voters and ballots.

For the other protocols, ProVerif is not able to verify the protocols, either due to the fact that equational theories with AC symbols are not supported by ProVerif or simply because of a state explosion problem. In these cases we show that our results nevertheless apply. Given recent progress in automated verification for equivalence properties [9,10,31] we hope that verification of some of these protocols will be possible soon. Our results would also be useful to simplify proofs by hand.

The results in this section are summarized in Figure 2. Our hypotheses were always satisfied wherever applicable. For several protocols, we could not conduct the analysis with ProVerif, either because the equational theory is out of reach of the tool or because we had to stop ProVerif execution after a couple of hours. The case studies are further detailed in the companion report [26]. The results in this section rely on ProVerif scripts available at `http://3voters.gforge.inria.fr`.

	3 ballots (Theorem 1)	
	Hyp	ProVerif
PaV (DM)	✓	✓
PaV (RM)	✓	×
Helios mix (weeding)	✓	✓
Helios mix (id in zkp)	✓	✓
Helios hom (weeding)	✓	×
Helios hom (id in zkp)	✓	×
Belenios mix	✓	✓
Belenios hom	✓	×

(a) Protocols without revoting.

	7 ballots (Theorem 3)		10 ballots (Theorem 2)	
	Hyp	ProVerif	Hyp	ProVerif
Helios mix (weeding)	✓	✓	✓	×
Helios mix (id in zkp)	✓	✓	✓	×
Helios hom (weeding)	✓	×	✓	×
Helios hom (id in zkp)	✓	×	✓	×
Belenios mix	✓	✓	✓	×
Belenios hom	✓	×	✓	×
JCJ	✓	×	✓	×

(b) Protocols with revoting.

Fig. 2. Summary of application of our results on case studies. A × in the "ProVerif" column indicates that we could not successfully run the analysis with ProVerif.

6 Conclusion

In this paper we propose reduction results for e-voting protocols that apply to vote privacy. We believe they also apply to stronger properties such as receipt-freeness. Our first reduction result states that whenever there is an attack, there is also an attack with only two honest voters and at most one dishonest voter. This considerably simplifies the proofs and encodings otherwise needed to verify such protocols using automated verification tools. We moreover consider the case where the protocol allows a voter to cast multiple votes and selects one vote according to a given re-vote policy, e.g. select the last vote casted. In that case verifying privacy is still complicated even when restricted to three voters. We therefore show a second reduction result that allows to consider at most 10 ballots. In case the protocol has *identifiable ballots* we reduce the number of necessary ballots to 7. We have shown that the hypotheses of our theorems are satisfied by many protocols: several variants of Helios, Prêt-à-Voter, as well as Civitas. For several of these protocols we were able to apply automated tool verification and provide the first automated proofs for an unbounded number of voters and ballots. For the decryption mixnets-based PaV protocol, we even provide the first proof of vote privacy.

An interesting direction for future work is to further tighten the bound on the number of ballots, possibly characterizing properties enjoyed by voting protocols. We also foresee to show similar reduction results for other properties of e-voting, such as verifiability. Given that the result is stated in a symbolic model, we also plan to investigate if the result can be transposed to a computational model.

Acknowledgments. This work has received funding from the European Research Council (ERC) under the EU's Horizon 2020 research and innovation program (grant agreement No 645865-SPOOC) and the ANR project SEQUOIA ANR-14-CE28-0030-01.

