Model Checking the SELENE E-Voting Protocol in Multi-Agent Logics

Wojciech Jamroga, Michal Knapik, and Damian Kurpiewski

Institute of Computer Science, Polish Academy of Sciences
{w.jamroga,michal.knapik,damian.kurpiewski}@ipipan.waw.pl

Abstract Selene is a recently proposed voting protocol that provides reasonable protection against coercion. In this paper, we make the first step towards a formalization of selected features of the protocol by means of formulae and models of *multi-agent logics*. We start with a very abstract view of the protocol as a public composition of a secret bijection from tracking numbers to voters and a secret mapping from voters to their choices. Then, we refine the view using multi-agent models of strategic interaction. The models define the space of strategies for the voters, the election authority, and the potential coercer. We express selected properties of the protocol using the strategic logic $\mathbf{ATL}_{\mathrm{ir}}$, and conduct preliminary verification by model checking. While $\mathbf{ATL}_{\mathrm{ir}}$ allows for intuitive specification of requirements like coercion-resistance, model checking of $\mathbf{ATL}_{\mathrm{ir}}$ is notoriously hard. We show that some of the complexity can be avoided by using a recent approach of *approximate model checking*, based on fixpoint approximations.

1 Introduction

Designing protocols for secure and verifiable voting is a difficult task. In this work, we present an attempt to use the techniques from multi-agent systems (MAS) in modeling and verification of e-voting. Agents in such systems are equipped with a larger degree of freedom than typical entities in a security protocol. They can have clearly defined objectives, capabilities, and knowledge about the world; they can also form coalitions working towards a joint goal. The benefits of MAS become especially noticeable in analysis of scenarios that involve interaction between human and technical agents, such as electronic voting.

Here, we come up with a simple MAS model of the recently proposed Selene protocol [19], and characterize several variants of coercion resistance with formulae of Alternating-time Temporal Logic (**ATL** [1]). Coercion resistance is essential in modern elections, and relies on the ability of voters to vote as they intend, and avoid the consequences of not obeying the coercer. Such requirements can be conveniently represented by **ATL** formulae following the scheme:

$$\neg\langle\langle \mathit{Coercer}\rangle\rangle\, \mathrm{F}\,\big(\textsf{election ends} \wedge (\textsf{voters have not obeyed}) \rightarrow (\mathit{Coercer}\ \textsf{knows})\big),$$

interpreted as *"The Coercer has no strategy to make sure that, when the election is over, he will detect disobedience of the coerced voters."* We use the semantics of **ATL**, based on memoryless imperfect information strategies, where agents' strategies assign choices of action to the agents' states of knowledge, rather than global states of the system. This variant of the logic is often referred to as $\mathbf{ATL}_{\mathrm{ir}}$ [20]. It is well known that model checking of $\mathbf{ATL}_{\mathrm{ir}}$ is $\mathbf{\Delta_2^P}$-complete [9,20] and does not have a natural fixpoint characterization [5]. To overcome the prohibitive complexity, we utilize a recently proposed idea of approximate verification based on fixpoint approximations of $\mathbf{ATL}_{\mathrm{ir}}$ formulae [10].

The article is organized as follows. We describe Selene, and discuss some of its formal aspects in Section 2. Then, we propose a multi-agent model of the protocol in Section 3. In Section 4, we present a brief introduction to $\mathbf{ATL}_{\mathrm{ir}}$, and propose some $\mathbf{ATL}_{\mathrm{ir}}$ formalizations of coercion-resistance. Section 5 reports our attempt at model checking the formulae from Section 4 in the models from Section 3. We conclude in Section 6, and discuss plans for future work.

1.1 Related Work

Over the years, the properties of *receipt-freeness* and *coercion resistance* were recognized as important for an election to work properly. They were studied and formalized in [3,7,8,13,18], see also [17,21] for an overview.

A number of papers used variants of epistemic logic to characterize coercion resistance [11,12]. Moreover, the agent logic **CTLK** together with the modeling methodology of interpreted systems was used to specify and verify properties of cryptographic protocols, including authentication protocols [4,16], and key-establishment protocols [4]. In particular, [4] used variants of the MCMAS model checker to obtain and verify models, automatically synthesized from high-level protocol description languages such as CAPSL, thus creating a bridge between multi-agent and process-based methods.

Our approach is closest to [21] where **ATL**-style formulae were used to encode different flavors of coercion resistance. However, the encodings in [21] were rather informal and imprecise, since neither formal semantics nor concrete model was given to interpret the formulae. In contrast, we use a precise semantics and provide a scalable class of models. Moreover, we use the formulae, the models, and the semantics to conduct verification of the protocol by model checking. Finally, [2] proposed a very simple attempt at model checking of Rivest's ThreeBallot protocol using $\mathbf{ATL}_{\mathrm{ir}}$, but the focus was on devising a model equivalence, and ThreeBallot served only to illustrate the idea.

2 Modeling Selene

We begin with a description of Selene, followed by a very abstract view of the conceptual backbone of the protocol. After that, we will move on to a more concrete model in Section 3.

2.1 Outline of Selene

Selene [19] has been proposed recently as a protocol for electronic voting targeted at low-coercion environments. The implemented cryptographic mechanisms should allow the voter to convince the coercer that the voter voted according to the coercer's request. One of the main advantages of the protocol is that, from the voter's perspective, the cryptography is put under the bonnet.

Roughly speaking, Selene works as follows. The Election Authority executes the initial setup of the system, which includes generation of the election keys and preparation of the cryptographic vote trackers, one for each voter. The trackers are then encrypted and mixed, and published on the Web Bulletin Board (WBB). The aim is to break any link between the voter and her encrypted tracker. Hence, the pool of trackers is public, while the assignment of the trackers is secret.

