Model checking agent knowledge in dynamic access control policies

Masoud Koleini, Eike Ritter and Mark Ryan

University of Birmingham,
Birmingham, B15 2TT, UK

Abstract. In this paper, we develop a modeling technique based on interpreted systems in order to verify temporal-epistemic properties over access control policies. This approach enables us to detect information flow vulnerabilities in dynamic policies by verifying the knowledge of the agents gained by both reading and reasoning about system information. To overcome the practical limitations of state explosion in model-checking temporal-epistemic properties, we introduce a novel abstraction and refinement technique for temporal-epistemic safety properties in ACTLK (ACTL with knowledge modality K) and a class of interesting properties that does fall in this category.

1 Introduction

Assume a conference paper review system in which all the PC members have access to the number of the papers assigned to each reviewer. Further assume that a PC member Alice can see the list of the papers that are assigned to another PC member and that are not authored by Alice. Then if Alice is the author of a submitted paper, she can find who the reviewer of her paper is by comparing the number of papers assigned to each reviewer (shown by the system) with the number of the assigned papers of that reviewer which she has access to.

The above is an example of a potential information leakage in *content management systems*, which are collaborative environments that allow users to create, store and manage data. They also allow controlling access to the data based on the user roles. In such multi-agent systems, access to the data is regulated by *dynamic access control policies*, which are a class of authorization rules that the permissions for an agent depend on the state of the system and change when agents interact with the system [1–3]. In complicated access control scenarios, there is always a risk that some required properties do not hold in the system. For instance and for a conference paper review system, the following properties need to hold in the policy:

- It should be impossible for the author of a paper to be assigned as the reviewer of his own paper (temporal safety property).
- There must be no way for the author of a paper to find out who is the reviewer of his paper (epistemic safety property).

Epistemic properties take *knowledge* of the agents into account. The knowledge can be gained by directly accessing the information, which complies with one of the

meanings of the knowledge in ordinary language, that means the agent *sees* the truth. But agent also knows the truth when he indirectly reasons about it [4].

Information flow as a result of reasoning is a critical vulnerability in many collaborative systems like conference paper review systems, social networks and document management systems, and is difficult to detect. The complication of access control policies in multi-agent collaborative frameworks makes finding such weaknesses more difficult using non-automated mechanisms. Moreover, the state of art dynamic access control verification tools are unable to find such properties as they do not handle epistemic property verification in general. Therefore as the *first contribution* of this paper, we propose a policy authorization language and express how to use the *interpreted systems framework* [5, 6] in order to model the related access control system. Using interpreted systems enables us to address misconfiguration in the policy and information disclosure to unauthorized agents by verifying temporal-epistemic properties expressed in the logic CTLK (CTL with knowledge modality K). The knowledge of an agent in our modelling covers both the knowledge gained by reasoning and by reading information when access permission is granted.

The practical limitation of interpreted systems is the state explosion for the systems of medium to large state space. There is also a limited number of research on the automated abstraction and refinement of the models defined in interpreted systems framework. As the *second contribution*, we develop an novel fully automated abstraction and refinement technique for verifying safety properties in ACTLK (which is a subset of CTLK) over an access control system modelled in the framework of interpreted systems. We extend counterexample guided abstraction refinement [7] to cover the counterexamples generated by the verification of temporal-epistemic properties and when the counterexample is tree-like [8]. In this paper, we only discuss the counterexamples with finite length paths, but this approach can be extended to the paths of infinite length using an unfolding mechanism [7]. We use a model-checker for multi-agent systems [9] and build the abstract model in its modelling language. The refinement is guided using the counterexample generated by the model-checker. The counterexample checking algorithm is provably sound and complete. We also introduce an interactive refinement for a class of epistemic properties that does not fall in ACTLK, but can specify interesting security properties.

We provide the details of the algorithms and proofs of the propositions in a technical report [10].

