Journal of Computer Security 1 (2021) 1–6
IOS Press

How to Measure Usable Security: Natural Strategies in Voting Protocols

Wojciech Jamroga [a,b], Damian Kurpiewski [b] and Vadim Malvone [c,*]
[a] *Interdisc. Centre on Security, Reliability and Trust, SnT, University of Luxembourg*
E-mail: wojciech.jamroga@uni.lu
[b] *Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland*
E-mail: d.kurpiewski@ipipan.waw.pl
[c] *Télécom Paris,France*
E-mail: vadim.malvone@telecom-paris.fr

Abstract. Formal analysis of security is often focused on the technological side of the system. One implicitly assumes that the users will behave in the right way to preserve the relevant security properties. 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 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.

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 voter-verifiability, and measure the voter's effort that they require. We also consider the dual view of graded security, measured by the complexity of the attacker's strategy to compromise the relevant properties of the election.

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. This concerns the mental difficulty of handling the right behavior, as well as the costs in terms of time, money, computing power etc. necessary to obtain it.

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) [9, 21, 23, 38, 40, 48]. More recently, significant progress has

[*]Corresponding author. E-mail: vadim.malvone@telecom-paris.fr.

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 [16, 50]. 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 [51] and its implementation vVote [17] 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 [57], which in turn depends on how understandable and easy to use the mechanism is [39, 43]. Preliminary evidence suggests that, often, the number of voters who check if their votes have been cast as intended and recorded as cast is quite small [10, 13, 27]. The same probably applies to mechanisms that support 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 often avoid it.

Thus, the right question is often not only *if* but also *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. A similar observation applies to systems that are essentially vulnerable, i.e., no matter what the user does the attacker has a strategy to compromise the security property φ. In that case, we can talk about the *practical degree of vulnerability* by considering how complex a successful attack strategy must be and what resources it requires. In that view, the simpler and cheaper to compromise φ, the higher the degree of vulnerability.

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 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 [24]. We consider three main types of agents participating in the voting process: the election system, a voter, and a potential coercer. Additionally, we consider an intruder that can infect the voting machine with malware to eavesdrop and even change the votes being cast on the machine. Then, we identify strategies for the voter to use the voter-verifiability mechanism, and estimate the voter's effort that they require. The strategic reasoning and its complexity is formalized by means of so called *natural strategies*, proposed in [36, 37] and consistent with psychological evidence on how humans use symbolic concepts [11, 25].

To illustrate reasoning about graded vulnerability, we look at how difficult it is to compromise the election through coercion. As it turns out, this requires a *coalitional* effort. Depending on the type of coercion, the coercer needs to team up with the voter or the eavesdropping intruder. Again, we identify coalitional strategies for coercion, and measure their complexity as well as necessary resources.

To create the models, we have used the UPPAAL model checker for distributed and multi-agent systems [5], 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 [29]. A more systematic approach, based on the idea of *security ceremonies*, was proposed and used in [6–8, 14, 46], and applied to formal

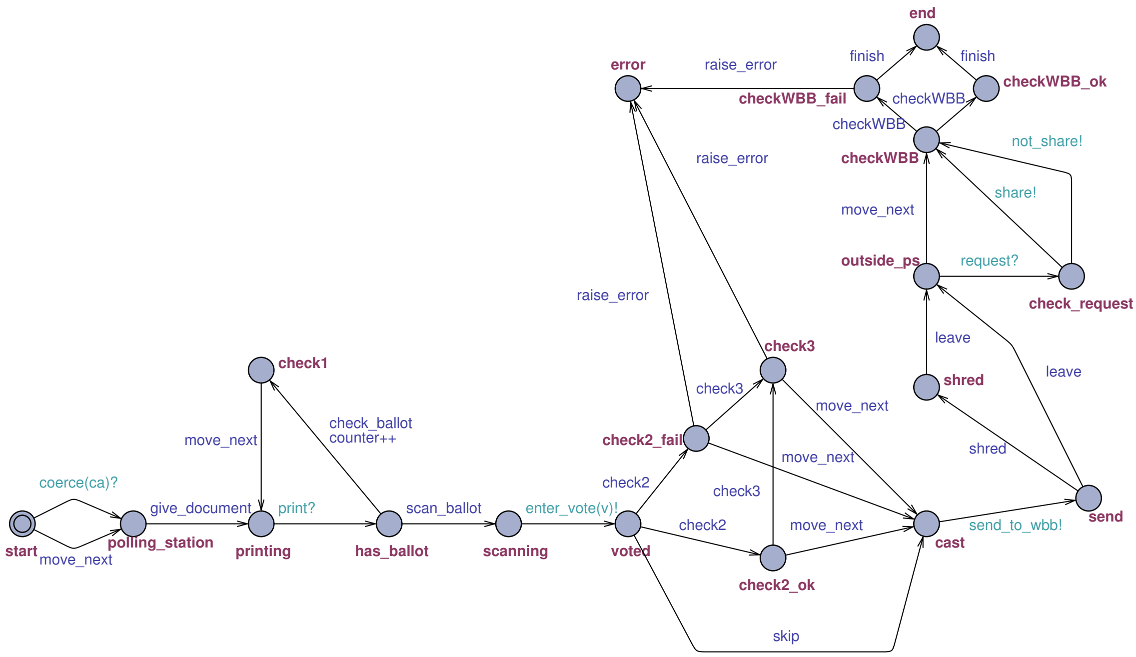


Fig. 1. Voter model

analysis of voting protocols in [45]. Here, we build on a different modeling tradition, namely on the framework of *multi-agent systems*. This modeling approach was only used in [30, 34]. In [30], a preliminary verification of the SELENE voting protocol was conducted. Moreover, [34] used UPPAAL to model the basic variant of Prêt à Voter and conduct tentative model checking of some interesting temporal and temporal-epistemic properties. 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.

Other (somewhat) related works include socio-technical modeling of attacks with timed automata [19] and especially game-theoretic analysis of voting procedures [3, 12, 18, 32]. Also, strategies for human users to obtain simple security requirements were investigated in [4]. Finally, specification of coercion-resistance and receipt-freeness in logics of strategic ability was attempted in [56].

A preliminary version of this article was published in the workshop paper [35].

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. 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. For instance, 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 local model, as well as the others, has been created using the modeling interface of the UPPAAL model checker [5]. The locations in UPPAAL are graphically represented as circles, with initial locations marked by a double circle. The edges are annotated by colored labels, depicting preconditions (also called guards, in green), synchronizations (in teal) and updates (in blue). The syntax of expressions is similar to 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 produces the 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 [56].

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 [2, 15, 47]. 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. 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 a central role in reasoning about purposeful agents [2, 54]. Formally, strategies are defined as functions from sequences of system states to actions. The simpler notion of *positional* strategies, that we will use here, is defined by 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, we proposed in [36, 37] to use a more human-friendly representation of strategies, based on lists of condition-action rules. The conditions are given by Boolean formulas for positional strategies and regular expressions over Boolean formulas in the general case. 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 [20] and planning [26], as well as the empirical results on how humans learn and use concepts [11, 25].

2.3. Natural Strategies and Their Complexity

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 positional strategies of agent a by *ordered 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. In this way, we guarantee that at least one action is available in each state of the system. 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.

By $length(s_a)$, we denote the number of guarded actions in s_a. Moreover, $cond_i(s_a)$ denotes the ith guard (condition) on the list, and $act_i(s_a)$ the corresponding action. Finally, $match(q, s_a)$ is the smallest $i \leqslant length(s_a)$ such that $q \models cond_i(s_a)$ and $act_i(s_a) \in d_a(q)$, where $d_a(q)$ are the available actions of agent a in state q. That is, $match(q, s_a)$ matches state q with the first condition in s_a that holds in q, and action available in q. 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:

$$out(q, s_A) = \{\lambda = q_0 q_1 \cdots \mid (q_0 = q) \ \wedge \ \forall_{i \geqslant 0} \exists_{\alpha_1, \ldots, \alpha_{|\mathcal{A}|}} . \, (a \in A \Rightarrow \alpha_a = act_{match(q_i, s_a)}(s_a)) \wedge$$
$$(a \notin A \Rightarrow \alpha_a \in d_a(q_i)) \wedge (q_{i+1} = succ(q_i, \alpha_1, \ldots, \alpha_{|\mathcal{A}|}))\}$$

where $succ(q, \alpha)$ is the successor state of state q given the tuple of actions α.

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 reflects its level of sophistication. Thus, it can be used to approximate the mental effort needed to come up with the strategy, memorize it, and execute it. Clearly, the approximation is not perfect, since the actual mental effort depends on the individual qualities of the agents, as well as the characteristics of the environment where the strategy is executed. For example, the interface of a voting system might present the voter with hints on how to verify one's vote; in that case, the mental effort to follow the verification strategy is obviously smaller. Some psychological evidence suggests that the approximation is a step in the right direction [11, 25]. A more accurate characterization might be obtained using the advances in the field of *user experience (UX)* [22, 28, 44]; we leave that path for future work.

2.4. Logical Specification of Natural Ability

To reason about natural strategic ability, the logic NatATL was introduced in [31, 36] with the following syntax:

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

where A is a group of agents and $k \in \mathbb{N}$ is a complexity bound. Intuitively, $\langle\langle A \rangle\rangle^{\leqslant k} \psi$ reads as "coalition A has a collective strategy of size less than or equal to k to enforce the temporal property ψ." 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 $\langle\langle cust \rangle\rangle^{\leqslant 10} \mathsf{F}\, \mathsf{ticket}$ 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.