In the voting phase, each voter fills in, encrypts, and signs her vote. The signed and encrypted ballot is then collected by the system. After several intermediate steps, a pair $(\mathit{Vote}_v, \mathit{tr}_v)$ is published in WBB for each $v \in \mathit{Voters}$, where Vote_v and tr_v are, respectively, the decrypted ballot and the tracker of v. At this stage, no voters know their tracker numbers. All the cast votes are presented in plaintext in WBB.

The final stage consists of the notification of tracker numbers. If the voter is not coerced, then she requests the special α_v term, which allows for obtaining the correct tracker tr_v. If some pressure was exerted on the voter to fill her ballot in a certain way, she sends a description of the requested vote to the election server. A fake α'_v term is sent, which can be presented to the coercer. The α'_v token, together with the public commitment of the voter, reveals a tracker pointing out to a vote compatible with the coercer's demand, assuming that there is one.

2.2 An Abstract View of the Protocol

We now propose a convenient way of describing the scheme behind Selene at the abstract level. Social choice can be seen as a function that, given a set of voters, produces a collective decision for the society. This can be decomposed into a mapping between voters and their individual choices, and a mapping from the choices to the collective decision. End-to-end voter-verifiable protocols strive to make the former individually verifiable (so that each voter can verify her part of the function), and the latter universally verifiable (so that the whole function can be verified by everybody). On the other hand, coercion resistant protocols strive to make the first part secret to anybody except for the voter in question. Selene's idea of how to combine the two objectives is to further decompose the connection between voters and their cast ballots by means of the trackers.

Formally, let *Voters*, *Trackers*, and *Choice* be three finite sets such that $|\mathit{Voters}| = |\mathit{Trackers}|$. The first part of the protocol corresponds to a random choice of a secret *tracker bijection* $\mathcal{F}_T\colon \mathit{Voters} \to \mathit{Trackers}$ that assigns a unique tracker to each voter. We denote the set of all such bijections by $\mathcal{T}$. Moreover, the final part of Selene can be presented as a *public bulletin function* $\mathcal{F}_P\colon \mathit{Trackers} \to \mathit{Choice}$ that assigns to each $\mathit{tr} \in \mathit{Trackers}$ the vote $\mathcal{F}_P(\mathit{tr})$ cast by the owner of the tracker. The secret *choice function* $\mathcal{F}_I = \mathcal{F}_P \circ \mathcal{F}_T \colon \mathit{Voters} \to \mathit{Choice}$ connects voters with their ballots. The Election Authority is the only entity in the process that can observe the choice function. Note that this view can be applied to any voting system based on publicly visible trackers.

2.3 Combinatorial Aspects

Let $\mathcal{F}_P$, $\mathcal{F}_I$, and $\mathcal{F}_T$ be the public bulletin function, the choice function, and the tracker bijection. We will now estimate the range of uncertainty of the coercer, with the intuition that the less he knows about the real tracker assignment $\mathcal{F}_T$, the more room is available for coercion resistance. For each $tr, tr' \in \mathit{Trackers}$, let $tr \approx_{\mathcal{F}_P} tr'$ iff $\mathcal{F}_P(tr) = \mathcal{F}_P(tr')$, i.e., two trackers are "vote-equivalent" if they point to identical votes. Moreover, let $\mathit{Trackers}/\approx_{\mathcal{F}_P}$ denote the set of equivalence classes of $\approx_{\mathcal{F}_P}$. The uncertainty of the coercer can be measured by the number of permutations in the set of trackers, that he cannot distinguish from the actual tracker assignment.

Formally, let $\Pi(\mathit{Trackers})$ denote the set of all permutations of trackers, i.e., bijections $\pi \colon \mathit{Trackers} \to \mathit{Trackers}$. Now, $\mathcal{T}_{\mathcal{F}_P} = \{\pi \in \Pi(\mathit{Trackers}) \mid \mathcal{F}_P = \mathcal{F}_P \circ \pi\}$ is the set of all permutations of trackers that are consistent with the public outcome of the election. To see this, observe that for each $\pi \in \mathcal{T}_{\mathcal{F}_P}$ we have $\mathcal{F}_I = \mathcal{F}_P \circ \mathcal{F}_T = \mathcal{F}_P \circ \pi \circ \mathcal{F}_T$. The size of $\mathcal{T}_{\mathcal{F}_P}$ reflects the space of defensive capabilities against coercion, should a part of the secret tracker bijection become public, under the assumption that voting is one-shot rather than repeated.

Definition 1 (Anti-coercion space). *The* anti-coercion space *of an election, given the public bulletin function $\mathcal{F}_P$, is defined as $\mathit{acspace}(\mathcal{F}_P) = \{\mathcal{F}_P \circ \pi \mid \pi \in \Pi(\mathit{Trackers})\}$. Intuitively, $\mathit{acspace}(\mathcal{F}_P)$ corresponds to all the possible choice functions $\mathcal{F}_I$ consistent with $\mathcal{F}_P$.*

Theorem 1. *If the result of the election consists of n votes for candidates $\mathfrak{c}_1, \ldots, \mathfrak{c}_k$, s.t. each candidate $\mathfrak{c}_i$ got m_i votes, then $|\mathit{acspace}(\mathcal{F}_P)| = \frac{n!}{(m_1!)\cdot\ldots\cdot(m_k!)}$.*

Proof. Notice that $|\mathcal{T}_{\mathcal{F}_P}| = \prod_{\rho \in \mathit{Trackers}/\approx_{\mathcal{F}_P}} (|\rho|!)$. By the orbit-stabilizer theorem [14] we have $|\mathit{acspace}(\mathcal{F}_P)| = \frac{|\mathcal{T}|}{|\mathcal{T}_{\mathcal{F}_P}|}$, which concludes the proof.