References

1. B. Blanchet, "An efficient cryptographic protocol verifier based on Prolog rules," in *14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Comp. Soc., 2001, pp. 82–96.
2. M. Rusinowitch and M. Turuani, "Protocol insecurity with finite number of sessions is NP-complete," in *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Comp. Soc., 2001, pp. 174–190.
3. H. Comon-Lundh and V. Shmatikov, "Intruder deductions, constraint solving and insecurity decision in presence of exclusive or," in *Proc. 18th Annual Symp. on Logic in Computer Science (LICS'03)*. IEEE Comp. Soc., 2003, pp. 271–280.
4. K. Bhargavan, R. Corin, C. Fournet, and E. Zalinescu, "Cryptographically verified implementations for tls," in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, October 2008, pp. 459–468.
5. S. Kremer and M. D. Ryan, "Analysis of an electronic voting protocol in the applied pi-calculus," in *Proc. 14th European Symposium On Programming (ESOP'05)*, ser. LNCS, vol. 3444. Springer, 2005, pp. 186–200.
6. K. Gjøsteen, "Analysis of an internet voting protocol," Cryptology ePrint Archive, Report 2010/380, 2010.
7. B. Blanchet, M. Abadi, and C. Fournet, "Automated Verification of Selected Equivalences for Security Protocols," in *20th Symp. on Logic in Computer Science (LICS'05)*, Jun. 2005, pp. 331–340.
8. A. Tiu and J. E. Dawson, "Automating open bisimulation checking for the spi calculus," in *Proc. 23rd Computer Security Foundations Symp. (CSF'10)*. IEEE Comp. Soc., 2010, pp. 307–321.
9. V. Cheval, H. Comon-Lundh, and S. Delaune, "Trace equivalence decision: Negative tests and non-determinism," in *Proc. 18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM, Oct. 2011.
10. R. Chadha, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," in *21th European Symposium on Programming (ESOP'12)*, ser. LNCS, vol. 7211. Springer, Mar. 2012, pp. 108–127.
11. M. Backes, C. Hritcu, and M. Maffei, "Automated verification of remote electronic voting protocols in the applied pi-calculus," in *21st IEEE Computer Security Foundations Symp. (CSF'08)*. IEEE Comp. Soc., 2008, pp. 195–209.
12. M. Arapinis, S. Bursuc, and M. Ryan, "Reduction of equational theories for verification of trace equivalence: re-encryption and ac," in *First Conference on Principles of Security and Trust (POST'12)*, 2012, pp. 169–188.
13. V. Cortier and B. Smyth, "Attacking and fixing helios: An analysis of ballot secrecy," *Journal of Computer Security*, vol. 21, no. 1, pp. 89–148, 2013.
14. M. Arapinis, V. Cortier, S. Kremer, and M. D. Ryan, "Practical Everlasting Privacy," in *Proceedings of the 2nd Conferences on Principles of Security and Trust (POST'13)*, ser. LNCS, vol. 7796. Springer, Mar. 2013, pp. 21–40.
15. V. Cortier and C. Wiedling, "A formal analysis of the norwegian e-voting protocol," in *Proceedings of the 1st International Conference on Principles of Security and Trust (POST'12)*, ser. LNCS, vol. 7215. Springer, Mar. 2012, pp. 109–128.
16. M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Proc. 28th ACM Symp. on Principles of Programming Languages (POPL'01)*. ACM, 2001, pp. 104–115.
17. T. Okamoto, "Receipt-free electronic voting schemes for large scale elections," in *Proc. 5th Int. Security Protocols Workshop*, ser. LNCS, vol. 1361. Springer, 1997, pp. 25–35.

18. B. Adida, "Helios: web-based open-audit voting," in *17th conference on Security symposium (SS'08)*. USENIX Association, 2008, pp. 335–348. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496734
19. V. Cortier, D. Galindo, S. Glondu, and M. Izabachene, "Election verifiability for helios under weaker trust assumptions," in *19th European Symp. on Research in Computer Security (ESORICS'14)*, ser. LNCS, vol. 8713. Springer, 2014, pp. 327–344.
20. P. Y. A. Ryan and S. A. Schneider, "Prêt-à-voter with re-encryption mixes." in *11th European Symp. On Research In Computer Security (ESORICS'06)*, ser. LNCS, vol. 4189. Springer, 2006, pp. 313–326.
21. J. Dreier, P. Lafourcade, and Y. Lakhnech, "Defining privacy for weighted votes, single and multi-voter coercion," in *17th European Symp. On Research In Computer Security (ESORICS'06)*, ser. LNCS, 2012.
22. A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections." in *Advances in Cryptology – AUSCRYPT '92*, ser. LNCS, vol. 718. Springer, 1992, pp. 244–251.
23. H. Comon-Lundh and V. Cortier, "Security properties: two agents are sufficient," *Science of Computer Programming*, vol. 50, no. 1-3, pp. 51–71, March 2004.
24. V. Cortier, A. Dallon, and S. Delaune, "Bounding the number of agents, for equivalence too," in *Proceedings of the 5th International Conference on Principles of Security and Trust (POST'16)*, ser. LNCS. Springer, 2016.
25. B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, 2008.
26. M. Arapinis, Veronique, and S. Kremer, "When are three voters enough for privacy properties?" Cryptology ePrint Archive, 2016.
27. D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "A comprehensive analysis of game-based ballot privacy definitions," in *Proceedings of the 36th IEEE Symp. on Security and Privacy (S&P'15)*. IEEE Comp. Soc., May 2015, pp. 499–516.
28. P. Bulens, D. Giry, and O. Pereira, "Running mixnet-based elections with helios," in *2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11*. USENIX Association, 2011.
29. A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *ACM workshop on Privacy in the electronic society (WPES'05)*. ACM, 2005, pp. 61–70.
30. M. Clarkson, S. Chong, and A. Myers, "Civitas: Toward a secure voting system," in *29th IEEE Symp. on Security and Privacy (S&P'08)*. IEEE Comp. Soc., 2008, pp. 354–368.
31. V. Cheval and B. Blanchet, "Proving more observational equivalences with proverif," in *Proc. 2nd International Conference on Principles of Security and Trust (POST'13)*, ser. LNCS, vol. 7796. Springer, 2013, pp. 226–246.