Note that the standard strategic operator $\langle\langle A \rangle\rangle \psi$ can be expressed in NatATL by $\langle\langle A \rangle\rangle^{\leqslant \infty} \psi$. Moreover, the path quantifier "for all paths" from temporal logic can be defined as $\mathsf{A}\psi \equiv \langle\langle \emptyset \rangle\rangle^{\leqslant 0} \psi$.

In the rest of the paper, in addition to the classic atomic propositions, we will use for each atomic proposition p its *persistent* version, denoted by $\boxed{\mathsf{p}}$. The idea is that, once proposition p gets true for the first time, $\boxed{\mathsf{p}}$ also becomes true and remains true forever, even if p changes its truth value to false again.

3. Specification and Verification of Voting Properties Based on Natural Strategies

NatATL can be used to specify interesting properties of the voting system. In this section, we show tentative formalizations of such properties. We also discuss different levels of "strategic refinement", and show at which level the graded notion of security can be derived. The focus is on (broadly understood) security properties, but the same pattern of reasoning can be applied to other kinds of requirements, such as usability.

3.1. How to Specify Voter-Verifiability

The requirement of *voter-verifiability* captures the ability of the voter to verify what happened to her vote. In our case, this is represented by the *checkWBB* phase, hence we can specify voter-verifiability with the formula $\langle\langle voter\rangle\rangle^{\leqslant k}\mathsf{F}\ (\mathsf{checkWBB\_ok} \vee \mathsf{error})$. The intuition is simple: the voter has a strategy of size at most k to successfully perform *checkWBB* or else signal an error.

A careful reader can spot one problem with the formalization: it holds if the voter signals an error regardless of the outcome of the check (and it shouldn't!). A better specification is given by $\langle\langle voter\rangle\rangle^{\leqslant k}\mathsf{F}\ (\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$, saying that the voter has a strategy of size at most k so that, at some point, she obtains either the positive or the negative outcome of *checkWBB*.

3.2. Towards Dispute Resolution

We can use formula $\mathsf{AG}\ (\mathsf{checkWBB\_fail} \rightarrow \langle\langle voter\rangle\rangle^{\leqslant k}\mathsf{F}\ \mathsf{error})$ to connect the negative outcome of the check with the voter's ability to report the problem. This property, which can be called "error signalling," captures one aspect of *dispute resolution*. To characterize dispute resolution in full, we would need to significantly extend our model of the election. For instance, it would have to include a process that handles submitting the relevant evidence to the right authority (electoral commission, the judge, etc.), the deliberation and decision-making steps to be taken by that authority, and finally the way the final decision is to be executed (e.g., the election being declared void and repeated). We conjecture that dispute resolution would require not only more complex models than voter verifiability, but also higher mental complexity of the voter's behaviour, i.e., more complex natural strategies to achieve it.

3.3. Strategic-Epistemic Specification of Voter-Verifiability

The above specification of voter-verifiability is rather technical and relies on appropriate labeling of model states (in particular, with propositions $\mathsf{checkWBB\_ok}$ and $\mathsf{checkWBB\_fail}$). On a more abstract level, one would like to say that the voter has a strategy to eventually know how her vote has been treated. Crucially, this refers to the *knowledge* of the voter. To capture the requirement, one can extend NatATL with knowledge operators K_a, where $K_a\varphi$ expresses that agent a knows that φ holds. For instance, $K_{voter}\mathsf{voted_i}$ says that the voter knows that her vote has been registered for the candidate i. Then, voter-verifiability could be re-formalized as:

$$\langle\langle voter\rangle\rangle^{\leqslant k}\mathsf{F}\ \bigwedge_{i\in Cand}(K_{voter}\mathsf{voted_i} \vee K_{voter}\neg\mathsf{voted_i}).$$

3.4. Receipt-Freeness

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, even if the voter cooperates [38]:

$$\bigwedge_{i\in Cand} \neg\langle\langle coerc, voter\rangle\rangle^{\leqslant k}\mathsf{F}\,(\mathsf{end} \wedge (K_{coerc}\mathsf{vote_i} \vee K_{coerc}\neg\mathsf{vote_i})).$$

That means that the coercer and the voter have no strategy with complexity at most k to ensure that the coercer learns, after the election is finished, whether the voter has voted for i or not. Note that this is only one of the possible formalizations 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: $\bigwedge_{i\in Cand} \neg\langle\langle coerc, voter\rangle\rangle^{\leqslant k}\mathsf{F}\,(\mathsf{end} \wedge \neg\mathsf{vote_i} \wedge K_{coerc}\neg\mathsf{vote_i})$. We note in passing that the related notion of *vote anonymity* can be specified as $\bigwedge_{a\in Agents\setminus\{voter\}} \bigwedge_{i\in Cand} \mathsf{AG}\,(\neg K_a\mathsf{vote_i} \wedge \neg K_a\neg\mathsf{vote_i}))$.

3.5. Levels of Strategic Refinement

When talking about an important property of a voting system (such as receipt-freeness, voter-verifiability, and so on), one can consider at least three different conceptual variants, based on our assumptions about the strategic play of participating agents:

(1) The *temporal variant* takes the temporal formula ψ, and requires that it is satisfied in all (resp. no) possible runs of the system. In other words, the model must satisfy the branching-time formula $\mathsf{A}\psi$ (resp. $\mathsf{A}\neg\psi$). For example, the temporal variant of voter-verifiability is $\mathsf{AF}\,(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$, saying that the voter will eventually verify her vote on WBB, no matter what she (and anybody else) decides to do.
Similarly, the temporal variant of receipt-freeness is $\mathsf{AG}\,\neg(\mathsf{end} \wedge \neg\mathsf{vote_i} \wedge K_{coerc}\neg\mathsf{vote_i})$, that is, it expresses the vote anonymity with respect to the knowledge of the coercer at the end of the election.
Typically, ψ captures a trace property (e.g., $\mathsf{F}\,(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$). However, it can also express an indistinguishability property by combining temporal and epistemic operators, cf. our formalizations of receipt-freeness and anonymity.

(2) The *strategic refinement* takes ψ, and requires that it can (or cannot) be enforced by the relevant participants. Clearly, the temporal variant of voter-verifiability is too strong: we want to provide a mechanism so that the voter has the ability to verify her vote, and not that she is forced to do it. This is captured by the ATL formula $\langle\langle voter\rangle\rangle\mathsf{F}\,(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$.
The strategic refinement of receipt-freeness is constructed analogously by referring to the joint strategic ability of the voter and the coercer: $\neg\langle\langle coerc, voter\rangle\rangle\mathsf{F}\,(\mathsf{end} \wedge \neg\mathsf{vote_i} \wedge K_{coerc}\neg\mathsf{vote_i})$.

(3) The *graded strategic refinement* takes ψ, and demands that it can (resp. cannot) be enforced by the relevant participants within the given mental complexity k. For example, $\langle\langle voter\rangle\rangle^{\leqslant k}\mathsf{F}\,(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$ can express that the voter has a *simple* strategy to verify her vote. This can be used to construct a *graded notion of practical voter-verifiability*.
Moreover, the formula $\neg\langle\langle coerc, voter\rangle\rangle^{\leqslant k}\mathsf{F}\,(\mathsf{end} \wedge \neg\mathsf{vote_i} \wedge K_{coerc}\neg\mathsf{vote_i})$ can be used to obtain a *graded notion of vulnerability* for receipt-freeness. This can be useful if there exists a successful attack to compromise the basic property ψ. In that case, we can still distinguish between systems that require different levels of complexity from the attacker.

Summarizing, one can talk about the basic property ψ (typically, a trace property or an indistinguishability property that is supposed to hold on all the executions of the system), its strategic variant where we only require that the relevant agent(s) have a strategy to enforce the basic property, and its graded refinement where we only allow for bounded strategies. Moreover, the latter level allows to parameterize the security property with arbitrary bounds on the complexity of strategic play.

3.6. 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 neither allows for strategic operators nor knowledge modalities [34]. That said it is still a very practical tool when it comes to creating the models, thanks to the well-designed graphical interface and extended modelling functionalities. Other existing model checking tools (such as MCMAS [42]) do not offer such functionalities, although they provide better requirement specification language. Still, we can use UPPAAL to verify concrete strategies if we carefully modify the input formula and the model. We will show how to do it in Section 8.

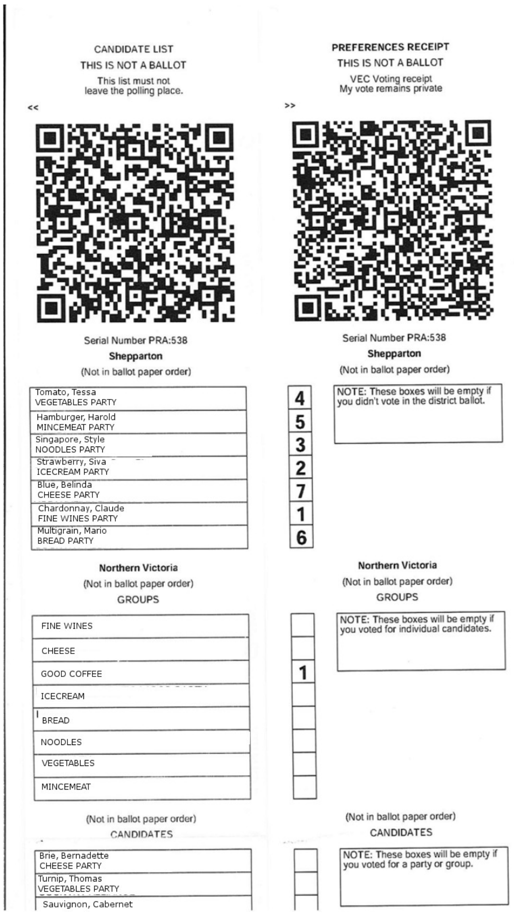
We also remark that combining strategic and epistemic aspects poses a number of semantic problems [1, 33]. In particular, one needs to choose the right notion of indistinguishability, and pair it with a matching type of strategies, available to the players. To avoid unnecessary distractions, in the rest of the paper we will concentrate on properties that use only strategic operators, such as the "technical" specification of voter-verifiability in Section 3.1.

