Natural Strategic Abilities in Voting Protocols

Wojciech Jamroga[1,2], Damian Kurpiewski[2], and Vadim Malvone[3]

[1] SnT, University of Luxembourg
[2] Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
[3] Université d'Évry, France

Abstract. Security properties are often focused on the technological side of the system. One implicitly assumes that the users will behave in the right way to preserve the property at hand. In real life, this cannot be taken for granted. In particular, security mechanisms that are difficult and costly to use are often ignored by the users, and do not really defend the system against possible attacks.

Here, we propose a graded notion of security based on the complexity of the user's strategic behavior. More precisely, we suggest that the level to which a security property φ is satisfied can be defined in terms of (a) the complexity of the strategy that the voter needs to execute to make φ true, and (b) the resources that the user must employ on the way. The simpler and cheaper to obtain φ, the higher the degree of security.

We demonstrate how the idea works in a case study based on an electronic voting scenario. To this end, we model the vVote implementation of the Prêt à Voter voting protocol for coercion-resistant and voter-verifiable elections. Then, we identify "natural" strategies for the voter to obtain receipt-freeness, and measure the voter's effort that they require. We also look at how hard it is for the coercer to compromise the election through a randomization attack.

Keywords: electronic voting, coercion resistance, natural strategies, multi-agent models, graded security

1 Introduction

Security analysis often focuses on the technological side of the system. It implicitly assumes that the users will duly follow the sequence of steps that the designer of the protocol prescribed for them. However, such behavior of human participants seldom happens in real life. In particular, mechanisms that are difficult and costly to use are often ignored by the users, even if they are there to defend those very users from possible attacks.

For example, protocols for electronic voting are usually expected to satisfy *receipt-freeness* (the voter should be given no certificate that can be used to break the anonymity of her vote) and the related property of *coercion-resistance* (the voter should be able to deceive the potential coercer and cast her vote in accordance with her preferences) [7, 26, 17, 27]. More recently, significant progress has been made in the development of voting systems that would be coercion-resistant and at the same time *voter-verifiable*, i.e., would allow the voter to verify her part

of the election outcome [31, 12]. The idea is to partly "crowdsource" an audit of the election to the voters, and see if they detect any irregularities. Examples include the Prêt à Voter protocol [32] and its implementation vVote [13] that was used in the 2014 election in the Australian state of Victoria.

However, the fact that a voting system includes a mechanism for voter-verifiability does not immediately imply that it is more secure and trustworthy. This crucially depends on how many voters will actually verify their ballots [37], which in turn depends on how understandable and easy to use the mechanism is. The same applies to mechanisms for coercion-resistance and receipt-freeness, and in fact to any optional security mechanism. If the users find the mechanism complicated and tiresome, and they can avoid it, they will avoid it.

Thus, the right question is often not *if* but *how much* security is obtained by the given mechanism. In this paper, we propose a graded notion of *practical security* based on the complexity of the strategic behavior, expected from the user if a given security property is to be achieved. More precisely, we suggest that the level to which property φ is "practically" satisfied can be defined in terms of (a) the complexity of the strategy that the user needs to execute to make φ true, and (b) the resources that the user must employ on the way. The simpler and cheaper to obtain φ, the higher the degree of security.

Obviously, the devil is in the detail. It often works best when a general idea is developed with concrete examples in mind. Here, we do the first step, and look how the level of coercion-resistance and voter-verifiability can be assessed in vVote and Prêt à Voter. To this end, we come up with a multi-agent model of vVote, inspired by interpreted systems [18]. We consider three main types of agents participating in the voting process: the election system, a voter, and a potential coercer. Then, we identify strategies for the voter to use the voter-verifiability mechanism, and estimate the voter's effort that they require. We also look at how difficult it is for the coercer to compromise the election through a randomization attack [26]. The strategic reasoning and its complexity is formalized by means of so called *natural strategies*, proposed in [24, 25] and consistent with psychological evidence on how humans use symbolic concepts [8, 19].

To create the models, we use the Uppaal model checker for distributed and multi-agent systems [4], with its flexible modeling language and intuitive GUI. This additionally allows to use the Uppaal verification functionality and check that our natural strategies indeed obtain the goals for which they are proposed.

Related work. Formal analysis of security that takes a more human-centered approach has been done in a number of papers, for example with respect to insider threats [21]. A more systematic approach, based on the idea of *security ceremonies*, was proposed and used in [10, 5, 6, 29], and applied to formal analysis of voting protocols [28]. Here, we build on different modeling tradition, namely on the framework of *multi-agent systems*. This modeling approach was only used in [22] where a preliminary verification of the Selene voting protocol was conducted. Moreover, to our best knowledge, the idea of measuring the security level by the complexity of strategies needed to preserve a given security requirement is entirely new.

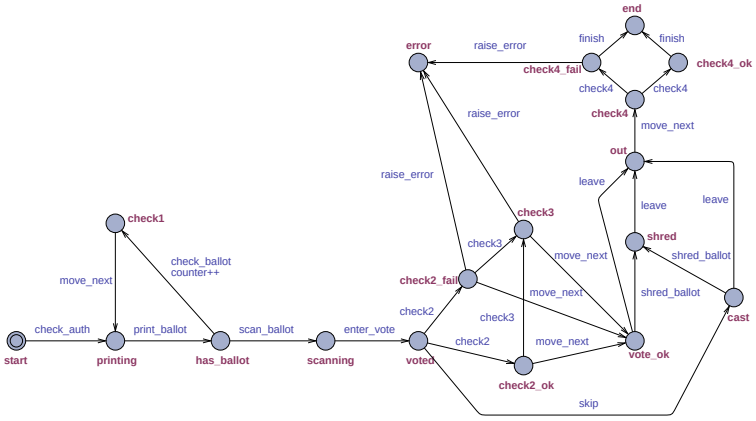

Fig. 1. Voter model

Other (somewhat) related works include social-technical modeling of attacks with timed automata [15] and especially game-theoretic analysis of voting procedures [9, 14, 2, 23]. Also, strategies for human users to obtain simple security requirements were investigated in [3]. Finally, specification of coercion-resistance and receipt-freeness in logics of strategic ability was attempted in [36].