Note that the space is typically vast, unless for very small elections or when almost all the voters voted for the same candidate. This is good news, as it makes it potentially hard for the coercer to obtain useful information about the real choices of the voters. On the other hand, a faithful representation of the coercer's state of knowledge leads to state-space explosion, which makes verification more complex. We will see it clearly in the next section.

3 Multi-Agent Model of Selene

In this section we present in detail a multi-agent model of Selene. We start with defining the formal structures used for modeling the entities participating in the protocol and their interactions, and move on to presenting the model of Selene, together with selected details of its implementation.

3.1 Models of Multi-Agent Interaction

Multi-agent systems are often modeled by a variant of transition systems where transitions are labeled with combinations of actions, one per agent. Moreover, epistemic relations are used to indicate states that look the same to a given agent. Formally, an *imperfect information concurrent game structure* or *iCGS* [1] is given by $M = \langle \mathbb{A}\mathrm{gt}, St, PV, V, Act, d, o, \{\sim_a \mid a \in \mathbb{A}\mathrm{gt}\}\rangle$ which includes a nonempty finite set of all agents $\mathbb{A}\mathrm{gt} = \{1, \ldots, k\}$, a nonempty set of states St, a set of atomic propositions PV and their valuation $V : PV \to 2^{St}$, and a nonempty finite set of (atomic) actions Act. The protocol function $d : \mathbb{A}\mathrm{gt} \times St \to 2^{Act}$ defines nonempty sets of actions available to agents at each state; we will write $d_a(q)$ instead of $d(a, q)$, and define $d_A(q) = \prod_{a \in A} d_a(q)$ for each $A \subseteq \mathbb{A}\mathrm{gt}, q \in St$. Furthermore, o is a (deterministic) transition function that assigns the outcome state $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to each state q and tuple of actions $\langle \alpha_1, \ldots, \alpha_k\rangle$ such that $\alpha_i \in d(i, q)$ for $i = 1, \ldots, k$.

Every $\sim_a \subseteq St \times St$ is an epistemic equivalence relation with the intended meaning that, whenever $q \sim_a q'$, the states q and q' are indistinguishable to agent a. The *iCGS* is assumed to be *uniform*, i.e., $q \sim_a q'$ implies $d_a(q) = d_a(q')$.

It should be mentioned that *iCGS* generalize transition networks as well as normal form games, repeated games, and extensive form games. Moreover, it is possible to define the notions of *strategic play* and *strategic ability* in *iCGS*.

3.2 A Multi-Agent Model of Selene

In what follows, we describe our multi-agent model of Selene. The system consists of the set *Voters* of voter agents, the single *Coercer*, the Election Defense System *ElectionDS*, and the *Environment* agent. We denote the set of all these agents by *Agents*. The local states of each agent are defined by its local variables. A global state of the system is a valuation of local variables of all the agents. Each agent can observe its local variables and selected local variables of the *Environment*. For simplicity, we assume a single coercer. This precludes the case when, e.g., two coercers request two different votes from the same voter and then compare the results. We plan to study this type of interactions in the future.

The model is parameterized by the following natural numbers: n voters; k possible choices (i.e., the ways that a ballot can be filled); *maxCoerced* voters that can be influenced by the coercer; *votingWaitTime* and *helpRequestTime* that reflect the maximal number of steps the system waits for votes and notifications about being coerced, respectively. We denote such model by $\mathcal{M}(n, k, \mathit{maxCoerced}, \mathit{votingWaitTime}, \mathit{helpRequestTime})$.

In what follows, we omit auxiliary variables and actions that are not relevant to understanding the interplay between agents.

Agent *Environment*. The purpose of the *Environment* agent is twofold. Firstly, it serves as a container for variables shared by selected agents. The agents can have read-only or write-only access to the variables (denoted by *Can observe* and *Can set*, respectively, in agent interfaces in Figures 1, 2, and 3). Secondly, it traces the passage of time and changes the stage of elections. Namely, the elections start

Variables

- *vote*: 0...k
- *demandedVote*: 0...k

Actions

- $Vote_i$, for $i \in \{1, \ldots, k\}$
- $INeedVote_i$, for $i \in \{1, \ldots, k\}$
- *FetchGoodTracker*
- *CopyRealTracker*
- *Wait*
- *Finish*

Can observe

- WBB: *public election function*
- *elections' stage (init/voting/defense)*
- *his real tracker (when permitted)*
- *his exposed tracker*

Can set

- *his exposed tracker (via CopyRealTracker)*

Figure 1: A Voter agent

in the **initial stage**, when the secret bijection is non-deterministically prepared. Then, the **voting phase** is open and the clock is started. This phase ends when either all the voters send their choices or time exceeds *votingWaitTime*. Then, the system enters the **defense stage** and the clock restarts. The defense stage ends either when the clock exceeds *helpRequestTime* or all the voters execute the *Finish* action. Note that every agent can observe WBB, i.e., public election function, the stage, and the clock value. The clock limits are also public knowledge.