4. Use Case Scenario: vVote

Secure and verifiable voting is becoming more and more important for democracy to function correctly. In this case study, 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 [17]. 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 casts her vote in the usual way, by placing a cross in the right hand column against the candidate of her choice. Then, she tears the ballot in two parts, destroys the left part, casts the right one, and takes a copy of it as her receipt. After the election her vote appears on the public Web Bulletin Board (WBB)[1] 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 Web Bulletin Board.

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 device (POD) specifying the district/region of the voter. If the authentication is valid (state *printing*) then the POD 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

[1] The WBB is an authenticated public broadcast channel with memory.


CANDIDATE LIST
THIS IS NOT A BALLOT
This list must not
leave the polling place.

PREFERENCES RECEIPT
THIS IS NOT A BALLOT
VEC Voting receipt
My vote remains private

Serial Number PRA:538
Shepparton
(Not in ballot paper order)

Tomato, Tessa VEGETABLES PARTY	4
Hamburger, Harold MINCEMEAT PARTY	5
Singapore, Style NOODLES PARTY	3
Strawberry, Siva ICECREAM PARTY	2
Blue, Belinda CHEESE PARTY	7
Chardonnay, Claude FINE WINES PARTY	1
Multigrain, Mario BREAD PARTY	6

NOTE: These boxes will be empty if you didn't vote in the district ballot.

Northern Victoria
(Not in ballot paper order)
GROUPS

FINE WINES	
CHEESE	
GOOD COFFEE	1
ICECREAM	
BREAD	
NOODLES	
VEGETABLES	
MINCEMEAT	

NOTE: These boxes will be empty if you voted for individual candidates.

(Not in ballot paper order)
CANDIDATES

Brie, Bernadette CHEESE PARTY	
Turnip, Thomas VEGETABLES PARTY	
Sauvignon, Cabernet	

NOTE: These boxes will be empty if you voted for a party or group.

Fig. 2. A vote printout in vVote [17]

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. 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 is a computer that assists the user in filling in a Prêt à Voter ballot. The EBM prints on a separate sheet the voter's 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, see Figure 2.

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 correct, the voter validates the vote, shreds the candidate list, and leaves the polling station. Finally,

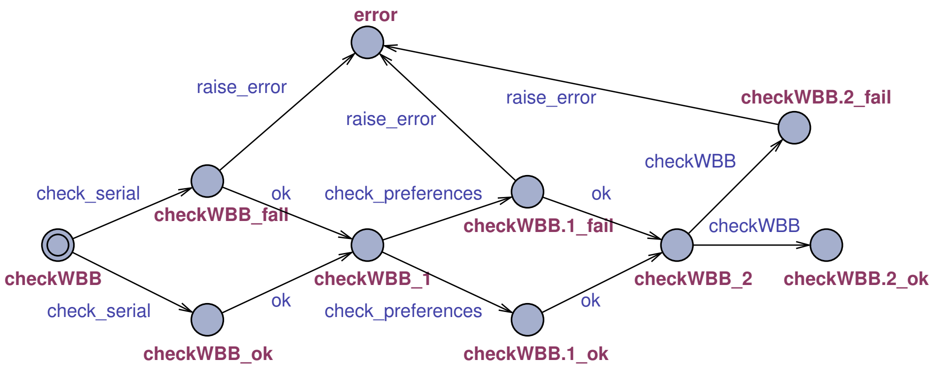


Fig. 3. Voter refinement: phase checkWBB

the voter can check her vote on the WBB after the election closes. She only needs to check the SN and the order of her preference numbers.

5. Models

In this section we present the model 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.

5.1. Voter Model

The local model already presented in Figure 1 captures the voter's actions, from her potential interaction with the coercer, through entering the polling station and casting her vote, to going back home and verifying her receipt on the Web Bulletin Board. As shown in the graph, some actions (in particular the additional checks) are optional for the voter. Furthermore, to simulate realistic human behavior, we included some additional actions, not described by the protocol itself. For example the voter can try to skip even obligatory steps, such as *check*2. This is especially important, as *check*2 may be the most time-consuming action for the voter and many voters may skip it in real life. To further simulate the real-life behavior of the voters, for each state we added a loop action labeled as *idle*, to allow the voter to wait for as long as she wants. We omit the loops from the graph for the clarity of presentation. Note that the actions colored in teal represent the synchronization actions. Given an action a, the label "a?" means that the voter has to wait that another agent does a to go in the next state, while "a!" means that when the voter selects a she determines also a transition in another local model. After every check, the voter can signal an error, thus ending up in the *error* state. The state represents the situation when communication is triggered with the election authority, signaling that the voter could not cast her vote or a machine malfunction was detected.

5.2. Refinements of the Voter Model

The model shown in Figure 1 is relatively abstract. For example, *checkWBB* is shown as an atomic action, but in fact it requires that the voter compares data from the receipt and the WBB. In order

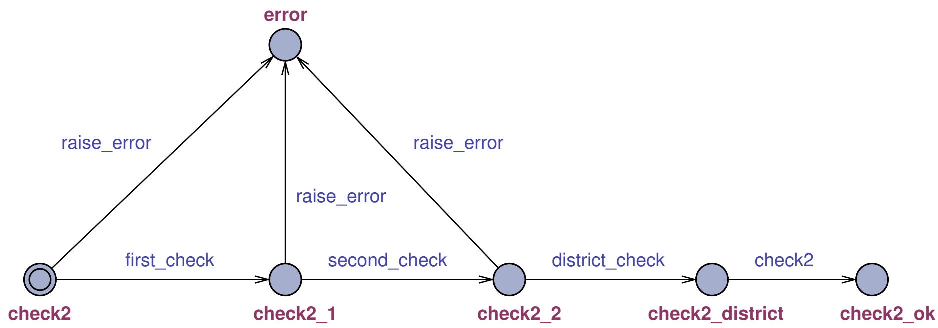


Fig. 4. Voter refinement: phase check2

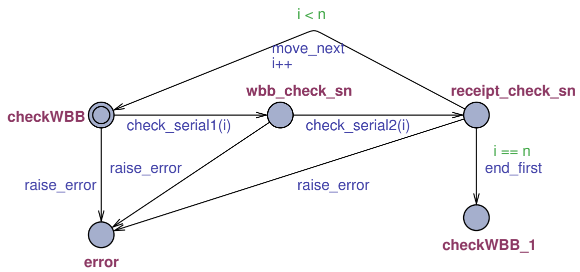


Fig. 5. Voter refinement: serial number check. Label *check_serial*(i) depicts checking the *i*th symbol of the serial number on the receipt and WBB

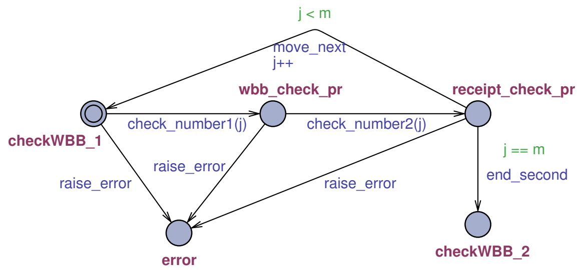


Fig. 6. Voter refinement: preferences order check. Label *check_number*(i) refers to checking the *i*th position on the receipt and WBB

to properly measure the complexity of the voter strategies, it is crucial to consider different levels of granularity.

CheckWBB phase. Recall that this is the last phase in the protocol and it is optional. Here, the voter can check if the printed receipt matches her recorded 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 $check\_preferences$). If both steps succeed, then the voter reaches state $checkWBB\_ok$. The refined model for this phase is presented in Figure 3.

Check2 phase. Other phases, such as *check*2, can be refined in a similar way. Recall that this is the only obligatory check phase; all the other ones are optional. Here, the voter should check that the printed receipt matches her intended vote. This includes checking that the serial numbers match (action $first\_check$), and that the printed preferences match her intended vote arranged according to the candidate order on her ballot (action *second_check*). So, if both the steps succeed, then the voter checks that the district is correct. A refined model for this phase is shown in Figure 4.

Serial number phase. In some cases the model shown in Figure 3 may still be too general. For example, the length of the serial number may have impact on the level of difficulty faced by the voter. To capture this, we split the step into atomic actions: *check_serial*1(i) for checking the *i*th symbol on the WBB, and *check_serial*2(i) for checking the *i*th symbol on the receipt. The resulting model is shown in Figure 5, where n is the length of the serial number.

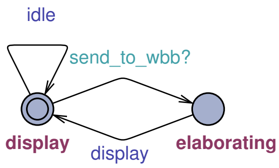


Fig. 7. Public WBB

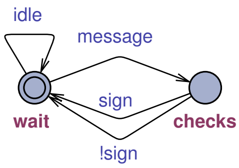


Fig. 8. Private WBB

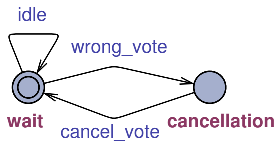


Fig. 9. Cancel station

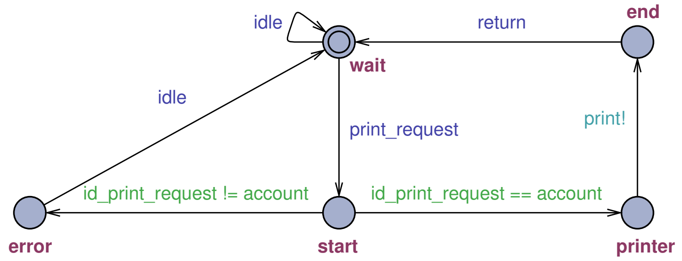


Fig. 10. Print-on-demand printer

Preferences order phase. Similarly to comparing the two serial numbers, the verification of 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 $check\_number2(i)$ refer to checking a number on the WBB and on the receipt, respectively. This is shown in the local model in Figure 6, where m is the number of candidates in the ballot.