2 Methodology

The main goal of this paper is to propose a framework for analyzing security and usability of voting protocols, based on how easy it is for the participants to use the functionality of the protocol and avoid a breach of security. Dually, we can also look at how difficult it is for the attacker to compromise the system. In this section we explain the methodology.

2.1 Modeling the Voting Process

The first step is to divide the description of the protocol into loosely coupled components, called agents. The partition is often straightforward: in our case, it will include the voter, the election infrastructure, the teller etc.

For each agent we define its local model, which consists of locations (i.e., the local states of the agent) and labeled edges between locations (i.e., local transitions). A transition corresponds to an action performed by the agent. An example model of the voter can be seen in Figure 1. When the voter has scanned her ballot and is in the state *scanning* she can perform action *enter_vote*, thus moving to the state *voted*. This model, as well as the others, has been created using the modeling interface of the Uppaal model checker [4]. The locations in Uppaal are graphically represented as circles, with initial locations marked by a double circle. The edges are annotated by colored labels: guards (green),

synchronizations (teal) and updates (blue). The syntax of expressions is like that of C/C++. Guards enable the transition if and only if the guard condition evaluates to true. Synchronizations allow processes to synchronize over a common channel. Update expressions are evaluated when the transition is taken.

The global model of the whole system consists of a set of concurrent processes, i.e., local models of the agents. The combination of the local models unfolds into a global model, where each global state represents a possible configuration of the local states of the agents.

2.2 Natural Strategic Ability

Many relevant properties of multi-agent systems refer to *strategic abilities* of agents and their groups. For example, voter-verifiability can be understood as the ability of the voter to check if her vote was registered and tallied correctly. Similarly, receipt-freeness can be understood as the inability of the coercer, typically with help from the voter, to obtain evidence of how the voter has voted [36].

Logics of strategic reasoning, such as ATL and Strategy Logic, provide neat languages to express properties of agents' behavior and its dynamics, driven by individual and collective goals of the agents [1, 11, 30]. For example, the ATL formula $\langle\langle cust \rangle\rangle \mathsf{F}\ \mathsf{ticket}$ may be used to express that the customer *cust* can ensure that he will eventually obtain a ticket, regardless of the actions of the other agents. Semantically, the specification holds if *cust* has a strategy whose every execution path satisfies ticket at some point in the future. Strategies in a multi-agent system are understood as conditional plans, and play central role in reasoning about purposeful agents [1, 35]. Formally, strategies are defined as functions from states to actions. However, real-life processes often have millions or even billions of possible states, which allows for terribly complicated strategies – and humans are notoriously bad at handling combinatorially complex objects.

To better model the way human agents strategize, the authors of [24, 25] proposed to use a more human-friendly representation of strategies, based on lists of condition-action rules. The conditions are given by Boolean formulas. Moreover, it was postulated that only those strategies should be considered whose complexity does not exceed a given bound. This is consistent with classical approaches to commonsense reasoning [16] and planning [20], as well as the empirical results on how humans learn and use concepts [8, 19].

Natural strategies. Let $\mathcal{B}(Prop_a)$ be the set of Boolean formulas over atomic propositions $Prop_a$ observable by agent a. In our case, $Prop_a$ consists of all the references to the local variables of agent a, as well as the global variables in the model. We represent natural strategies of agent a by *lists of guarded actions*, i.e., sequences of pairs $\phi_i \rightsquigarrow \alpha_i$ such that: (1) $\phi_i \in \mathcal{B}(Prop_a)$, and (2) α_i is an action available to agent a in every state where ϕ_i holds. Moreover, we assume that the last pair on the list is $\top \rightsquigarrow \alpha$ for some action α, i.e., the last rule is guarded by a condition that will always be satisfied. A *collective natural strategy* for a group of agents $A = \{a_1, \ldots, a_{|A|}\}$ is a tuple of individual natural strategies $s_A = (s_{a_1}, \ldots, s_{a_{|A|}})$. The set of such strategies is denoted by Σ_A.

The "outcome" function $out(q, s_A)$ returns the set of all paths (i.e., all maximal traces) that occur when coalition A executes strategy s_A from state q onward, and the agents outside A are free to act in an arbitrary way.

Complexity of strategies. We will use the following complexity metric for strategies: $compl(s_A) = \sum_{(\phi,\alpha)\in s_A} |\phi|$, with $|\phi|$ being the number of symbols in ϕ, without parentheses. That is, $compl(s_A)$ simply counts the total length of guards in s_A. Intuitively, the complexity of a strategy is understood as its level of sophistication. It corresponds to the mental effort needed to come up with the strategy, memorize it, and execute it.

2.3 Specification of Properties Based on Natural Strategies

To reason about natural strategic ability, the logic NatATL was introduced in [24, 25] with the following syntax:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle A\rangle\rangle^{\leq k}\mathsf{X}\,\varphi \mid \langle\langle A\rangle\rangle^{\leq k}\mathsf{F}\,\varphi \mid \langle\langle A\rangle\rangle^{\leq k}\mathsf{G}\,\varphi \mid \langle\langle A\rangle\rangle^{\leq k}\varphi\,\mathsf{U}\,\varphi.$$

where A is a group of agents and $k \in \mathbb{N}$ is a complexity bound. Intuitively, $\langle\langle A\rangle\rangle^{\leq k}\gamma$ reads as "coalition A has a collective strategy of size less or equal than k to enforce the property γ." The formulas of NatATL make use of classical temporal operators: "X" ("in the next state"), "G" ("always from now on"), "F" ("now or sometime in the future"), and U (strong "until"). For example, the formula $\langle\langle cust\rangle\rangle^{\leq 10}\mathsf{F}\ \mathsf{ticket}$ expresses that the customer can obtain a ticket by a strategy of complexity at most 10. This seems more appropriate as a functionality requirement than to require the existence of *any* function from states to actions. The path quantifier "for all paths" can be defined as $\mathsf{A}\gamma \equiv \langle\langle\emptyset\rangle\rangle^{\leq 0}\gamma$.