***Voter* agents.** Each *Voter* shares the same structure, presented in Figure 1. It is able to record via the *vote* variable the vote cast for choice $i \in \{1, \ldots, k\}$ by executing the action $Vote_i$. This action can be used only once, in the voting phase. It also records the coercer's request to vote for *demandedVote*. In both the cases 0 denotes that the variable is not set, i.e. the agent did not vote yet and has not been contacted by the coercer, respectively. In addition to the public variables of *Environment*, each *Voter* can observe his real tracker, obtained in the defense phase by executing action *FetchGoodTracker*. The agent can also observe his exposed tracker, i.e., the number assigned by *ElectionDS*, as presented to the *Coercer* agent. This becomes possible after requesting in the defense phase a tracker that points to a specific choice $i \in \{1, \ldots, k\}$, by firing action $INeedVote_i$. After obtaining his real tracker a *Voter* can decide to make it visible to the coercer by executing action *CopyRealTracker*. Finally, the agent can always *Wait*, unless the clock reaches the limit set for a phase. In the latter case, if it is the voting

Variables

- $\mathit{falseTrackerSentToVoter}_i$: Boolean, for $i \in \{1, \ldots, n\}$

Actions

- $\mathit{SetFalseTrackerOfVoter}_i\mathit{To}_j$, for $1 \leq i \leq n$, $1 \leq j \leq k$
- *Wait*

Can observe

- WBB: *public election function*
- *the secret bijection*
- *elections' stage (init/voting/defense)*

Can set

- *the exposed tracker of every Voter*

Figure 2: The *ElectionDS* agent

phase, then the *Voter* needs to decide on the vote immediately, and if it is the defense phase, then it automatically ends his participation by firing action *Finish*. It should be noted that these actions are autonomous, e.g., a *Voter* can signal *ElectionDS* that he is coerced to vote in a selected way, even if coercion does not take place.

Agent *ElectionDS*. The structure of *ElectionDS* agent is presented in Figure 2. The agent can, in addition to the public variables of *Environment*, observe the secret bijection function. This gives *ElectionDS* the full knowledge of the secret election function. The boolean variables $\mathit{falseTrackerSentToVoter}_i$ record that a voter $i \in \{1, \ldots, n\}$ requested and has been provided with a false tracker. This request is fulfilled by executing an action $\mathit{SetFalseTrackerOfVoter}_i\mathit{To}_j$ that sets the exposed tracker of voter $1 \leq i \leq n$ to choice $1 \leq j \leq k$. Note that while *ElectionDS* can set the value of the exposed tracker of any *Voter*, it cannot read the current value of the variable. Therefore, each *Voter* can first request a false tracker pointing to any choice and expose his real tracker afterwards, unknowingly to *ElectionDS*. Finally, *ElectionDS* can always *Wait*.

Agent *Coercer*. The structure of *Coercer* is presented in Figure 3. Starting from the initial phase until the votes are published, the agent can demand from any voter $1 \leq j \leq n$ to vote for $1 \leq i \leq k$, by executing $\mathit{ReqVote}_i\mathit{FromVoter}_j$ action. Such a request can be made at most once per voter and the total number of requests cannot exceed *maxCoerced*. These choices are recorded using variables $\mathit{voteDemandedFromVoter}_i$, where $1 \leq i \leq n$. As previously, the value of 0 signifies that no request has been made. The agent can observe all public variables of *Environment* and all the exposed trackers of all voters. At any step, the *Coercer* agent can *Wait*.

Variables

- $voteDemandedFromVoter_i : 0, \ldots, k$, for $i \in \{1, \ldots, n\}$

Actions

- $ReqVote_i FromVoter_j$, for $1 \leq i \leq k$, $1 \leq j \leq n$
- *Wait*

Can observe

- WBB: *public election function*
- *elections' stage (init/voting/defense)*
- *the exposed tracker of every Voter*

Figure 3: The *Coercer* agent

Atomic propositions. In order to construct formulae that can be interpreted in the model, we need some atomic propositions. We set $PV = \{\mathsf{finished}\} \cup \{\mathsf{vote_{v,i}} \mid 1 \leq v \leq n, 1 \leq i \leq k\}$. Proposition finished denotes that the execution of the protocol has come to an end, and it holds iff all the voters have executed *Finish* or the clock has exceeded *helpRequestTime*. Formula $\mathsf{vote_{v,i}}$ says that voter v has voted for candidate i; it holds iff v's variable *vote* contains i.

3.3 Implementation of the Model

We have used two different model checkers to verify properties of the model presented in Section 3.2. For exact model checking, we used MCMAS [15], which is the only publicly available tool for $\mathbf{ATL}_{\mathrm{ir}}$. MCMAS is based on the Interpreted Systems Programming Language (ISPL), which allows for higher-level descriptions of agents and their interaction. In Figure 4, we present the ISPL code implementing the *Coercer* agent. The local variables of the agent are denoted by *Vars*, *Lobsvars* denotes the set of the environment variables that the agent can observe, and *Actions* are action labels. The *protocol* section specifies which actions are available at what states; the *evolution* section defines the consequences of their execution. We refer to [15] for more details about MCMAS and ISPL. Unfortunately, exact model checking of abilities under imperfect information works only for very small models. To overcome this, we used the approximate model checking technique from [10]. We have developed a prototype tool implementing the technique, in which the explicit state variant of the model from Section 3.2 is hard-coded. The explicit state representation is completely isomorphic with the ISPL code. Both the ISPL code generator and the prototype tool are available online at `https://github.com/SeleneMC16/SeleneModelChecker`.

```
Agent Coercer

Lobsvars = {exposedTrackerOfVoter1, exposedTrackerOfVoter2};

Vars:
    coercedVoters: 0..2;
    voteDemandedFromVoter1: 0..2;
    voteDemandedFromVoter2: 0..2;
end Vars

Actions = {ReqVote1FromVoter1, ReqVote2FromVoter1,
           ReqVote1FromVoter2, ReqVote2FromVoter2, Wait};

Protocol:
    coercedVoters < maxCoerced and voteDemandedFromVoter1 = 0
    and Environment.votesPublished = false:
    {ReqVote1FromVoter1, ReqVote2FromVoter1, Wait};

    coercedVoters < maxCoerced and voteDemandedFromVoter2 = 0
    and Environment.votesPublished = false:
    {ReqVote1FromVoter2, ReqVote2FromVoter2, Wait};

    Other: {Wait};
end Protocol

Evolution:
    coercedVoters = coercedVoters + 1 if
    (Action = ReqVote1FromVoter1
    or Action = ReqVote2FromVoter1
    or Action = ReqVote1FromVoter2
    or Action = ReqVote2FromVoter2);

    voteDemandedFromVoter1 = 1 if Action = ReqVote1FromVoter1;
    voteDemandedFromVoter1 = 2 if Action = ReqVote2FromVoter1;
    voteDemandedFromVoter2 = 1 if Action = ReqVote1FromVoter2;
    voteDemandedFromVoter2 = 2 if Action = ReqVote2FromVoter2;
end Evolution

end Agent
```