5.3. 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 7–11.

Public WBB. In Figure 7 we present the public WBB. Simply, this component displays a new information when it receives a new one. The action *send_to_wbb* is synchronized with the Voter model and is executed after the voter has cast her vote.

Private WBB. In Figure 8 we present the private WBB. Here, for each message received, the system component decides to sign or not to sign the message.

Cancel station. In Figure 9 we present the cancel station. This component is a supervised interface for canceling a vote that has not been properly submitted or has not received a valid PWBB signature.

The three models described above have a very similar structure. In fact, each component waits for an external event and performs an action that returns in all the cases in the initial state.

Print-on-demand printer. Figure 10 models the behavior of the printing process. The initial state is *wait*, where the printer stays idle until another agent sends a print request. Then, the printer checks whether the agent that has made the request has an account. If this is the case, then the printer prints the

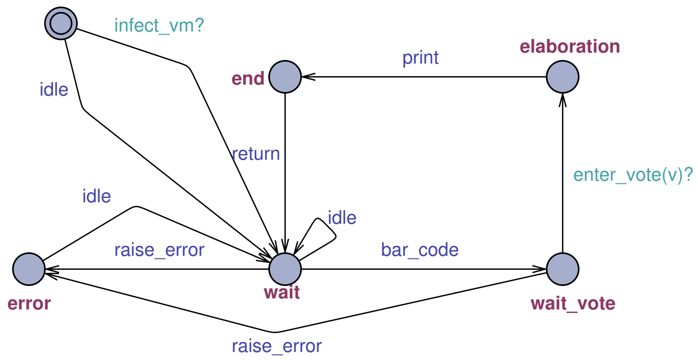


Fig. 11. Electronic Ballot Marker (EBM)

document, and returns to the waiting state. Otherwise, the printer does not accept the request and also returns to *wait*.

EBM. Finally, in Figure 11 we present the possible interactions of the Electronic Ballot Marker. At the beginning the EBM machine can be infected with the malicious software by the intruder. This is represented with the action $infect\_vm$ synchronized with the Intruder model. Next, the EBM receives a ballot (event) and checks if the bar code is correct. If the check succeeds, the EBM waits for the voter to enter her vote (synchronized action *enter_vote*). If the machine has been infected, the vote is also sent to the intruder, which means that this action will be synchronized between the three agents at once, where the Voter plays the role of sender, and both the EBM and the Intruder receive the vote. In this scenario we only consider one type of infection, which results in leaking the vote to the intruder. Different types of infection, such as causing the malfunction of the device, are left for future work. The last step is to print the vote.

In all other cases the EBM passes through the *error* state and then returns to the initial state (*wait*).

5.4. Opponent Model

To model the opponent, 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 a specified candidate? Notice that there are two spheres of interaction for the opponent: one with the voter and another with the system. To capture these aspects, we split our threat model into two agents: the *coercer* and the *intruder*. The former is used to model the adversarial interaction with the voter (threatening the voter, forcing her to change her vote, and potentially punishing for disobedience). The latter is used to model the interaction with the system (infecting the voting machine with malware, eavesdropping for the content of the ballot, and relaying it to the coercer).

Thus, we use the modular approach to modeling, provided by UPPAAL, in order to split the potential competences of the attacker into relevant subsets, and later combine them into appropriate threat models by looking at the abilities of *sets of agents*, i.e., coalitions. For example, the coercer that can only blackmail and punish is modeled by the singleton agent set $\{coercer\}$, while one which can also compromise the privacy in the system can be referred to through the coalition $\{coercer, intruder\}$.

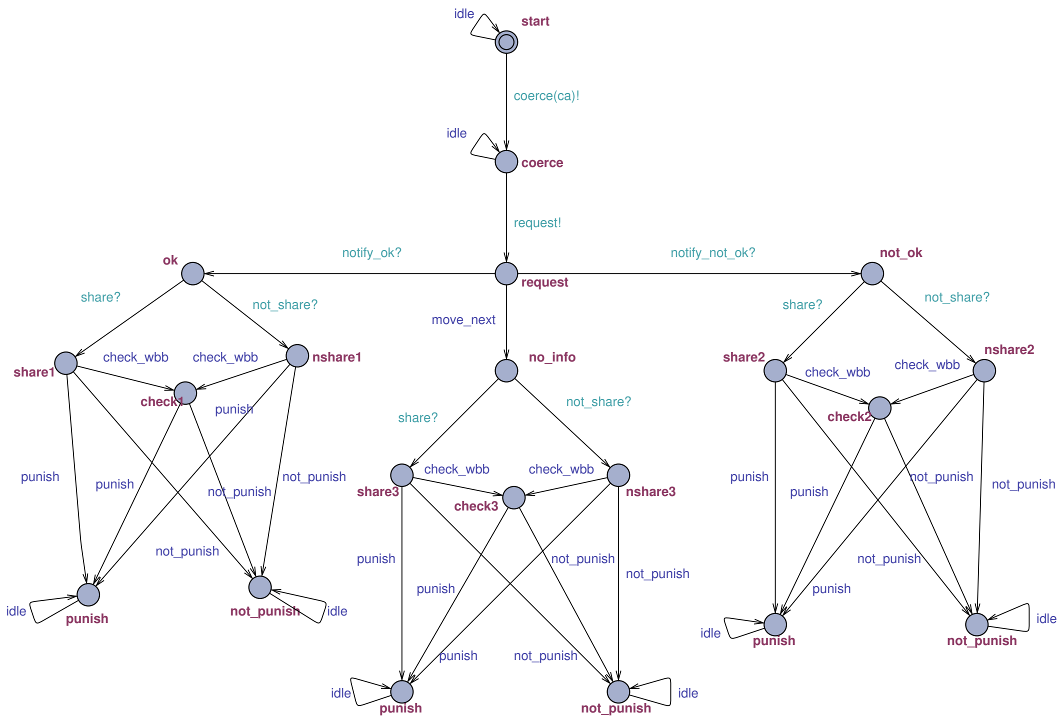


Fig. 12. Coercer model. States *punish* and *not_punish* are duplicated for better readability.

Coercer Model. The model of the coercer is depicted in Figure 12. Starting from the initial state (*start*), the coercer can remain in it by using the action *idle* and can move to state *coerce* by using the synchronized action $coerce(ca)$. The latter action means that the coercer coerces the voter to vote for the candidate *ca*. From *coerce*, he can wait or request the ballot receipt from the voter. To do this, the voter must have finished her actions at the polling station and proceeded to direct communication with the coercer. From this point (*request*) the next state depends on the coercer capabilities (in particular, whether he collaborates with the intruder). If he does, then the intruder notifies the coercer if the vote was cast for the candidate *ca* (action *notify_ok*) or not (action *notify_not_ok*). If he does not, then the coercer executes action *move_next*.

The next step depends on the voter's choice. In fact, if she decides to give the receipt to the coercer then the next state will be $share_i, i = 1, 2, 3$ while if she decides not to give the receipt to the coercer then the next state will be $nshare_i, i = 1, 2, 3$. Note that, in the latter situation, we capture also the cases in which the voter lost the receipt or she has not terminated the voting process. In the next step the coercer can decide to check the WBB (action *check_wbb*), or he can skip this step. Either way, the last action is to punish the voter or refrain from the punishment (actions *punish* and *not_punish*). After that, the coercer remains in the last state of his module.

Intruder Model. The model of the intruder is depicted in Figure 13. Starting from the initial state (*start*), the intruder can remain in *start* by using the action *idle* or he can move forward by selecting his preferred candidate ($select\_candidate(ca)$). Further, he can move to the state *infection* by using the action *infect_vm*. The latter means that the intruder can infect and take control of the voting machine. From *infection*, he can *idle* or eavesdrop on the voting machine to capture the voter's vote. In the latter

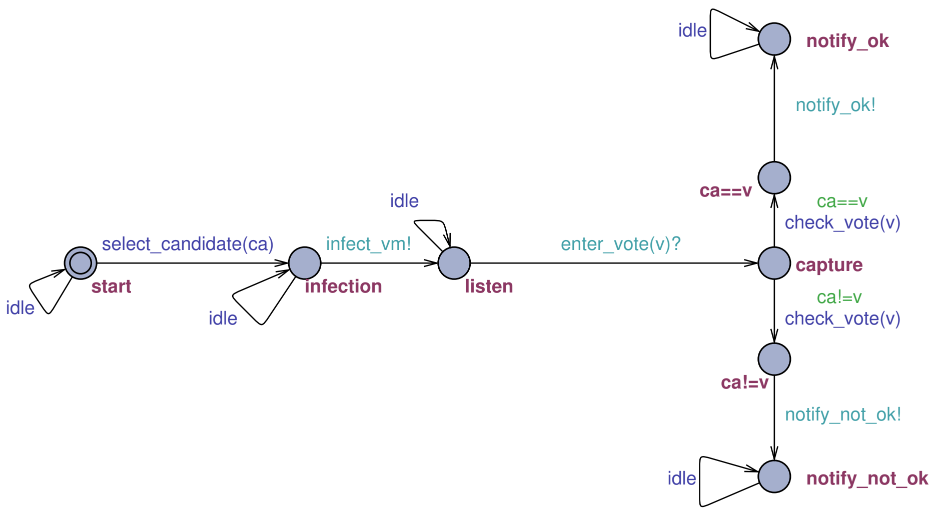


Fig. 13. Intruder model

case, the intruder compares the voter's vote with his preferred candidate *ca*. If the comparison shows the same candidate then the intruder proceeds to the state $ca = v$ and informs the coercer that the vote has been cast for the required candidate (action *notify_ok*). Otherwise he moves to the state $ca \neq v$ and informs the coercer about the discrepancy (action *notify_not_ok*). After that, the intruder remains in the last state of his module.