We will use NatATL to specify requirements on the voting system. For example, voter-verifiability captures the ability of the voter to verify her vote after the election. In our case, this is represented by the *check*4 phase, hence we can specify voter-verifiability with the formula $\langle\langle voter\rangle\rangle^{\leq k}\mathsf{F}\ (\mathsf{check4\_ok} \vee \mathsf{error})$. The intuition is simple: the voter has a strategy of size at most k to successfully perform *check*4 or else signal an error. A careful reader can spot one problem with the formalization: it holds if the voter can signal an error regardless of the outcome of the check (and it shouldn't!). A better specification is given by $\langle\langle voter\rangle\rangle^{\leq k}\mathsf{F}\ (\mathsf{check4\_ok} \vee \mathsf{check4\_fail})$. Moreover, we can use the formula $\mathsf{AG}\ (\mathsf{check4\_fail} \rightarrow \langle\langle voter\rangle\rangle^{\leq k}\mathsf{F}\ \mathsf{error})$ to capture *dispute resolution*.

The conceptual structure of receipt-freeness is different. In that case, we want to say that the voter has no way of proving how she has voted, and that the coercer (or a potential vote-buyer) does not have a strategy that allows him to learn the value of the vote. Crucially, this refers to the *knowledge* of the coercer. To capture the requirement, we need to extend NatATL with knowledge operators K_a, with $K_a\varphi$ expressing that agent a knows that φ holds. For instance, $K_{coerc}\mathsf{voted_i}$ says that the coercer knows that the voter has voted for the candidate i. Then, receipt-freeness can be formalized as

$$\bigwedge_{i\in Cand} \neg\langle\langle coerc, voter\rangle\rangle^{\leq k}\mathsf{G}\ (\mathsf{end} \rightarrow (K_{coerc}\mathsf{voted_i} \vee K_{coerc}\neg\mathsf{voted_i})).$$

That means that the coercer and the voter have no strategy with complexity at

most k to learn, after the election is finished, whether the voter has voted for i or not. Note that this is only one of the possible formalization of the requirement. For example, one may argue that, to violate receipt-freeness, it suffices that the coercer can detect *whenever the voter has not obeyed*; he does not have to learn the exact value of her vote. This can be captured by the following formula: $\bigwedge_{i \in Cand} \neg\langle\!\langle coerc, voter \rangle\!\rangle^{\leq k} \mathsf{G}\,((\mathsf{end} \wedge \neg \mathsf{voted_i}) \rightarrow K_{coerc} \neg \mathsf{voted_i})$.

2.4 Using Verification Tools to Facilitate Analysis

The focus of this work is on modeling and specification; the formal analysis is done mainly by hand. However, having the models specified in UPPAAL suggests that we can also benefit from its model checking functionality. Unfortunately, the requirement specification language of UPPAAL is very limited, and allows for neither strategic operators nor knowledge modalities. Still, we can use it to verify concrete strategies if we carefully modify the input formula and the model. We will show how to do it in Section 6.

3 Use Case Scenario: vVote

Secure and verifiable voting is becoming more and more important for the democracy to function correctly. In this paper, we analyze the vVote implementation of Prêt à Voter which was used for remote voting and voting of handicapped persons in the Victorian elections in November 2014 [13]. The main idea of the Prêt à Voter protocol focuses on encoding the vote using a randomized candidate list. In this protocol the ballot consists of two parts: the randomized order of candidates (left part) and the list of empty checkboxes along with the number encoding the order of the candidates (right part). The voter cast her vote in an usual way, by placing a cross in the right hand column against the candidate of her choice. After that she tears the ballot in two parts, destroys the left part, cast the right one and takes the copy of it as her receipt. After the election her vote appears on the Web Bulletin Board as the pair of the encoding number and the marked box, which can be compared with the receipt for verification. We look at the whole process, from the voter entering the polling station, to the verification of her vote on the public Web Bulletin Board (WBB).

After entering the polling station, the Poll Worker (PW) authenticates the voter (using the method prescribed by the appropriate regulations), and sends a print request to the Print On Demand (PON) device specifying the district/region of the voter. If the authentication is valid (state *printing*) then the PON retrieves and prints an appropriate ballot for the voter, including a Serial Number (SN) and the district, with a signature from the Private Web Bulletin Board (PWBB). The PWBB is a robust secure database which receives messages, performs basic validity checks, and returns signatures. After that, the voter may choose to check and confirm the ballot. This involves demanding a proof that the ballot is properly formed, i.e., that the permuted candidate list corresponds correctly to the cipher-texts on the public WBB for that serial number. The

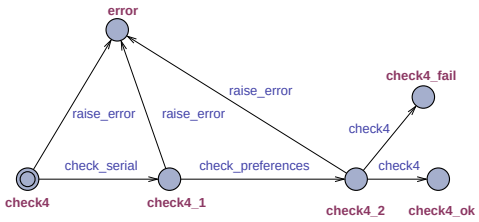

Fig. 2. Voter refinement: phase check4

WBB is an authenticated public broadcast channel with memory. If the ballot has a confirmation check, the voter returns to the printing step for a new ballot (transition from state *check*1 to *printing*).

Having obtained and possibly checked her ballot (state *has_ballot*), the voter can scan it by showing the ballot bar code to the Electronic Ballot Marker (EBM). Then, she enters her vote (state *scanning*) via the EBM interface. The EBM prints on a separate sheet the voters receipt with the following information: (i) the electoral district, (ii) the Serial Number, (iii) the voter's vote permuted appropriately to match the Prêt à Voter ballot, and (iv) a QR code with this data and the PWBB signature.

Further, the voter must check the printed vote against the printed candidate list. In particular, she checks that the district is correct and the Serial Number matches the one on the ballot form. If all is well done, she can optionally check the PWBB signature, which covers only the data visible to the voter. Note that, if either *check*2 or *check*3 fails, the vote is canceled using the cancellation protocol. If everything is OK, the voter validates the vote, shreds the candidate list, and leaves the polling station. Finally, the voter can check her vote on the WBB after the election closes. She only needs to check the SR and the order of her preference numbers.

4 Models

In this section we present models of a simplified version of vVote, focusing on the steps that are important from the voter's perspective. We use UPPAAL as the modeling tool because of its flexible modeling language and user-friendly GUI.