2 Related work

In the area of knowledge-based policy verification, Aucher et al. [11] define *privacy policies* in terms of permitted or forbidden knowledge. The dynamic part of their logic deals with sending or broadcasting data. Their approach is limited in modeling knowledge gained by the interaction of agents in a multi-agent system. RW framework [2] has the most similar approach with ours. The transition system in RW is build over the knowledge of the active coalition of agents. In each state, the knowledge of the coalition is the accumulation of the knowledge obtained by performing actions or sampling system variables in previous transitions together with the initial knowledge. In the other

words, knowledge in RW is gained by reading or altering system variables, not by reasoning about them. This is similar to PoliVer [12], which approximates knowledge by readability. Such verification tools are not able to detect information flow as a result of reasoning.

In the field of abstraction and refinement for temporal-epistemic logic, Cohen et al. [13] introduced the theory of simulation relation and existential abstraction for interpreted systems. Their approach is not automated and they have not provided how to refine the abstract model if the property does not hold and the counterexample is spurious. A recent research on abstraction and refinement for interpreted systems is done by Zhou et al. [14]. Although their work is about abstraction and refinement of interpreted systems, their paper is abstract and mainly discusses the technique to build up a tree-like counterexample when verifying ACTLK properties.

3 Interpreted systems

Fagin et al. [6] introduced interpreted systems as the framework to model multi-agent systems in games scenarios. They introduced a detailed transition system which contains agents, local states and actions. Such a framework enables reasoning about both temporal and epistemic properties of the system.

Definition 1 (Interpreted system). *Let Φ be a set of atomic propositions and $\Omega = \{e, 1, \ldots, n\}$ be a set of agents. An* interpreted system *I is a tuple:*

$$I = \langle (L_i)_{i\in\Omega}, (P_i)_{i\in\Omega}, (ACT_i)_{i\in\Omega}, S_0, \tau, \gamma \rangle$$

where (1) L_i is the set of local states of agent i, and the set of global states is defined as $S = L_e \times L_1 \times \cdots \times L_n$. We also use the notation of L_i as the function that accepts a set of global states and returns the corresponding set of local states for agent i. For each $s \in S$, $l_i(s)$ denotes the local state of agent i in s (2) ACT_i is the set of actions that agent i can perform, and $ACT = ACT_e \times ACT_1 \times \cdots \times ACT_n$ is the set of joint actions. We also use ACT_i as the function that accepts a joint action and returns the action of agent i (3) $S_0 \subseteq S$ is the set of initial states (4) $\gamma : S \times \Phi \to \{\top, \bot\}$ is called the interpretation function *(5) $P_i : L_i \to 2^{ACT_i} \setminus \{\emptyset\}$ is the protocol for agent i which defines the set of possible actions for agent i in a specific local state (6) $\tau : ACT \times S \to S$ is called the* partial transition function *with the property that if $\tau(\alpha, s)$ is defined, then for all $i \in \Omega : \ ACT_i(\alpha) \in P_i(l_i(s))$. We also write $s_1 \xrightarrow{\alpha} s_2$ if $\tau(\alpha, s_1) = s_2$.*

Definition 2 (Reachability). *A global state $s \in S$ is* reachable *in the interpreted system I if there exists $s_0 \in S_0$, $s_1, \ldots, s_n \in S$ and $\alpha_1, \ldots, \alpha_n \in ACT$ such that for all $1 \leq i \leq n : \ s_i = \tau(\alpha_i, s_{i-1})$ and $s = s_n$. In this paper, we use G to denote the set of reachable states.*

For an interpreted system I and each agent i we define an epistemic accessibility relation on the global states as follows:

Definition 3 (Epistemic accessibility relation). *Let I be an interpreted system and i be an agent. We define the* Epistemic accessibility relation for agent *i, written $\sim_i$, on the global states of I by $s \sim_i s' \quad$ iff $\quad l_i(s) = l_i(s')$ and s and s' are reachable.*