6. Assessing the Degree of Voter-Verifiability: Voter's 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 way he instructs, 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 Sections 2 and 3. 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 goal that the voter may want to pursue is the verification of her vote. Given the model in Figure 1, we can use the formula $\varphi_1 = \langle\langle voter\rangle\rangle^{\leqslant k}\mathrm{F}(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$, as discussed in Section 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, i.e., different admissible values of k. This is why we proposed several variants of the voter model in Section 5. In this section, we will show how it affects the outcome of the analysis. To this end, we take a closer look at the previously defined models, and try to list possible strategies for the participants.

6.1. Strategies for the Voter

In this section we focus on natural strategies for variants of voter-verifiability. Consider the following recipe for the voter's behavior, aimed at making $\varphi_1 = \langle\langle voter \rangle\rangle^{\leqslant k}\mathrm{F}(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$ true.

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

(1) $\mathsf{start} \vee \mathsf{check2\_ok} \vee \mathsf{check2\_fail} \vee \mathsf{outside\_ps} \rightsquigarrow move\_next$
(2) $\mathsf{polling\_station} \rightsquigarrow give\_document$
(3) $\mathsf{has\_ballot} \rightsquigarrow scan\_ballot$
(4) $\mathsf{scanning} \rightsquigarrow enter\_vote(v)$
(5) $\mathsf{voted} \rightsquigarrow check2$
(6) $\mathsf{cast} \rightsquigarrow send\_to\_wbb$
(7) $\mathsf{send} \rightsquigarrow shred$
(8) $\mathsf{shred} \rightsquigarrow leave$
(9) $\mathsf{check\_request} \rightsquigarrow not\_share$
(10) $\mathsf{checkWBB} \rightsquigarrow checkWBB$
(11) $\top \rightsquigarrow \star$

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 $\star$).

In Natural Strategy 1, we have 11 guarded commands in which the command (1) costs 7 since in its condition there are seven symbols (four atoms plus three disjunctions), while the other guarded commands cost 1, so the total complexity is $1 \cdot 10 + 7 \cdot 1 = 17$. So, the formula φ_1 is true with any k of 17 or more.

The next natural strategy comes with additional guarded commands in case the voter wants to do the optional phases *check*1 and *check*3. The strategy aims to satisfy the formula $\varphi_2 = \langle\langle voter \rangle\rangle^{\leqslant k}\mathrm{F}(\boxed{\mathsf{check1}} \wedge \boxed{\mathsf{check3}} \wedge (\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail}))$ which can be seen as a refinement of φ_1. In particular, φ_2 asks if there exists a natural strategy for the voter such that sooner or later she has executed *check*1, *check*3, and verified her vote. Besides ordinary atomic propositions such as check1 and check3, the formula uses also their *persistent* versions, denoted by $\boxed{\mathsf{check1}}$ and $\boxed{\mathsf{check3}}$.

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

(1) $\mathsf{start} \vee \mathsf{check1} \vee \mathsf{check3} \vee \mathsf{outside\_ps} \rightsquigarrow move\_next$
(2) $\mathsf{polling\_station} \rightsquigarrow give\_document$
(3) $\mathsf{has\_ballot} \wedge \mathsf{counter} == 0 \rightsquigarrow check\_ballot$
(4) $\mathsf{has\_ballot} \rightsquigarrow scan\_ballot$
(5) $\mathsf{scanning} \rightsquigarrow enter\_vote(v)$
(6) $\mathsf{voted} \rightsquigarrow check2$
(7) $\mathsf{check2\_ok} \vee \mathsf{check2\_fail} \rightsquigarrow check3$
(8) $\mathsf{cast} \rightsquigarrow send\_to\_wbb$
(9) $\mathsf{send} \rightsquigarrow shred$

(10) shred $\rightsquigarrow$ *leave*
(11) check_request $\rightsquigarrow$ *not_share*
(12) checkWBB $\rightsquigarrow$ *checkWBB*
(13) $\top \rightsquigarrow \star$

In Natural Strategy 2, we introduce the verification of *check*1 and *check*3. To do this we add two new guarded commands (3) and (7), and update some clauses such as (1). Note that, in (3) we use a counter to determine if *check*1 is done or not. This gives the total complexity of $1 \cdot 11 + 3 \cdot 2 + 7 \cdot 1 = 24$. Thus, the formula φ_2 is true for any $k \geqslant 24$.

6.2. Further Refinements

An important aspect of the strategic complexity arises from a more detailed analysis of the *checkWBB* phase. Some interesting questions are: how does the voter perform *checkWBB*? How does she compare the printed preferences with the information on the public WBB? These questions open up several scenarios both from a strategic point of view and for the model to be used. From the strategic point of view, we can consider a refinement of Natural Strategy 1, in which action *checkWBB* is divided into several more primitive steps. If we consider that the *checkWBB* includes: comparing preferences with the information in the public WBB and checking the serial number, we can already divide the single action into two different steps for each of the checks to be performed. Thus, given the model in Figure 3, to verify that the voter does each step of *checkWBB*, we need to provide a formula that verifies atoms checkWBB_ok, checkWBB.1_ok, and checkWBB.2_ok. To do this in NatATL, we use the formula $\varphi_3 = \langle\langle voter \rangle\rangle^{\leqslant k} \mathrm{F}((\boxed{\text{checkWBB\_ok}} \wedge \boxed{\text{checkWBB.1\_ok}} \wedge \boxed{\text{checkWBB.2\_ok}}) \vee \text{checkWBB\_fail} \vee \text{checkWBB.1\_fail} \vee \text{checkWBB.2\_fail})$. Consequently, we refine the previous natural strategy of the voter as follows.

Natural Strategy 3. *A strategy for the voter that works in the refined model of phase checkWBB is:*

(1) start $\vee$ check2_ok $\vee$ check2_fail $\vee$ outside_ps $\rightsquigarrow$ *move_next*
(2) polling_station $\rightsquigarrow$ *give_document*
(3) has_ballot $\rightsquigarrow$ *scan_ballot*
(4) scanning $\rightsquigarrow$ *enter_vote*(v)
(5) voted $\rightsquigarrow$ *check*2
(6) cast $\rightsquigarrow$ *send_to_wbb*
(7) send $\rightsquigarrow$ *shred*
(8) shred $\rightsquigarrow$ *leave*
(9) check_request $\rightsquigarrow$ *not_share*
(10) checkWBB $\rightsquigarrow$ *check_serial*
(11) checkWBB_ok $\rightsquigarrow$ *ok*
(12) checkWBB.1 $\rightsquigarrow$ *check_preferences*
(13) checkWBB.1_ok $\rightsquigarrow$ *ok*
(14) checkWBB.2 $\rightsquigarrow$ *checkWBB*
(15) $\top \rightsquigarrow \star$

In Natural Strategy 3, we have 15 guarded commands in which all the conditions are defined with a single atom but (1) in which there is a disjunction of four atoms. So, the total complexity is $1 \cdot 14 + 7 \cdot 1 = 21$. So, φ_3 is true for any $k \geqslant 21$; 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 5 and 6, we can consider a formula that checks whether the voter has a strategy that satisfies the following properties:

(1) sooner or later she enters the *checkWBB* phase;
(2) she verifies the symbols of the SN on the public WBB against 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 $\varphi_4 = \langle\langle voter \rangle\rangle^{\leqslant k} \mathrm{F}((\boxed{\mathsf{checkWBB}} \wedge \boxed{\mathsf{wbb\_check\_sn}} \wedge \boxed{\mathsf{receipt\_check\_sn}} \wedge \boxed{\mathsf{checkWBB.1}} \wedge \boxed{\mathsf{wbb\_check\_pr}} \wedge \boxed{\mathsf{receipt\_check\_pr}} \wedge \boxed{\mathsf{checkWBB.2}}) \vee \mathsf{checkWBB\_fail} \vee \mathsf{checkWBB.1\_fail} \vee \mathsf{checkWBB.2\_fail})$. We can define a natural strategy that satisfies φ_4, as follows.

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

(1) $\mathsf{start} \vee \mathsf{check2\_ok} \vee \mathsf{check2\_fail} \vee \mathsf{receipt\_check\_sn} \vee \mathsf{receipt\_check\_pr} \vee \mathsf{outside\_ps} \rightsquigarrow move\_next$
(2) $\mathsf{polling\_station} \rightsquigarrow give\_document$
(3) $\mathsf{has\_ballot} \rightsquigarrow scan\_ballot$
(4) $\mathsf{scanning} \rightsquigarrow enter\_vote(v)$
(5) $\mathsf{voted} \rightsquigarrow check2$
(6) $\mathsf{cast} \rightsquigarrow send\_to\_wbb$
(7) $\mathsf{send} \rightsquigarrow shred$
(8) $\mathsf{shred} \rightsquigarrow leave$
(9) $\mathsf{check\_request} \rightsquigarrow not\_share$
(10) $\mathsf{checkWBB} \rightsquigarrow check\_serial1$
(11) $\mathsf{wbb\_check\_sn} \rightsquigarrow check\_serial2$
(12) $\mathsf{receipt\_check\_sn} \wedge \mathsf{i} == \mathsf{n} \rightsquigarrow end\_first$
(13) $\mathsf{checkWBB\_1} \rightsquigarrow check\_number1$
(14) $\mathsf{wbb\_check\_pr} \rightsquigarrow check\_number2$
(15) $\mathsf{receipt\_check\_pr} \wedge \mathsf{j} == \mathsf{m} \rightsquigarrow end\_second$
(16) $\mathsf{checkWBB\_2} \rightsquigarrow checkWBB$
(17) $\top \rightsquigarrow \star$

To conclude, the above natural strategy has 17 guarded commands in which the conditions in (12) and (15) are conjunctions of two atoms, the condition in (1) is a disjunction of six atoms, and all the other conditions are defined with a single atom. Therefore, the complexity of Natural Strategy 4 is $1 \cdot 14 + 3 \cdot 2 + 11 \cdot 1 = 31$. So, the formula φ_4 is true with $k \geqslant 31$.