4.1 Voter Model

The model already presented in Figure 1 captures the voter actions from entering the polling station to casting her vote, going back home and verifying her receipt on the web bulletin board. As shown in the model, some actions, i.e. additional checks, are optional for the voter. Furthermore, to simulate the human behavior we added some additional actions, not described in the protocol itself. For example the voter can skip even obligatory steps, like *check*2. This

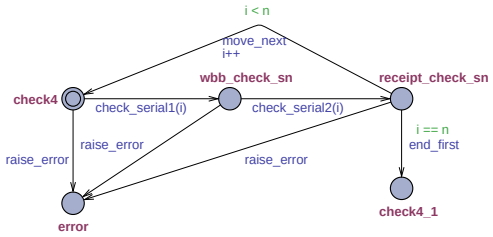

Fig. 3. Serial number check

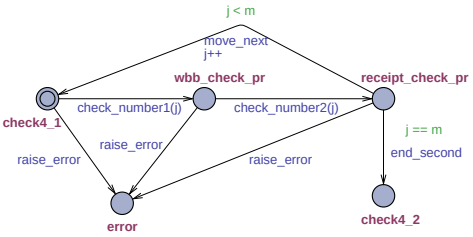

Fig. 4. Preferences order check

is especially important, as *check*2 may be the most time-consuming action for the voter and many voters may skip it in the real life. After every check, the voter can signal an error, thus ending up in the *error* state. The state represents communication with the election authority, signaling that the voter could not cast her vote or a machine malfunction was detected.

4.2 Refinements of the Voter Model

The model shown in Figure 1 is relatively abstract. For example, *check*4 is shown as an atomic action, but in fact it requires that the voter compares data from the receipt and the WBB. In order to properly measure the complexity of the voter strategies, it is crucial to consider different levels of granularity.

Check4 phase. Recall that this is the last phase in the protocol and it is optional. Here, the voter can check that the printed receipt matches her intended vote on the WBB. This includes checking that the serial numbers match (action *check_serial*), and that the printed preferences order match the one displayed on the WBB (action $check\_preferences$). So, if both steps succeed, then the voter reach state *check*4_*ok*. The whole model for this phase is presented in Figure 2. Other phases, like *check*2 can be presented in a similar way.

Serial number phase. In some cases the model shown in Figure 2 may still be too general. Depending on the length of the serial number we can have different levels of difficulty. For example comparing two alphanumeric sequences of length 2 is easier then comparing such sequences of length 10. To express this concept, we split this step into atomic actions: $check\_serial1(i)$ for checking the ith symbol on the WBB, and $check\_serial2(i)$ for checking the ith symbol on the receipt. The resulting model is shown in Figure 3, where n is the length of the serial number.

Preferences order phase. Similarly to comparing the two serial numbers, verifying the printed preferences can also be troublesome for the voter. In order to make sure that her receipt matches the entry on the WBB, the voter must check each number showing her preference. Actions $check\_number1(i)$ and $check\_number2(i)$ mean checking number on the WBB and on the receipt, respectively. This is shown on the model in Figure 4, where m is the number of candidates in the ballot.

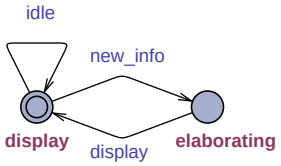

Fig. 5. Public WBB

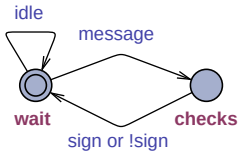

Fig. 6. Private WBB

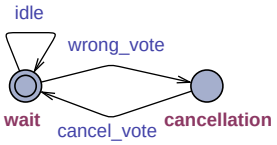

Fig. 7. Cancel station

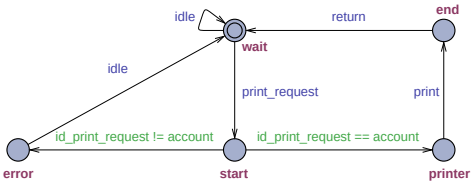

Fig. 8. Print-on-demand printer

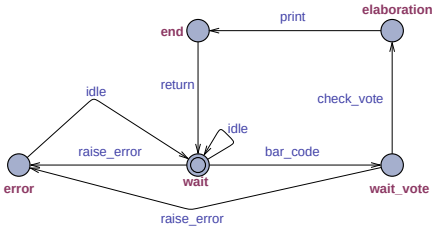

Fig. 9. Electronic Ballot Marker (EBM)

4.3 Voting Infrastructure

The voter is not the only entity taking part in the election procedure. The election infrastructure and the electronic devices associated with it constitute a significant part of the procedure. Since there are several components involved in the voting process, we decided to model each component as a separate agent. The models of the Public WBB, Private WBB, the cancel station, the print-on-demand printer, and the EBM are shown in Figures 5–9.

4.4 Coercer Model

To model the coercer, we first need to determine his exact capabilities. Is he able to interact with the voter, or only with the system? Should he have full control over the network, like the Dolev-Yao attacker, or do we want the agent to represent implicit coercion, where the relatives or subordinates are forced to vote for the specified candidate. There are many possibilities, and to this end we have decided that one model for the Coercer agent is not enough. Because of that and due to lack of space, we omit the details, and only describe the possible actions of the coercer:

- $Coerce(v, ca)$: the coercer coerces the voter v to vote for his candidate ca;
- $ModifyBallot(v, ca)$: the coercer modifies the ballot for v by setting ca;
- $RequestVote(v)$: the coercer requests a vote from v;
- $Punish(v)$: the coercer punishes v;
- $Infect$: the coercer infects the voting machine with malicious code;
- $Listen(v)$: the coercer listens to the vote of v from the voting machine;
- $Replace(v, ca)$: the coercer replaces the vote of v with ca.

Some of the actions may depend on each other. For example, *Listen* and *Replace* actions should be executed only after the $Infect$ action has succeed, as the Coercer needs some kind of access to the voting machine.

5 Strategies and Their Complexity

There are many possible objectives for the participants of a voting procedure. A voter's goal could be to just cast her vote, another one could be to make sure that her vote was correctly counted, and yet another one to verify the election results. The same goes for the coercer: he may just want to make his family vote in the "correct," or to change the outcome of the election. In order to define different objectives, we can use formulas of NatATL and look for appropriate natural strategies, as described in Section 2. More precisely, we can fix a subset of the participants and their objective with a formula of NatATL, find the smallest strategy that achieves the objective, and compute its size. The size of the strategy will be an indication of how hard it is to make sure that the objective is achieved.