Figure 4: ISPL code of the *Coercer* agent

4 Specification of Properties

In this section, we provide a list of example coercion-related specifications, formulated in the strategic logic $\mathbf{ATL}_{\mathrm{ir}}$. We reduce vulnerability to coercion to the ability of the coercer to learn about the value of the voter's vote. We also assume that the voter prefers to evade coercion, rather than cooperate with the coercer. Intuitively, this reflects the typical voters' attitude towards intimidation, rather than vote buying and bribery. We begin by a brief introduction to the logic.

4.1 Alternating-time Temporal Logic

Alternating-time temporal logic with imperfect information and imperfect recall ($\mathbf{ATL}_{\mathrm{ir}}$ [1,20]) generalizes the branching-time temporal logic **CTL** by replacing the path quantifiers E, A with *strategic modalities* $\langle\langle A \rangle\rangle$. Informally, $\langle\langle A \rangle\rangle\gamma$

expresses that the coalition A has a collective strategy to enforce the temporal property γ. The formulae make use of temporal operators: "X" ("next"), "G" ("always from now on"), "F" ("now or sometime in the future"), and U ("until").

Syntax. The language of $\mathbf{ATL}_{\mathrm{ir}}$ formulae is defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle A\rangle\!\rangle \mathrm{X}\varphi \mid \langle\!\langle A\rangle\!\rangle \mathrm{G}\varphi \mid \langle\!\langle A\rangle\!\rangle \mathrm{F}\varphi \mid \langle\!\langle A\rangle\!\rangle \varphi \,\mathrm{U}\, \varphi,$$

where p stands for atomic propositions, and $A \subseteq \mathbb{A}\mathrm{gt}$ for any coalition of agents.

Strategies. A *strategy* of agent $a \in \mathbb{A}\mathrm{gt}$ is a conditional plan that specifies what a is going to do in every possible situation. Formally, it can be represented by a function $s_a : St \to Act$ satisfying $s_a(q) \in d_a(q)$ for each $q \in St$. Moreover, we require that $s_a(q) = s_a(q')$ whenever $q \sim_a q'$, i.e., strategies specify same choices in indistinguishable states. A *collective strategy* s_A for coalition $A \subseteq \mathbb{A}\mathrm{gt}$ is a tuple of individual strategies, one per agent from A. By $s_A[a]$ we denote the strategy of agent $a \in A$ selected from s_A.

Outcome paths. A *path* $\lambda = q_0q_1q_2\ldots$ is an infinite sequence of states such that there is a transition between each q_i, q_{i+1}. We use $\lambda[i]$ to denote the ith position on path λ (starting from $i = 0$). Function $out(q, s_A)$ returns the set of all paths that can result from the execution of strategy s_A from state q. Formally:

$out(q, s_A) = \{\lambda = q_0, q_1, q_2 \ldots \mid q_0 = q$ and for each $i = 0, 1, \ldots$ there exists $\langle \alpha^i_{a_1}, \ldots, \alpha^i_{a_k}\rangle$ such that $\alpha^i_a \in d_a(q_i)$ for every $a \in \mathbb{A}\mathrm{gt}$, and $\alpha^i_a = s_A[a](q_i)$ for every $a \in A$, and $q_{i+1} = o(q_i, \alpha^i_{a_1}, \ldots, \alpha^i_{a_k})\}$.

Function $out^{\mathrm{ir}}(q, s_A) = \bigcup_{a\in A} \bigcup_{q\sim_a q'} out(q', s_A)$ collects all the outcome paths that start from states indistinguishable from q to at least one agent in A.

Semantics. Let M be an *iCGS* and q its state. The semantics of **ATL** can be defined by the clauses below. We omit all the clauses for temporal operators except for "sometime", as they are not relevant for this paper.

- $M, q \models p$ iff $q \in V(p)$, and $M, q \models \neg\varphi$ iff $M, q \not\models \varphi$,
- $M, q \models \varphi \wedge \psi$ iff $M, q \models \varphi$ and $M, q \models \psi$, and $i \in \mathbb{N}$ we have $M, \lambda[i] \models \varphi$,
- $M, q \models \langle\!\langle A\rangle\!\rangle \mathrm{F}\varphi$ iff there exists a collective strategy s_A such that for all $\lambda \in out^{\mathrm{ir}}(q, s_A)$ there exists $i \in \mathbb{N}$ such that $M, \lambda[i] \models \varphi$.

In order to reason about the knowledge of agents, we add *modalities* K_a:

- $M, q \models \mathsf{K}_a\varphi$ iff $M, q' \models \varphi$ for all q such that $q \sim_a q'$.

That is, $\mathsf{K}_a\varphi$ says that φ holds in all the states that agent a considers possible at the current state of the world q.