6.3. 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 $\langle\langle voter \rangle\rangle^{\leqslant k} \mathsf{F}\, \varphi$ and a natural strategy s_v of 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 $out(q, s_v)$, 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 goals and strategies presented in Section 6.1.

For example, for Natural Strategy 3, starting from the initial state, the voter needs $11+5 = 16$ steps in the worst case to achieve $(\boxed{\mathsf{checkWBB\_ok}} \wedge \boxed{\mathsf{checkWBB.1\_ok}} \wedge \boxed{\mathsf{checkWBB.2\_ok}}) \vee \mathsf{checkWBB\_fail} \vee \mathsf{checkWBB.1\_fail} \vee \mathsf{checkWBB.2\_fail}$. More precisely, 11 steps are needed to achieve *checkWBB* in the local model shown in Figure 1, and 5 more steps to reach $\mathsf{checkWBB.2\_ok} \vee \mathsf{checkWBB.2\_fail}$ in the refinement of the final section of the procedure (see Figure 3).

Similarly, the voter executing Natural Strategy 1 needs 12 steps to achieve the state $\mathsf{checkWBB\_fail}$ or the state $\mathsf{checkWBB\_ok}$. Finally, Natural Strategy 2 requires 16 steps to conclude the verification of the voter's vote.

6.4. Towards a Graded View of Usable Security

In the preceding subsections, we have presented a sequence of formalizations for the requirement of voter-verifiability, each next one stronger and more detailed than the previous ones. Then, we presented natural strategies for the voter to bring about their strategic variants, and calculated the complexity of those strategies as well as the time (i.e., the number of steps) necessary to achieve the goal. Based on this, one may compare the degree of "usable voter-verifiability" for different variants ψ_1, ψ_2 of the requirement, based on the standard notion of Pareto dominance. If each ψ_i requires a strategy of complexity ρ_i which achieves the goal in ξ_i steps, and ψ_1 Pareto-dominates ψ_2 (that is, $\rho_1 \leqslant \rho_2$, $\xi_1 \leqslant \xi_2$, and at least one of the inequalities is strict), then ψ_1 is intuitively easier to achieve than ψ_2.

For example, the basic property $\psi_1 = \mathrm{F}(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$ has strategic complexity of 17 and needs of 12 steps, whereas $\psi_2 = \mathrm{F}(\boxed{\mathsf{check1}} \wedge \boxed{\mathsf{check3}} \wedge (\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail}))$ has complexity 24 and needs of 16 steps, which suggests that, while the strategic variants of both ψ_1, ψ_2 are satisfied in our model of vVote, the former presents the voter with a lighter burden. We note in passing that ψ_2 is strictly stronger than ψ_1; thus, it seems to promise a *higher* level of security. It introduces additional verification checks, which should make the system more verifiable. However, this results in higher complexity values, which suggests that the *usable* security of the system is reduced.

The same conceptual pattern can be employed to compare two different voting protocols with respect to a given security property ψ: if the characterization of protocol P_1 (in terms of strategic complexity and time) Pareto-dominates the characterization of protocol P_2, then P_1 seems to have higher usable security towards ψ than P_2. Of course, the problem with comparing different protocols is that, so far, our analysis is very sensitive to the level of abstraction and granularity in the modeling. This could be seen in Section 6.2 where a refinement of our model of vVote resulted in a (seemingly) higher complexity of voter's strategies. How can one make sure that the security mechanisms being compared are modeled with the same granularity? We do not have an answer to this question yet. Clearly, without a common reference model (or metamodel), the numbers obtained in our computations are rather arbitrary. In this sense, our work is preliminary, and should be considered as the first step rather than a ready-to-use framework.

7. Quantifying the Degree of Vulnerability for Coercion-Resistance

In the previous section, we looked at the complexity of voters' strategies for voter-verifiability. That is, we tried to estimate how hard it is for the voters to obtain a property which is, in principle, satisfied by the voting system. Now we will consider the alternative situation, namely properties for which no such strategy exists and, in fact, the potential attackers have a strategy to compromise it. In some cases, it makes sense to consider the complexity of the available attack strategies, and assess the mental effort required to exploit the vulnerability.

As it happens, the coercion-related vulnerabilities require cooperation of the coercer with another agent: either the eavesdropping intruder, or the voter herself. This can be nicely used to demonstrate how the complexity of coalitional play is quantified in the framework of natural strategies.

7.1. Natural Strategies for the Coalition of the Coercer and the Voter

We consider coalitional attack strategies against a weak variant of coercion-resistance, stating that "the coercer cannot obtain the receipt of the voter's vote even if the voter cooperates with him." The opposite of the property can be formalized as:

$$\psi_1 = \langle\langle coerc, voter \rangle\rangle^{\leqslant k} \mathsf{F}\, (\mathsf{share1} \vee \mathsf{share2} \vee \mathsf{share3}).$$

While the receipts in Prêt à Voter and vVote do not reveal for whom the vote was cast, such a strategy can be used to construct a randomization attack [38]: it suffices that the coercer asks the voter to mark the first row in the ballot, no matter what the order of the candidates was, and later checks if the voter obeyed the instruction.

Appropriate natural strategies for the coalition $\{coerc, voter\}$, aiming at property ψ_1, are presented below.

Natural Strategy 5 (Coalitional strategy, the voter's part)**.**

(1) $\mathsf{start} \vee \mathsf{check2\_ok} \vee \mathsf{check2\_fail} \rightsquigarrow move\_next$
(2) $\mathsf{polling\_station} \rightsquigarrow give\_document$
(3) $\mathsf{has\_ballot} \rightsquigarrow scan\_ballot$
(4) $\mathsf{scanning} \rightsquigarrow enter\_vote$
(5) $\mathsf{voted} \rightsquigarrow check2$
(6) $\mathsf{cast} \rightsquigarrow send\_to\_wbb$
(7) $\mathsf{send} \rightsquigarrow shred$
(8) $\mathsf{shred} \rightsquigarrow leave$
(9) $\mathsf{check\_request} \rightsquigarrow share$
(10) $\top \rightsquigarrow \star$

Natural Strategy 6 (Coalitional strategy, the coercer's part)**.**

(1) $\mathsf{start} \rightsquigarrow coerce(ca)$
(2) $\mathsf{coerce} \rightsquigarrow request$
(3) $\mathsf{request} \rightsquigarrow move\_next$
(4) $\top \rightsquigarrow \star$

In Natural Strategy 5, we have 10 guarded commands in which all the conditions are defined with a single atom except for (1) which is a disjunction of three atoms (and hence includes five symbols altogether). Thus, the total complexity is $1 \cdot 9 + 5 \cdot 1 = 14$. Moreover, Natural Strategy 6 for the coercer has complexity 4 since it has three guarded commands with a single symbol. So, ψ_1 is true for any $k \geqslant 18$.

We can also consider the case in which if the coercer does not receive the receipt from the voter he punishes her, otherwise he does not punish her. This property can be formalized as:

$$\psi_2 = \langle\langle coerc, voter \rangle\rangle^{\leqslant k} F(((\boxed{\mathsf{share1}} \vee \boxed{\mathsf{share2}} \vee \boxed{\mathsf{share3}}) \wedge \mathsf{not\_punish}) \vee ((\boxed{\mathsf{nshare1}} \vee \boxed{\mathsf{nshare2}} \vee \boxed{\mathsf{nshare3}}) \wedge \mathsf{punish})).$$

For the voter, we can use again the Natural Strategy 5. The natural strategy for the coercer is presented below.

Natural Strategy 7 (Coalitional strategy, the coercer's part)**.**

(1) $\mathsf{start} \rightsquigarrow coerce(ca)$
(2) $\mathsf{coerce} \rightsquigarrow request$
(3) $\mathsf{request} \rightsquigarrow move\_next$
(4) $\mathsf{share1} \vee \mathsf{share2} \vee \mathsf{share3} \rightsquigarrow not\_punish$
(5) $\mathsf{nshare1} \vee \mathsf{nshare2} \vee \mathsf{nshare3} \rightsquigarrow punish$
(6) $\top \rightsquigarrow \star$

Natural Strategy 7 for the coercer has complexity 14 since it has six guarded commands in which four of them have a single atom and the remaining two have a disjunction of three atoms. So, ψ_2 is true for any $k \geqslant 28$.

7.2. Coalitional Strategies with Eavesdropping

Finally, we consider an important property stating that the coercing party can punish the voter if the voter disobeyed the coercer's demand, and refrain from punishment otherwise. To capture that, we formally model the attacker by the coalition of the coercer and the intruder. More precisely, we specify the property by means of the formula:

$$\psi_3 = \langle\langle coerc, intruder \rangle\rangle^{\leqslant k} F((\boxed{\mathsf{ok}} \wedge \mathsf{not\_punish}) \vee (\boxed{\mathsf{not\_ok}} \wedge \mathsf{punish})).$$

The natural strategies for the coalition are presented below.