An example of specification that the voter wants to achieve is the verifiability of her vote. Given the model in Figure 1, we can use the formula $\langle\langle v\rangle\rangle^{\leq k}\mathrm{F}check4\_ok$ to check whether the voter has a natural strategy of size less or equal than k to verify that her receipt has the same information displayed in the public WBB. If we want to check only if the voter has a natural strategy to verify her receipt, i.e. we want also consider the case in which the voter's receipt and the information in the public WBB are different, we can consider the formula $\langle\langle v\rangle\rangle^{\leq k}\mathrm{F}(check4\_ok \vee check4\_fail)$ as we discussed in Section 2.3.

Note that it is essential to fix the granularity level of the modeling right. When shifting the level of abstraction, we obtain significantly different "measurements" of strategic complexity. This is why we proposed several variants of the voter model in Section 4. In this section, we will show how it affects the outcome of the analysis.

In the following we take another look at the previously defined models and try to list possible strategies for the participants.

5.1 Strategies for the Voter

In this section we focus on natural strategies for voter-verifiability. Given the model in Figure 1, we can analyze an example of natural strategy that can be used by the voter to achieve the end of the voting procedure, i.e., to satisfy the NatATL formula $\varphi_1 = \langle\langle v\rangle\rangle^{\leq k}\mathrm{F}\mathsf{end}$. Clearly, φ_1 specifies the fact that the voter has a natural strategy of size less or equal than k (captured by $\langle\langle v\rangle\rangle^{\leq k}$) to reach sooner or later (captured by the eventually operator F) the end of procedure (i.e. the state labeled with atom end).

Natural Strategy 1 *A strategy for the voter is:*

1. has_ballot $\rightsquigarrow$ *scan_ballot*
2. scanning $\rightsquigarrow$ *enter_vote*

3. voted $\rightsquigarrow$ *check*2
4. check2_ok $\vee$ check2_fail $\vee$ out $\rightsquigarrow$ *move_next*
5. vote_ok $\rightsquigarrow$ *shred_ballot*
6. shred $\rightsquigarrow$ *leave*
7. check4 $\rightsquigarrow$ *check*4
8. check4_ok $\vee$ check4_fail $\rightsquigarrow$ *finish*
9. $\top \rightsquigarrow \star$

Recall that the above is an ordered sequence of guarded commands. The first condition (guard) that evaluates to *true* determines the action of the voter. Thus, if the voter has the ballot and she has not scanned it (proposition has_ballot), she scans the ballot. If has_ballot is false and scanning is true then she enters her vote, and so on. If all the preconditions except $\top$ are false, then she executes an arbitrary available action (represented by the wildcard $\star$). For example, the voter will do *print_ballot* at the state *printing*, where the voter needs to wait while the Pool Walker identifies her and generates a new ballot.

In Natural Strategy 1, we have 9 guarded commands in which the command (4) costs 5 since in its condition there are five symbols (three atoms plus two disjunctions), the command (7) costs 3 since in its condition there are three symbols (two atoms plus the disjunction), while the other guarded commands cost 1, so the total complexity is $1 \cdot 7 + 3 \cdot 1 + 5 \cdot 1 = 15$. So, the formula φ_1 is true with any k of 15 or more. Further, by Natural Strategy 1, starting from the state has_ballot, the voter needs of 9 steps to achieve the state end.

Note that Natural Strategy 1 can be also used to demonstrate that the formula $\psi = \langle\langle v \rangle\rangle^{\leq k}\mathrm{F}(\mathsf{check4\_ok} \vee \mathsf{check4\_fail})$ holds. In that case, we can reduce the size of the strategy by removing the guarded command (8). Thus, ψ is satisfied even for $k \geq 12$.

For completeness, in what follows, we show a natural strategy with the additional guarded commands in case the voter wants to do the optional phases *check*1 and *check*3, i.e., we want to satisfy the formula $\varphi_2 = \langle\langle v \rangle\rangle^{\leq k}\mathrm{F}(\mathsf{checked1} \wedge \mathsf{checked3} \wedge \mathsf{end})$. In particular, φ_2 checks whether there exist a natural strategy for the voter such that sooner or later she does *check*1, *check*3, and ends the whole voting process. Note that, apart from the standard propositions like *check*1, we also add their persistent version like *checked*1, i.e., once it gets true, it remains always true.

Natural Strategy 2 *A strategy for the voter that considers the optional phases* *check*1 *and* *check*3 *is:*

1. has_ballot $\wedge$ counter $==$ 0 $\rightsquigarrow$ *check*1
2. has_ballot $\rightsquigarrow$ *scan_ballot*
3. scanning $\rightsquigarrow$ *enter_vote*
4. voted $\rightsquigarrow$ *check*2
5. check2_ok $\vee$ check2_fail $\rightsquigarrow$ *check*3
6. check1 $\vee$ check3 $\vee$ out $\rightsquigarrow$ *move_next*
7. vote_ok $\rightsquigarrow$ *shred_ballot*
8. shred $\rightsquigarrow$ *leave*

9. check4 $\rightsquigarrow$ *check*4
10. check4_ok $\vee$ check4_fail $\rightsquigarrow$ *finish*
11. $\top \rightsquigarrow \star$

In Natural Strategy 2, we introduce the verification of *check*1 and *check*3. To do this we add two new guarded commands (5) and (6), and we update (1) by adding a control on a counter to determine if *check*1 is done or not. Here, we have a total complexity of $1 \cdot 7 + 3 \cdot 3 + 5 \cdot 1 = 21$. So, the formula φ_2 is true for any $k \geq 21$. Further, by Natural Strategy 2, starting from the state has_ballot, the voter needs of 13 steps to achieve the state end.

An important aspect to evaluate in this subject concerns the detailed analysis of *check*4. Some interesting questions on this analysis could be: how does the voter perform *check*4? How does she compare the printed preferences with the information in the public WWB? These questions open up several scenarios both from a strategic point of view and from the model to be used. From a strategic point of view, we could consider a refinement of Natural Strategy 1, in which the action *check*4 is evaluated as something of atomic. If we consider that the *check*4 includes: comparing preferences with the information in the public WWB and checking the serial number, we can already divide the single action into two different actions for each of the checks to be performed. So, given the model in Figure 2, we can refine the natural strategy for the voter, as follows.