4.2 Formulae for Coercion

Let us consider a coercer attempting to force a group of voters $A \subseteq \mathbb{A}\mathrm{gt}$ to vote for his preferred candidate. We can assume w.l.o.g. that the number of the candidate is 1. The formulae in Figure 5 express different "flavors" of the coercer's coercive ability, with the following reading:

$$\Phi_1 \equiv \langle\!\langle Coercer \rangle\!\rangle \mathrm{F} \big(\mathsf{finished} \wedge (\bigwedge_{v\in A} \neg\mathsf{vote}_{v,1} \rightarrow \mathsf{K}_{Coercer}(\bigvee_{v\in A} \neg\mathsf{vote}_{v,1})\big)$$

$$\Phi_2 \equiv \langle\!\langle Coercer \rangle\!\rangle \mathrm{F} \big(\mathsf{finished} \wedge (\bigvee_{v\in A} \neg\mathsf{vote}_{v,1} \rightarrow \bigvee_{v\in A} \mathsf{K}_{Coercer}(\neg\mathsf{vote}_{v,1})\big)$$

Figure 5: Formulae for model checking

configuration	#states	tgen	Lower approx.		Upper approx.		Approx.	Exact	
			tverif	result	tverif	result	result	(tg+tv)	result
$(2, 1, 1, 1, 1)$	427	<1	<1	False	<1	True	?	<1	True
$(2, 1, 1, 4, 4)$	16777	1	<1	False	<1	True	?	249	True
$(2, 1, 2, 4, 4)$	22365	1	<1	True	<1	True	True	timeout	
$(2, 2, 1, 4, 4)$	331441	19	1	False	16	True	?	timeout	
$(2, 2, 2, 4, 4)$	596577	36	2	True	31	True	True	timeout	
$(3, 2, 1, 1, 1)$	281968	40	<1	False	4	True	?	timeout	
$(4, 1, 1, 1, 1)$	146001	2	<1	False	2	True	?	timeout	
$(4, 2, 1, 1, 1)$	memout							timeout	

Figure 6: Experimental results for formula Φ_1

- Φ_1 expresses that the coercer can enforce a state where the elections are over and, if no one in A followed his orders, then the coercer knows that at least one of them disobeyed (but does not necessarily know who);
- Φ_2 says that if some of the voters did not vote as ordered, then the coercer will identify at least one of them.

Conceptually, the formulae capture the extent to which the coercer can identify the disobedience of the coerced voters, and hence knows when to execute his threats. Note that, when $A = \{v\}$, then both formulae are equivalent.

5 Verification

Selene is supposed to provide protection against coercion, so the formulae in Section 4 should in principle be all false. However, every protection mechanism has its limits. As noted in [19], even a single coercer can defeat the election defense system if the number of ballots is small or the coerced voter is particularly unlucky and the vote demanded by the coercer is not present in WBB. Also, the coercer's power intuitively increases with the number of voters he can simultaneously coerce. Finally, the exact limits of the coercer's ability to coerce become unclear when we consider more complex models, due to their combinatorial complexity. This is exactly when model checking can help to detect threats or verify correctness. In this section, we provide a preliminary attempt at model checking of the properties specified in Section 4.2 with respect to the models proposed in Section 3.2.

5.1 Exact and Approximate Model Checking of $\mathbf{ATL_{ir}}$

Synthesis and verification of strategies under partial observability is hard. More precisely, model checking of **ATL** variants with imperfect information has been proved $\mathbf{\Delta_2^P}$- to **PSPACE**-complete for agents that play memoryless strategies [9,20]. In our case, the following result applies.

Proposition 1 ([9,20]). *Model checking* $\mathbf{ATL}_{\mathrm{ir}}$ *is* $\mathbf{\Delta_2^P}$*-complete with respect to the number of the transitions in the model and length of the formula.*

The only publicly available tool for verification of imperfect information strategies is MCMAS [15], which essentially searches through the space of all the possible strategies. Nevertheless, no better algorithm is currently known, despite some recent attempts [6]. We employ MCMAS for *exact* model checking of our coercion specifications for Selene. An interesting alternative has been proposed recently in the form of *approximate model checking* based on fixpoint approximations of formulae [10]. The idea is to model check, instead of formula $\varphi \equiv \langle\langle A\rangle\rangle \mathrm{F}\mathsf{p}$, its upper and lower approximations $tr_U(\varphi)$ and $tr_L(\varphi)$:

- $tr_U(\varphi)$ verifies the existence of a *perfect information strategy* that achieves $\mathrm{F}\mathsf{p}$. Clearly, when there is no perfect information strategy to achieve it, then $\langle\langle A\rangle\rangle \mathrm{F}\mathsf{p}$ must also be false;
- $tr_L(\varphi)$ is a more sophisticated property, expressed in Alternating Epistemic μ-Calculus with Steadfast Next Step, with the property that the truth of $tr_L(\varphi)$ always implies φ. We refer the interested readers to [10] for details.

We have implemented the approximate algorithm from [10], together with a model generator for Selene, and ran a number of experiments with both exact and approximate model checking. The setup of the experiments, as well as the results, are presented in the rest of the section.

5.2 Experiments and Results

We collect the results of the evaluation for each of the specified formulae in tables presented in Figures 6 to 7. We show performance results for the approximation algorithms, both for the lower and the upper bound, and compare them to the exact verification done with MCMAS. Each row in a table corresponds to a single run of an experiment over the selected model. The columns contain the following information:

- the parameters of the model (*configuration*), consisting of the numbers of voters and available candidates, the maximal numbers of voters that the coercer can try to coerce and the clock steps that the system waits for incoming votes and for notifications from the voters about coercion attempts. E.g., configuration (2,2,2,4,4) describes the model with 2 voters, 2 candidates, the coercer coercing up to 2 voters, and the maximal time units for the system to wait for votes and coercion notifications being both set to 4;

configuration	#states	tgen	Lower approx.		Upper approx.		Approx.	Exact	
			tverif	result	tverif	result	result	(tg+tv)	result
$(2,1,1,1,1)$	427	<1	<1	False	<1	True	?	<1	True
$(2,1,1,4,4)$	16777	1	<1	False	1	True	?	257	True
$(2,1,2,4,4)$	22365	1	<1	True	<1	True	True	timeout	
$(2,2,1,4,4)$	331441	19	1	False	4	False	False	timeout	
$(2,2,2,4,4)$	596577	36	1	False	7	False	False	timeout	
$(3,2,1,1,1)$	281968	40	<1	False	1	False	False	timeout	
$(4,1,1,1,1)$	146001	2	<1	False	3	True	?	timeout	
$(4,2,1,1,1)$	memout							timeout	