Natural Strategy 8 (Coalitional strategy, the coercer's part)**.**

(1) $\mathsf{start} \rightsquigarrow coerce(ca)$
(2) $\mathsf{coerce} \rightsquigarrow request$
(3) $\mathsf{share1} \vee \mathsf{nshare1} \rightsquigarrow not\_punish$
(4) $\mathsf{share2} \vee \mathsf{nshare2} \rightsquigarrow punish$
(5) $\top \rightsquigarrow \star$

Natural Strategy 9 (Coalitional strategy, the intruder's part)**.**

(1) $\mathsf{start} \rightsquigarrow \mathit{select\_candidate}(ca)$
(2) $\mathit{infection} \rightsquigarrow \mathit{infect\_vm}$
(3) $\mathsf{capture} \rightsquigarrow \mathit{check\_vote}(v)$
(4) $\mathsf{ca} = \mathsf{v} \rightsquigarrow \mathit{notify\_ok}$
(5) $\mathsf{ca!} = \mathsf{v} \rightsquigarrow \mathit{notify\_not\_ok}$
(6) $\top \rightsquigarrow \star$

In Natural Strategy 8, we have 5 guarded commands in which all the conditions are defined with a single atom but (3) and (4) in which there are disjunctions of two atoms. So, the total complexity is $1 \cdot 3 + 3 \cdot 2 = 9$. In Natural Strategy 9, we have 6 guarded commands in which all the guarded commands cost 1, so the total complexity is 6. So, the formula ψ_3 is true with any k of 15 or more.

7.3. Discussion

In Section 6, we argued that even if the system is in principle secure, its security is reduced by the complexity of the voter's strategy. Here, we consider systems that are *not* secure in the first place. Still, it is sometimes worth looking at how hard it is to compromise the system. The validity of this kind of reasoning with respect to attack strategies depends on the underlying concept of the attacker. If we assume a powerful adversary who has (almost) unlimited resources and does not make mistakes, then the complexity of attack strategies plays a small role. On the other hand, many coercion scenarios involve human coercers who are neither skilled hackers nor good reasoners. In-house coercion by a family member is a prime example here. In that case, one may argue that the vulnerability is reduced by the complexity of the attacker's strategy.

Another remark concerns the use of coalitional strategies in our analysis. A careful reader might have noticed that the coalitions considered in Sections 7.1 and 7.2 serve different purposes. The former refers to the cooperation of different participants of the voting process (the coercer and the voter in this case). The latter offers an neat way of modularizing the threat model: we distribute different potential capabilities of the attacker between different agents, and compose them by considering the relevant "coalition."

8. 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 6. To verify selected formulas and the corresponding natural strategies, we need to modify several things, namely: *(i)* the formula, *(ii)* the natural strategy, and finally *(iii)* the model. We explain the modifications step by step.

Formula. To specify the required properties for the protocol, we have used a variant of strategic logic, i.e., NatATL. Unfortunately, UPPAAL supports neither NatATL nor plain ATL, but only a fragment of the branching-time temporal logic CTL. Thus, we cannot use UPPAAL to model-check the formulas of Section 6. What we can do, however, is to verify if *a given natural strategy achieves a given goal*. To this end, we replace the strategic operator $\langle\langle A\rangle\rangle^{\leqslant k}$ in the formula with the universal path quantifier A ("for all paths"). For example, instead of formula $\varphi_1 \equiv \langle\langle v\rangle\rangle \mathsf{F}\,(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$ we use $\varphi'_1 = \mathsf{AF}\,(\mathsf{checkWBB\_ok} \vee \mathsf{checkWBB\_fail})$. At this point we do not differentiate between the persistent and standard propositions, as they are handled at the model level. Furthermore, we "prune" the model according to the given strategy, see below for the details.

Natural Strategy. In order to efficiently merge the natural strategy with the model, the strategy should 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 (conjunctively) the negated preconditions from all the previous guards. Furthermore, if the strategy includes multiple entries $\phi_1 \rightsquigarrow \alpha, \dots, \phi_k \rightsquigarrow \alpha$ for the same action α, they are all merged into a single entry: $\phi_1 \vee \cdots \vee \phi_k \rightsquigarrow \alpha$.

For example, Natural Strategy 1 becomes:

(1) $\mathsf{has\_ballot} \rightsquigarrow \mathit{scan\_ballot}$
(2) $\neg\mathsf{has\_ballot} \wedge \mathsf{scanning} \rightsquigarrow \mathit{enter\_vote}$
(3) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \mathsf{voted} \rightsquigarrow \mathit{check2}$
(4) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge (\mathsf{check2\_ok} \vee \mathsf{check2\_fail}) \rightsquigarrow \mathit{move\_next}$
(5) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail}) \wedge \mathsf{cast} \rightsquigarrow \mathit{send\_to\_wbb}$
(6) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail}) \wedge \neg\mathsf{cast} \wedge \mathsf{send} \rightsquigarrow \mathit{shred}$
(7) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail}) \wedge \neg\mathsf{cast} \wedge \neg\mathsf{send} \wedge \mathsf{shred} \rightsquigarrow \mathit{leave}$
(8) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail}) \wedge \neg\mathsf{cast} \wedge \neg\mathsf{send} \wedge \neg\mathsf{shred} \wedge \mathsf{check\_request} \rightsquigarrow \mathit{not\_share}$
(9) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail}) \wedge \neg\mathsf{cast} \wedge \neg\mathsf{send} \wedge \neg\mathsf{shred} \wedge \neg\mathsf{check\_request} \wedge \mathsf{checkWBB} \rightsquigarrow \mathit{checkWBB}$
(10) $\neg\mathsf{has\_ballot} \wedge \neg\mathsf{scanning} \wedge \neg\mathsf{voted} \wedge \neg(\mathsf{check2\_ok} \vee \mathsf{check2\_fail}) \wedge \neg\mathsf{cast} \wedge \neg\mathsf{send} \wedge \neg\mathsf{shred} \wedge \neg\mathsf{check\_request} \wedge \neg\mathsf{checkWBB} \rightsquigarrow \star$

Model. Semantically, a strategy of player a serves as a behavioral constraint that restricts the possible transitions in the module of a. To verify the selected strategy of the voter specified in the formula of NatATL, we merge the strategy with the voter model by adding the guard conditions from the strategy to the preconditions of the corresponding local transitions in the model. Thus, we effectively remove all the transitions that are not in accordance with the strategy.

This is done as follows: for every guarded command $\phi \rightsquigarrow \alpha$ in the strategy, find all the transitions in the module of the voter labeled by α and update their preconditions conjunctively with ϕ. That is, if the original precondition was ψ, it now becomes $\psi \wedge \phi$.

In consequence, only the paths that are consistent with the strategy will be considered by the model-checker. This of course requires the proper handling of the persistent propositions and variables. They need to be defined in the model in such way, that once the value is assigned to them, it will never be changed.

Levels of granularity. As we showed in Section 5, 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 *checkWBB*, we added synchronization edges in the voter model (Figure 1) and in the *checkWBB* model (Figure 3). Then, when going through the *checkWBB* phase in the voter model, UPPAAL will proceed to the more detailed submodel, and come back after getting to its final state.

Automated script. The procedure described above is troublesome for larger models and prone to errors if executed manually. Because of that, we have created a script in Python to automatically modify the xml file with the UPPAAL model according to the specified strategy. The script can be seen as an external plugin for UPPAAL. The only additional requirement is that the action names used in the natural strategy must be put in the comments section of the corresponding transition. The script takes two parameters as input: the xml file containing the specification of the model,

Voters	φ_1	φ_2	φ_3	φ_4
1	< 1	< 1	< 1	3
2	< 1	< 1	1	77
3	3	8	18	> 120
4	58	> 120	> 120	> 120
5	> 120	> 120	> 120	> 120

Table 1

Verification time in seconds of the formulas $\varphi_1, \ldots, \varphi_4$

Voters	ψ_1	ψ_2	ψ_3	ψ_3*
1	< 1	< 1	> 120	< 1
2	< 1	< 1	> 120	< 1
3	< 1	< 1	> 120	< 1
4	< 1	< 1	> 120	< 1
5	< 1	< 1	> 120	< 1

Table 2

Verification time in seconds of the formulas ψ_1, ψ_2, and ψ_3

and the text file with the natural strategy as well as the name of the agent. The output is the modified xml file with added guards to the specified transitions. In case of a coalitional strategy, the tool needs to be run multiple times, once for each agent and its strategy. The script is available at https://github.com/blackbat13/stv/blob/development/stv/parsers/uppaal_parser.py.

Running the verification. Following the procedure explained before, we have modified models, formulas, and strategies from Section 6. To apply natural strategies to models we used an automated script. Then, we used UPPAAL to verify that Natural Strategies 1–9 indeed enforce the prescribed properties, and measured the running time of the model checking procedure. We considered scenarios with two candidates, multiple voters, one coercer and one intruder. As the scaling factor we chose the number of the voters.

Experimental results for strategies of the voter. We begin by discussing the performance of model checking for strategies of the voter. The results of the experiments are presented in Table 1. The columns refer to the number of voters and the verified formulas. The model checking time is given in seconds. The timeout was set to 120 s. The verification output for each formula was always the same: the property holds in the model.