Natural Strategy 3 *A strategy for the voter that refines check*4 *is:*

1. has_ballot $\rightsquigarrow$ *scan_ballot*
2. scanning $\rightsquigarrow$ *enter_vote*
3. voted $\rightsquigarrow$ *check*2
4. check2_ok $\vee$ check2_fail $\vee$ out $\rightsquigarrow$ *move_next*
5. vote_ok $\rightsquigarrow$ *shred_ballot*
6. shred $\rightsquigarrow$ *leave*
7. check4 $\rightsquigarrow$ *check_serial*
8. check4_1 $\rightsquigarrow$ *check_preferences*
9. check4_2 $\rightsquigarrow$ *check*4
10. check4_ok $\vee$ check4_fail $\rightsquigarrow$ *finish*
11. $\top \rightsquigarrow \star$

In Natural Strategy 3, we have 11 guarded commands in which all the conditions are defined with a single atom but (4) in which there is a disjunction of three atoms and (10) in which there is a disjunction of two atoms. So, the total complexity is $1 \cdot 9 + 3 \cdot 1 + 5 \cdot 1 = 17$.

To verify that the voter does each step of *check*4, we need to provide a formula that verifies atoms check4, check4_1, and check4_2. To do this in NatATL, we use the formula $\varphi_3 = \langle\langle v \rangle\rangle^{\leq k} \mathrm{F}(\mathsf{checked4} \wedge \mathsf{checked4\_1} \wedge \mathsf{checked4\_2})$. Note that φ_3 is true for any $k \geq 17$; one can use Natural Strategy 3 to demonstrate that.

In addition, to increase the level of detail, one could consider that the voter checks the preferences and the serial number one by one in an ordered fashion. So, given the models in Figures 3 and 4, we can consider a formula that checks whether the voter has a strategy that satisfies the following properties:

1. sooner or later she enters in the *check*4 phase;
2. she verifies a symbols of the SN in the public WBB and in her receipt;
3. she does (2) until the last symbol of the serial number is verified;
4. she does a similar approach as in (2)-(3) for the verification of preferences;
5. she finishes the whole procedure.

This can be captured by the formula $\varphi_4 = \langle\langle v \rangle\rangle^{\leq k} \mathrm{F}(\mathsf{checked4} \wedge \mathsf{wbb\_checked\_sn} \wedge \mathsf{receipt\_checked\_sn} \wedge \mathsf{checked4\_1} \wedge \mathsf{wbb\_checked\_pr} \wedge \mathsf{receipt\_checked\_pr} \wedge \mathsf{checked4\_2})$. So, we can define a natural strategy that satisfies φ_4, as follows.

Natural Strategy 4 *A strategy for the voter that still refines check*4 *is:*

1. $\mathsf{has\_ballot} \rightsquigarrow \mathit{scan\_ballot}$
2. $\mathsf{scanning} \rightsquigarrow \mathit{enter\_vote}$
3. $\mathsf{voted} \rightsquigarrow \mathit{check}2$
4. $\mathsf{vote\_ok} \rightsquigarrow \mathit{shred\_ballot}$
5. $\mathsf{shred} \rightsquigarrow \mathit{leave}$
6. $\mathsf{out} \vee \mathsf{check2\_ok} \vee \mathsf{check2\_fail} \vee \mathsf{receipt\_check\_sn} \vee \mathsf{receipt\_check\_pr} \rightsquigarrow \mathit{move\_next}$
7. $\mathsf{check4} \rightsquigarrow \mathit{check\_serial}1$
8. $\mathsf{wbb\_check\_sn} \rightsquigarrow \mathit{check\_serial}2$
9. $\mathsf{receipt\_check\_sn} \wedge \mathsf{i} == \mathsf{n} \rightsquigarrow \mathit{end\_first}$
10. $\mathsf{check4\_1} \rightsquigarrow \mathit{check\_number}1$
11. $\mathsf{wbb\_check\_pr} \rightsquigarrow \mathit{check\_number}2$
12. $\mathsf{receipt\_check\_pr} \wedge \mathsf{j} == \mathsf{m} \rightsquigarrow \mathit{end\_second}$
13. $\mathsf{check4\_2} \rightsquigarrow \mathit{check}4$
14. $\mathsf{check4\_ok} \vee \mathsf{check4\_fail} \rightsquigarrow \mathit{finish}$
15. $\top \rightsquigarrow \star$

To conclude, the above natural strategy has 15 guarded commands in which the conditions in (9), (12), and (14) are conjunctions of two atoms, the condition in (6) is a disjunction of five atoms, and all the other conditions are defined with a single atom. Therefore, the complexity of Natural Strategy 4 is $1{\cdot}11{+}3{\cdot}3{+}9{\cdot}1 = 29$. So, the formula φ_4 is true with $k \geq 29$.

5.2 Counting Other Kinds of Resources

So far, we have measured the effort of the voter by how complex strategies she must execute. This helps to estimate the mental difficulty related, e.g., to voter-verifiability. However, this is not the only source of effort that the voter has to invest. Verifying one's vote might require money (for example, if the voter needs to buy special software or a dedicated device), computational power, and, most of all, time. Here, we briefly concentrate on the latter factor.

For a voter's task expressed by the NatATL formula $\langle\langle v \rangle\rangle^{\leq k} \mathsf{F}\, \varphi$ and a natural strategy s_v for the voter, we can estimate the time spent on the task by the number of transitions necessary to reach φ. That is, we take all the paths in $\mathit{out}(q, s_v)$, where q is the initial state of the procedure. On each path, φ must occur at some point. We look for the path where the first occurrence of φ happens

latest, and count the number of steps to φ on that path. We will demonstrate how it works on the last two strategies from Section 5.1.

For Natural Strategy 3, starting from the starting state, the voter needs of $9+2=11$ steps to achieve $\mathsf{check4} \wedge \mathsf{wbb\_check\_sn} \wedge \mathsf{receipt\_check\_sn} \wedge \mathsf{check4\_1} \wedge \mathsf{wbb\_check\_pr} \wedge \mathsf{receipt\_check\_pr} \wedge \mathsf{check4\_2}$. More precisely, 9 steps are needed to achieve *check*4 in the local model shown in Figure 1, and 2 more steps to reach *check*4_2 in the refinement of the final section of the procedure (see Figure 2).