Figure 7: Experimental results for formula Φ_2

– The size of the state space (*#states*) and the time that the algorithm spent on generating the data structures for the model (*tgen*);
– The running time and output of the verification algorithm (*tver, result*) for model checking the lower approximation $tr_L(\phi)$, and similarly for the upper approximation $tr_U(\phi)$;
– The result of the approximation (*Approx. result*), with "?" in case of inconclusive output;
– The total running time (*tg+tv*) and the result (*result*) of the exact $\mathbf{ATL}_{\mathrm{ir}}$ model checking with MCMAS.

The running times are given in seconds. *Timeout* indicates that the process did not terminate in 2 hours. *Memout* indicates that the process is terminated by the system due to allocating too much memory.

The exact $\mathbf{ATL}_{\mathrm{ir}}$ model checking is performed with MCMAS 1.3.0. To perform the approximate verification, we used the explicit representations of models from Section 3.2, and an implementation of the fixpoint algorithms from [10] in a stand-alone tool written in C++. The models used in both approaches were isomorphic. The tests were conducted on a Intel Core i7-6700 CPU with dynamic clock speed of 2.60 – 3.50 GHz, 32 GB RAM, running 64bit Ubuntu 16.04 Linux.

5.3 Discussion of the Results

As confirmed by the experiments, the question posed by formula Φ_2 is the most restrictive. Namely, in Φ_2 we ask whether the coercer has a general strategy to find out exactly which voter voted against his demands, assuming that there was a disobedient one. The answer to this question is true only in special cases of a single candidate. On the other hand, the results of verification of Φ_1 reveal that the system is sometimes not able to fully defend a coerced group and the coercer can detect that at least one of the members did not follow the demands. To illustrate this on a simple example, in a model of two voters and two ballots the coercer has a trivial strategy: request a vote for candidate 1 from both the voters. Moreover, in this case the coercer has even more knowledge, as he knows which one of the voters deceived (they both did).

$$\Phi_2^{dist_{A'}} \equiv \langle\langle Coercer \rangle\rangle \mathrm{F}\left(\mathsf{finished} \wedge ((\bigvee_{v\in A} \neg \mathsf{vote}_{v,1} \wedge \bigwedge_{v'\in Agents\setminus A} dist_{A'}(v')) \to \bigvee_{v\in A} \mathsf{K}_{Coercer}(\neg \mathsf{vote}_{v,1})\right)$$

Figure 8: A formula for quantitative analysis

Exact model checking with MCMAS seems infeasible in most of the cases, except for the small models up to hundreds of states. The approximations offer a dramatic speedup, enabling verification of models up to hundreds of thousands of states. Although the approximate method is faster, the results can be inconclusive; namely, we observe cases where the truth value of $tr_L(\varphi)$ differs from the value of $tr_U(\varphi)$. It should be noted that in the approximate approach the graphs are represented explicitely in memory, unlike in the case of BDD-based symbolic methods. Still, memory is cheaper and easier to buy than time.

5.4 Counting Coercion-Friendly Configurations

The validity of a property is a strong result: if a formula is true in the model, then the coercer has a strategy to achieve his goal under all possible circumstances. The system is therefore completely insecure against the considered type of attack. If, however, the formula turns out false, it does not mean that the system is always able to defend itself. In such case we only know that there is no uniform strategy that allows the coercer to break the system's defenses, given no information about the initial state of affairs (e.g., a partially uncovered choice function). We thus attempt to quantitatively estimate the extent to which our model is safe from the attacks expressed by Φ_2. To this end we inspect in detail all the possible distributions $D_{A'}$ of votes of voters outside of the coerced group A and check under which of these the coercer can precisely point to a disobedient voter. Formally, $dist_{A'} \in D_{A'}$ iff $dist_{A'}$ is a function from $Agents \setminus A$ to PV such that for each $v \in Agents \setminus A$ there exists $1 \leq i \leq k$ such that $dist_{A'}(v) = \mathsf{vote}_{\mathsf{v,i}}$.

To perform quantitative analysis we utilise the formula $\Phi_2^{dist_{A'}}$ presented in Figure 8. Note that the formula depends on $dist_{A'} \in D_{A'}$ and $A \subseteq Agents$. Intuitively, it expresses the ability of the coercer to enforce that if some of the agents in A did not vote for candidate 1 and the remaining voters voted according to $dist_{A'}$, then the coercer can identify a voter in A that did not vote for 1.