As the results show, we were able to finish the verification within the timeout for up to 4 voters for the simplest formula φ_1, and up to 2 voters for the formula φ_4. The more complicated the formula or the model is, the more time it takes to finish the verification process. For example, verifying the formula φ_4 for 2 voters took more time than verifying the formula φ_1 for 4 voters. The explanation is simple: the UPPAAL model considered in the formula φ_4 is more detailed, and includes more local states and transitions. Furthermore, it contains some (finite) loops in state templates (i.e., groups of related states, associated with the same node in the graph, but differing by the values of some underlying variables). Checking the serial number on the ballot is a good example of such a loop.

Experimental results for coalitional strategies. The second part of the experiments concerned coalition formulas ψ_1, ψ_2, and ψ_3. The setting was the same as before: 2 candidates, one coercer, one intruder and a scalable number of voters. Without loss of generality, we assumed that the coercer can only interact with one arbitrarily chosen voter. This can be extended in a simple way by introducing several coercers or by adding new variables to the coercer model, that would hold the information about the coercer's interactions with different voters. As previously, we prepared the models and the formulas, and ran the verification with UPPAAL. The output of the verification was the same as before: the properties hold in the model, except for the formula ψ_3 (see below). The performance results are presented in Table 2.

As we can see, the verification times form a surprising pattern, compared to the previous set of experiments. Each time, the verification process took approximately 1 second, even for more voters. It seems that introducing new voters does not affect the verification time significantly when only one voter is interacting with the coercer. The reason may come from the fact that the other voters do not affect the actions of the chosen voter and the coercer.

For formula ψ_3, UPPAAL did not complete the verification within the assumed timeout. This was probably because the voter can infinitely loop on the check1 phase, within an *infinite* state template. That is, each transition in the loop increases the value of the unbounded variable *counter*. Limiting the transitions along the loop to a finite number made UPPAAL verify the property in under 1 second, which is presented in the column labeled by ψ_3*.

9. Conclusions

Each voting protocol is designed to provide a certain set of functionalities to the voter. Consequently, in the analysis of a protocol, it is important to make sure that the voter has a strategy to use those functionalities. 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 measure its complexity as the size of the representation. Among other things, this can help to assess the difficulty associated with obtaining relevant security properties, such as voter-verifiability, receipt-freeness, and coercion-resistance.

In this paper, we make the first step towards a graded notion of security, based on the complexity of the participants' strategies that must be used to obtain a given temporal or temporal-epistemic pattern. We identify three relevant levels of formalizing security that consist of the basic trace/indistinguishability property, its strategic refinement, and the graded variant of the strategic refinement from which the graded view of security can be derived. We also propose that, in case the security property does not hold on the strategic level, the graded refinement can provide means to assess how vulnerable the system is.

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, such as the time needed to execute the strategy or the financial cost of the strategy. This may lead to trade-offs 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 is probably more relevant when we analyze the strategy of the coercer. Clearly, finding an optimal strategy in such settings may require to solve a multicriterial optimization problem [49, 58], e.g., by identifying the Pareto frontier and choosing a criterion to select a point on the frontier. We leave a closer study of such trade-offs for future work.

We emphasize that the work presented in this paper is preliminary, and should be considered as the first step rather than a ready-to-use framework. It is evident from the presented examples that the numbers obtained in our computations are, to a large degree, arbitrary. We believe that the idea can be further developed into a more robust methodology. One way to proceed is to identify empirical experiments that help to assess some of the values in a psychologically meaningful way. To achieve this, we plan to harness the recent developments in User eXperience methods [22, 28, 44], for example the concept of forcing functions as a means to influence the user's behavior by controlling their cognitive effort [55].

It would also be interesting 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 [4]. Consequently, 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. Then, the success level of a strategy can be computed, e.g., by using the probabilistic model checker PRISM [41].

Finally, we point out that the methodology proposed in this paper can be applied outside of 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.

Acknowledgements. The authors thank the anonymous reviewers of STAST for their valuable comments. W. Jamroga and D. Kurpiewski acknowledge the support of the National Centre for Research and Development, Poland (NCBR), and the Luxembourg National Research Fund (FNR), under the PolLux/FNR-CORE projects VoteVerif (POLLUX-IV/1/2016) and STV (POLLUX-VII/1/2019).

References

[1] T. Ågotnes, V. Goranko, W. Jamroga, and M. Wooldridge. Knowledge and ability. In H.P. van Ditmarsch, J.Y. Halpern, W. van der Hoek, and B.P. Kooi, editors, *Handbook of Epistemic Logic*, pages 543–589. College Publications, 2015.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.

[3] 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.

[4] 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.

[5] 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.

[6] Giampaolo Bella and Lizzie Coles-Kemp. Layered analysis of security ceremonies. In *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 273–286. Springer, 2012.

[7] 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.

[8] Giampaolo Bella, Paul Curzon, and Gabriele Lenzini. Service security and privacy as a socio-technical problem. *J. Comput. Secur.*, 23(5):563–585, 2015.

[9] 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.

[10] Matthew Bernhard, Allison McDonald, Henry Meng, Jensen Hwa, Nakul Bajaj, Kevin Chang, and J. Alex Halderman. Can voters detect malicious manipulation of ballot marking devices? In *IEEE Symposium on Security and Privacy*, pages 679–694. IEEE, 2020.

[11] L. E. Bourne. Knowing and using concepts. *Psychol. Rev.*, 77:546–556, 1970.

[12] 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.

[13] Craig Burton, Chris Culnane, and Steve A. Schneider. Secure and verifiable electronic voting in practice: the use of vVote in the Victorian state election. *CoRR*, abs/1504.07098, 2015.

[14] 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.

[15] K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. *Information and Computation*, 208(6):677–693, 2010.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] E. Davis and G. Marcus. Commonsense reasoning. *Communications of the ACM*, 58(9):92–103, 2015.

[21] 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.

[22] Verena Distler, Marie-Laure Zollinger, Carine Lallemand, Peter B. Rønne, Peter Y. A. Ryan, and Vincent Koenig. Security - visible, yet unseen? In *Proceedings of Conference on Human Factors in Computing Systems, CHI*, page 605. ACM, 2019.

[23] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. A formal taxonomy of privacy in voting protocols. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6710–6715. IEEE, 2012.

[24] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[25] J. Feldman. Minimization of Boolean complexity in human concept learning. *Nature*, 407:630–3, 11 2000.

[26] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[27] Kristian Gjøsteen and Anders Smedstuen Lund. An experiment on the security of the Norwegian electronic voting protocol. *Ann. des Télécommunications*, 71(7-8):299–307, 2016.

[28] Marc Hassenzahl and Noam Tractinsky. User experience-a research agenda. *Behaviour & Information Technology*, 25(2):91–97, 2006.

[29] 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.

[30] 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.

[31] W. Jamroga, V. Malvone, and A. Murano. Reasoning about natural strategic ability. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 714–722. IFAAMAS, 2017.

[32] 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.

[33] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63(2–3):185–219, 2004.

[34] Wojciech Jamroga, Yan Kim, Damian Kurpiewski, and Peter Y. A. Ryan. Towards model checking of voting protocols in uppaal. In *Proceedings of E-Vote-ID*, volume 12455 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 2020.

[35] Wojciech Jamroga, Damian Kurpiewski, and Vadim Malvone. Natural strategic abilities in voting protocols. In *Proceedings of STAST 2020*, 2021. To appear.

[36] Wojciech Jamroga, Vadim Malvone, and Aniello Murano. Natural strategic ability. *Artificial Intelligence*, 277, 2019.

[37] 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.

[38] 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.

[39] Oksana Kulyk, Melanie Volkamer, Monika Müller, and Karen Renaud. Towards improving the efficacy of code-based verification in internet voting. In *Financial Cryptography and Data Security Workshops, Revised Selected Papers*, volume 12063 of *Lecture Notes in Computer Science*, pages 291–309. Springer, 2020.

[40] 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.

[41] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: probabilistic symbolic model checker. In *Proceedings of TOOLS*, volume 2324 of *Lecture Notes in Computer Science*, pages 200–204. Springer, 2002.

[42] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: an open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.*, 19(1):9–30, 2017.

[43] Karola Marky, Oksana Kulyk, Karen Renaud, and Melanie Volkamer. What did I really vote for? In *Proceedings of Conference on Human Factors in Computing Systems, CHI*, page 176. ACM, 2018.

[44] Karola Marky, Marie-Laure Zollinger, Markus Funk, Peter Y. A. Ryan, and Max Mühlhäuser. How to assess the usability metrics of e-voting schemes. In *Financial Cryptography Workshops*, volume 11599 of *LNCS*, pages 257–271. Springer, 2019.

[45] T. Martimiano, E. Dos Santos, M. Olembo, and J.E. Martina. Ceremony analysis meets verifiable voting: Individual verifiability in Helios. In *SECURWARE*, 2015.

[46] 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.
[47] 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.
[48] T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols*, pages 25–35. Springer, 1998.
[49] Roxana Radulescu, Patrick Mannion, Diederik M. Roijers, and Ann Nowé. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi Agent Systems*, 34(1):10, 2020.
[50] 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.
[51] P.Y.A. Ryan. The computer ate my vote. In *Formal Methods: State of the Art and New Directions*, pages 147–184. Springer, 2010.
[52] F.P. Santos. *Dynamics of Reputation and the Self-organization of Cooperation*. PhD thesis, University of Lisbon, 2018.
[53] 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.
[54] Y. Shoham and K. Leyton-Brown. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.
[55] Mads Soegaard and Rikke Friis Dam, editors. *The Glossary of Human Computer Interaction*. Interaction Design Foundation.
[56] 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.
[57] Verified Voting. Policy on direct recording electronic voting machines and ballot marking devices. 2019.
[58] Stanley Zionts. A multiple criteria method for choosing among discrete alternatives. *European Journal of Operational Research*, 7(2):143–147, 1981. Fourth EURO III Special Issue.