For Natural Strategy 4, the calculation is slightly more complicated. Starting from state start, the voter needs of $9+((2\cdot n)+1)+((2\cdot m)+1)$ steps to achieve her goals, where n and m are the sizes of the serial number and the list of preferences, respectively. In particular, 9 steps are needed to achieve *check*4 in Figure 1, then $((2\cdot n)+1)$ steps to achieve *check*4_1 in Figure 3, and finally $((2\cdot m)+1)$ steps to achieve *check*4_2 in Figure 4.

5.3 Strategies for the Coercer

For the coercer, we can start the analysis by considering the basic setting in which he requests the vote and if the voter does not give him the ballot or the vote is different with the one imposed by him, then he punish the voter. This reasoning is captured by the following natural strategy.

1. $\neg\mathsf{coerced_v} \rightsquigarrow Coerce(v, ca)$
2. $\mathsf{coerced_v} \wedge \neg\mathsf{requested_v} \rightsquigarrow RequestVote(v)$
3. $\mathsf{coerced_v} \wedge \mathsf{requested_v} \wedge \neg\mathsf{punished_v} \wedge (\mathsf{ca_v} \neq \mathsf{ca} \vee \mathsf{not\_show_v}) \rightsquigarrow Punish(v)$

The total complexity of the above natural strategy is 16 since (1) has 2 symbols (atom + negation), (2) has 4 symbols (two atoms + conjunction + negation), and (3) has 10 symbols (five atoms + three conjunctions + negation + disjunction).

Another intervention of the coercer regards the possibility to infect one or more machines and to replace the vote of a voter.

1. $\neg\mathsf{infected} \rightsquigarrow Infect$
2. $\mathsf{infected} \wedge \neg\mathsf{replaced_v} \rightsquigarrow Replace(v, ca)$

In this case the complexity is 6 since we have three atoms involved, two negations, and a conjunction.

We can extend the above by considering that the coercer can infect the machine and then can coerce the voter. So, if the coercer see a different vote on the machine he can punish the voter. Note that this setting is interesting in the case the coercer has infected a machine that only displays informations without having the power to modify the vote.

1. $\neg\mathsf{infected} \rightsquigarrow Infect$
2. $\neg\mathsf{coerced_v} \rightsquigarrow Coerce(v, ca)$
3. $\mathsf{infected} \wedge \mathsf{listen_v} \neq \mathsf{ca} \rightsquigarrow Punish(v)$

In the above, the complexity is 7, where (1) and (2) have complexity 2 and (3) has complexity 3.

6 Automated Verification of Strategies

In this section we explain how the model checking functionality of UPPAAL can be used for an automated verification of the strategies presented in Section 5. To verify selected formulas and the corresponding natural strategies, we need to modify several things: *(i)* the formula, *(ii)* the natural strategy and finally, *(iii)* the model. We start by explaining the required modifications step by step.

Formula. To specify the required properties for the protocol, we have used a variant of strategic logic, as it is one of the most suited logic to specify properties for agents in an intuitive way. However, UPPAAL does not support NatATL, so the formula needs to be modified accordingly. In the formula we replace the strategic operator $\langle\langle A \rangle\rangle^{\leq k}$ with an universal path quantifier A. For example, we can consider the formula φ_1 used in Section 5. In this context, we produce the CTL formula $\varphi_1' = \mathrm{AF}\mathsf{end}$.

Natural Strategy. The natural strategy needs to be modified so that all the guard conditions are mutually exclusive. To this end, we go through the preconditions from top to bottom, and refine them by adding the negated preconditions from all the previous guarded commands. For example, Natural Strategy 1 is modified as follows:

1. $\mathsf{has\_ballot} \rightsquigarrow \mathit{scan\_ballot}$
2. $\neg\mathsf{has\_ballot} \wedge \mathsf{scanning} \rightsquigarrow \mathit{enter\_vote}$
3. $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \mathsf{voted} \rightsquigarrow \mathit{check}2$
4. $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge (\mathsf{check2\_ok} \vee \mathsf{check2\_fail} \vee \mathsf{out}) \rightsquigarrow \mathit{move\_next}$
5. $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail} \vee \mathsf{out}) \wedge \mathsf{vote\_ok} \rightsquigarrow \mathit{shred\_ballot}$
6. $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail} \vee \mathsf{out}) \wedge \neg\mathsf{vote\_ok} \wedge \mathsf{shred} \rightsquigarrow \mathit{leave}$
7. $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail} \vee \mathsf{out}) \wedge \neg\mathsf{vote\_ok} \wedge \neg\mathsf{shred} \wedge (\mathsf{check4\_ok} \vee \mathsf{check4\_fail}) \rightsquigarrow \mathit{finish}$
8. $\top \rightsquigarrow \star$

Model. To verify the selected strategy, we fix it in the model by adding the preconditions of the guards to the preconditions of the corresponding local transitions in the voter's model. Thus, we effectively remove all transitions that are not in accordance with the strategy. This way, only the paths that are consistent with the strategy will be considered by the model-checker.

Levels of granularity. As we showed in Section 4, it is often important to have variants of the model for different levels of abstraction. To handle those in UPPAAL, we have used synchronizations edges. For example, to have a more detailed version of the phase *check*4, we added synchronization edges in the voter model (Figure 1) and in the *check*4 model (Figure 2). Then, when going through the *check*4 phase in the voter model, UPPAAL will proceed to the more detailed model and come back after getting to its final state.

Running the verification. We have modified the models, formulas, and strategies from Sections 4 and 5 following the above steps. Then, we used UPPAAL to

verify that Natural Strategies 1–4 indeed enforce the prescribed properties. The tool reported that each formula holds in the corresponding model. The execution time was always at most a few seconds.

7 Conclusions

In the analysis of a voting protocols it is important to make sure that the voter has a strategy to use the functionality of the protocol. That is, she has a strategy to fill in and cast her ballot, verify her vote on the bulletin board, etc. However, this is not enough: it is also essential to see how hard that strategy is. In this paper, we propose a methodology that can be used to this end. One can assume a natural representation of the voter's strategy, and to measure its complexity as the size of the representation.

We mainly focus on one aspect of the voter's effort, namely the mental effort needed to produce, memorize, and execute the required actions. We also indicate that there are other important factors, like the time needed to execute the strategy or the financial cost of the strategy. This may lead to tradeoffs where optimizing the costs with respect to one resource leads to higher costs in terms of another resource. Moreover, resources can vary in their importance for different agents. For example, time may be more important for the voter, while money more relevant when we analyze the strategy of the coercer. We leave a closer study of such tradeoffs for future work.