The process of quantitative analysis is performed as follows. For a given model configuration, we fix an arbitrary coalition A. Then, for each distribution $dist_{A'} \in D_{A'}$ the formula $\Phi_2^{dist_{A'}}$ is verified. Our approach is based on state-labelling, hence we can inspect all the states reached just after publishing votes. In Fig. 9 by $\#vote$ we denote the aggregate number of such states that are consistent with any $dist_{A'} \in D_{A'}$ and $\#rvote$ collects the count of how many of these states satisfy $\Phi_2^{dist_{A'}}$. As we can observe, there are cases where the coercer can gain advantage in nearly half of considered distributions.

configuration	#states	tgen	result	#vote	#rvote	pvote $= \frac{\#rvote}{\#vote} \times 100\%$
$(2,2,1,4,4)$	331441	14	False	200	50	25%
$(2,2,2,1,1)$	9651	<1	False	144	36	25%
$(2,2,2,4,4)$	596577	24	False	360	90	25%
$(2,3,1,2,2)$	289423	10	False	378	168	44%
$(2,3,2,1,1)$	64829	1	False	576	256	44%
$(2,3,2,2,2)$	661501	22	False	864	384	44%
$(2,3,2,2,2)$	281968	15	False	672	84	12%
$(2,3,2,2,2)$	765232	41	False	1824	228	12%
$(4,1,1,1,1)$	146001	2	False	240	0	0%

Figure 9: Experimental results for formula $\Phi_2^{dist_{A'}}$ with percentage coverage

It should be emphasized that approximate algorithms are used for model checking $\Phi_2^{dist_{A'}}$. Thus, the *pvote* shows only the percentage of *confirmed cases* where a successful coercion strategy exists. The actual counts may be larger, since the approximations provide only guaranteed lower bound estimation.

6 Conclusions

In this paper, we present our first step towards model checking of e-voting protocols with respect to strategic abilities of their participants. We propose a simple multi-agent model of Selene, together with several formulae of $\mathbf{ATL}_{\mathrm{ir}}$ expressing coercion, and conduct preliminary experiments with model checking.

Our construction of the model is based on a natural pattern of dividing the outcome of an election into the public bulletin and the secret choice function, together with a secret bijection. We argue that the MAS approach provides a flexible framework for modelling security properties of voting protocols. In particular, it offers a natural separation of social and technical components and their interactions. Moreover, $\mathbf{ATL}_{\mathrm{ir}}$ offers a sensible trade-off between expressivity and veracity as the property specification language.

Model checking is done in two variants: exact, using MCMAS [15], and approximate, using the recently proposed methodology of fixpoint approximations [10]. As the experiments show, despite the prohibitive complexity of model checking with $\mathbf{ATL}_{\mathrm{ir}}$, the approximate method enables the analysis of many instances of our models, even in the presence of combinatorial explosion.

In the future, we plan to apply some recent developments in model reduction methods for strategic logics and allow for verification of more complex models. These include techniques such as abstraction, bisimulation-based reduction, and partial-order reduction. Moreover, we would like to extend the model of Selene with additional actors, such as coercers with conflicting goals.

Acknowledgements. The authors acknowledge the support of the National Centre for Research and Development (NCBR), Poland, under the PolLux project VoteVerif (POLLUX-IV/1/2016).

References

1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time Temporal Logic. *Journal of the ACM*, 49:672–713, 2002.
2. F. Belardinelli, R. Condurache, C. Dima, W. Jamroga, and A.V. Jones. Bisimulations for verification of strategic abilities with application to ThreeBallot voting protocol. In *Proc. of AAMAS*, pages 1286–1295. IFAAMAS, 2017.
3. J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Proc. of the 26th ann. ACM symp. on Theory of Computing*, pages 544–553. ACM, 1994.
4. I. Boureanu, P. Kouvaros, and A. Lomuscio. Verifying security properties in unbounded multiagent systems. In *Proceedings of AAMAS*, pages 1209–1217, 2016.
5. N. Bulling and W. Jamroga. Alternating epistemic mu-calculus. In *Proceedings of IJCAI-11*, pages 109–114, 2011.
6. S. Busard, C. Pecheur, H. Qu, and F. Raimondi. Reasoning about memoryless strategies under partial observability and unconditional fairness constraints. *Information and Computation*, 242:128–156, 2015.
7. 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.
8. J. Dreier, P. Lafourcade, and Y. Lakhnech. A formal taxonomy of privacy in voting protocols. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6710–6715. IEEE, 2012.
9. W. Jamroga and J. Dix. Model checking ATL$_{ir}$ is indeed Δ_2^P-complete. In *Proceedings of EUMAS'06*, volume 223 of *CEUR Workshop Proc.* CEUR-WS.org, 2006.
10. W. Jamroga, M. Knapik, and D. Kurpiewski. Fixpoint approximation of strategic abilities under imperfect information. In *Proc. of AAMAS*, pages 1241–1249, 2017.
11. H. L. Jonker and W. Pieters. Receipt-freeness as a special case of anonymity in epistemic logic. *Proc. of the 19th Comp. Sec. Found. workshop*, pages 28–42, 2006.
12. R. Kusters and T. Truderung. An epistemic approach to coercion-resistance for electronic voting protocols. In *Security and Privacy*, pages 251–266, 2009.
13. R. Küsters, T. Truderung, and A. Vogt. A game-based definition of coercion-resistance and its applications. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 122–136. IEEE, 2010.
14. S. Lang. *Algebra*. Addison-Wesley, 1993.
15. A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: An open-source model checker for the verification of multi-agent systems. *Int. Journal on Software Tools for Technology Transfer*, 2015. Available online.
16. Alessio Lomuscio and Wojciech Penczek. LDYIS: a framework for model checking security protocols. *Fundamenta Informaticae*, 85(1-4):359–375, 2008.
17. Bo Meng. A critical review of receipt-freeness and coercion-resistance. *Information Technology Journal*, 8(7):934–964, 2009.
18. T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols*, pages 25–35. Springer, 1998.
19. P.Y.A. Ryan, P.B. Rønne, and V. Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *Proc. of Financial Cryptography and Data Security*, volume 9604 of *LNCS*, pages 176–192. Springer, 2016.
20. P. Y. Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82–93, 2004.
21. M. Tabatabaei, W. Jamroga, and P. Y. A. Ryan. Expressing receipt-freeness and coercion-resistance in logics of strategic ability: Preliminary attempt. In *Proc. of the PrAISe@ECAI Workshop*, pages 1:1–1:8. ACM, 2016.