Another interesting extension would be to further analyze the parts of the protocol where the voter compares two numbers, tables, etc. As the voter is a human being, it is natural for her to make a mistake. Furthermore, the probability of making a mistake at each step can be added to the model to analyze the overall probability of successfully comparing two data sets by the voter.

Finally, we point out that the methodology proposed in this paper can be applied outside the e-voting domain. For example, one can use it to study the usability of policies for social distancing in the current epidemic situation, and whether they are likely to obtain the expected results.

References

1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.
2. David A. Basin, Hans Gersbach, Akaki Mamageishvili, Lara Schmid, and Oriol Tejada. Election security and economics: It's all about Eve. In *Proceedings of E-Vote-ID*, pages 1–20, 2017.
3. David A. Basin, Sasa Radomirovic, and Lara Schmid. Modeling human errors in security protocols. In *Computer Security Foundations Symposium, CSF*, pages 325–340. IEEE Computer Society, 2016.
4. G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems: SFM-RT*, number 3185 in LNCS, pages 200–236. Springer, 2004.

5. Giampaolo Bella, Paul Curzon, Rosario Giustolisi, and Gabriele Lenzini. A socio-technical methodology for the security and privacy analysis of services. In *COMPSAC Workshops*, pages 401–406. IEEE Computer Society, 2014.
6. Giampaolo Bella, Paul Curzon, and Gabriele Lenzini. Service security and privacy as a socio-technical problem. *J. Comput. Secur.*, 23(5):563–585, 2015.
7. J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of Computing*, pages 544–553. ACM, 1994.
8. L. E. Bourne. Knowing and using concepts. *Psychol. Rev.*, 77:546–556, 1970.
9. Ahto Buldas and Triinu Mägi. Practical security analysis of e-voting systems. In *Proceedings of IWSEC*, volume 4752 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2007.
10. Marcelo Carlomagno Carlos, Jean Everson Martina, Geraint Price, and Ricardo Felipe Custódio. A proposed framework for analysing security ceremonies. In *SECRYPT*, pages 440–445. SciTePress, 2012.
11. K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. *Information and Computation*, 208(6):677–693, 2010.
12. V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung. SoK: Verifiability notions for e-voting protocols. In *IEEE Symposium on Security and Privacy*, pages 779–798, 2016.
13. C. Culnane, P.Y.A. Ryan, S.A. Schneider, and V. Teague. vvote: A verifiable voting system. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.
14. Chris Culnane and Vanessa Teague. Strategies for voter-initiated election audits. In *Decision and Game Theory for Security: Proceedings of GameSec*, volume 9996 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2016.
15. Nicolas David, Alexandre David, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr. Olesen, and Christian W. Probst. Modelling social-technical attacks with timed automata. In *Proceedings of International Workshop on Managing Insider Security Threats, MIST*, pages 21–28. ACM, 2015.
16. E. Davis and G. Marcus. Commonsense reasoning. *Communications of the ACM*, 58(9):92–103, 2015.
17. S. Delaune, S. Kremer, and M. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Computer Security Foundations Workshop, 2006. 19th IEEE*, pages 12–pp. IEEE, 2006.
18. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
19. J. Feldman. Minimization of Boolean complexity in human concept learning. *Nature*, 407:630–3, 11 2000.
20. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.
21. Jeffrey Hunker and Christian W. Probst. Insiders and insider threats - an overview of definitions and mitigation techniques. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 2(1):4–27, 2011.
22. W. Jamroga, M. Knapik, and D. Kurpiewski. Model checking the SELENE e-voting protocol in multi-agent logics. In *Proceedings of the 3rd International Joint Conference on Electronic Voting (E-VOTE-ID)*, volume 11143 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2018.
23. W. Jamroga and M. Tabatabaei. Preventing coercion in e-voting: Be open and commit. In *Electronic Voting: Proceedings of E-Vote-ID 2016*, volume 10141 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2017.

24. Wojciech Jamroga, Vadim Malvone, and Aniello Murano. Natural strategic ability. *Artificial Intelligence*, 277, 2019.
25. Wojciech Jamroga, Vadim Malvone, and Aniello Murano. Natural strategic ability under imperfect information. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2019*, pages 962–970. IFAAMAS, 2019.
26. A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 61–70. ACM, 2005.
27. R. Küsters, T. Truderung, and A. Vogt. A game-based definition of coercion-resistance and its applications. In *Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium*, pages 122–136. IEEE Computer Society, 2010.
28. T. Martimiano, E. Dos Santos, M. Olembo, and J.E. Martina. Ceremony analysis meets verifiable voting: Individual verifiability in Helios. In *SECURWARE*, 2015.
29. Taciane Martimiano and Jean Everson Martina. Threat modelling service security as a security ceremony. In *11th International Conference on Availability, Reliability and Security, ARES*, pages 195–204. IEEE Computer Society, 2016.
30. F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Transactions on Computational Logic*, 15(4):1–42, 2014.
31. Peter Y. A. Ryan, Steve A. Schneider, and Vanessa Teague. End-to-end verifiability in voting systems, from theory to practice. *IEEE Security & Privacy*, 13(3):59–62, 2015.
32. P.Y.A. Ryan. The computer ate my vote. In *Formal Methods: State of the Art and New Directions*, pages 147–184. Springer, 2010.
33. F.P. Santos. *Dynamics of Reputation and the Self-organization of Cooperation*. PhD thesis, University of Lisbon, 2018.
34. F.P. Santos, F.C. Santos, and J.M. Pacheco. Social norm complexity and past reputations in the evolution of cooperation. *Nature*, 555:242–245, 2018.
35. Y. Shoham and K. Leyton-Brown. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.
36. M. Tabatabaei, W. Jamroga, and Peter Y. A. Ryan. Expressing receipt-freeness and coercion-resistance in logics of strategic ability: Preliminary attempt. In *Proceedings of the 1st International Workshop on AI for Privacy and Security, PrAISe@ECAI 2016*, pages 1:1–1:8. ACM, 2016.
37. Verified Voting. Policy on direct recording electronic voting machines and ballot marking devices. 2019.