4 CTLK logic

We specify our properties in CTLK [15], which adds the epistemic modality K to the CTL (Computational Tree Logic). CTLK is defined as follows:

Definition 4. *Let Φ be a set of atomic propositions and Ω be a set of agents. If $p \in \Phi$ and $i \in \Omega$, then CTLK formulae are defined by:*

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid K_i\phi \mid EX\phi \mid EG\phi \mid E(\phi U \phi)$$

The symbol E is existential path quantifier and X, G and U are the standard CTL symbols. All CTLK temporal connectives including the pairs of symbols starting with universal path quantifier A can be written in terms of EX, EG and EU. For example, $AG\phi$ can be written as $\neg EF\neg\phi$. Epistemic connective K_i means "agent i knows that".

Example 1. Consider a conference paper review system. Then the safety property that says if a_2 is the reviewer of paper p_1 (the proposition $\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2)$ is true), then a_1 does not know the fact that a_2 is the reviewer of p_1 can be written as $AG(\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow \neg K_{\mathsf{a}_1}\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2))$. In an student information system where lecturers can assign one student as the demonstrator of another student, the property that states no two students, let us say a_2 and a_3, can be assigned as the demonstrator of each other is specified by the formula $AG(\neg(\mathsf{demOf}(\mathsf{a}_2, \mathsf{a}_3) \wedge \mathsf{demOf}(\mathsf{a}_3, \mathsf{a}_2)))$.

Definition 5 (Satisfaction relation). *For any CTLK-formula ϕ, the notation $(I, s) \models \phi$ means ϕ holds at state s in interpreted system I. The relation $\models$ is defined inductively in [16]. Given G as the set of reachable states in I, we have*

$$(I, s) \models K_i\phi \quad \Leftrightarrow \quad (I, s') \models \phi \text{ for all } s' \in G \text{ such that } s \sim_i s'$$

We use the notation $I \models \phi$ if for all $s_0 \in S_0 :\ (I, s_0) \models \phi$.

5 Policy syntax

Multi-agent access control systems grant or deny user access to the resources and services depending on the access rights defined in the policy. Access to the resources is divided into *write access*, which when granted, allows updating some system variables (in the context of this work, Boolean variables) and *read access*, that returns the value of some variables when granted. In this section, we present a policy syntax to define actions, permissions and evolutions. In the following section, we give semantics of the policy language by constructing an interpreted system from it.

Technical preliminaries: Let V be a finite set of variables and $Pred$ a finite set of predicates. The notation $\boldsymbol{v}$ is used to specify a sequence of distinct variables. An *atomic formula* or simply an *atom* is a predicate that is applied to a sequence of variables with the appropriate length. An access control policy is a finite set of rules defined as follows:

$$L ::= \top \mid \bot \mid w(\boldsymbol{v}) \mid L \vee L \mid L \wedge L \mid L \rightarrow L \mid \neg L \mid \forall v\ [L] \mid \exists v\ [L]$$
$$W ::= +w(\boldsymbol{v}) \mid\ -w(\boldsymbol{v}) \mid \forall v.\ W$$
$$W_s ::= W \mid W_s, W$$
$$A_R ::= \mathsf{id}(\boldsymbol{v}) : \{W_s\} \leftarrow L \qquad \text{Action rule}$$
$$R_R ::= \mathsf{id}(\boldsymbol{v}) : w(\boldsymbol{u}) \leftarrow L \qquad \text{Read permission rule}$$

In the above, $w \in Pred$, and $w(\boldsymbol{v})$ is an atom. L denotes a logical formula over atoms, which is the condition for performing an action or reading information. $\{W_s\}$ is the effect of the action that include the updates. $+w(\boldsymbol{v})$ in the effect means executing the action will set the value of $w(\boldsymbol{v})$ to true and $-w(\boldsymbol{v})$ means setting the value to false. In the case of $\forall v.W$ in the effect, the action updates the signed atom in W for all possible values of v. In the case that an atom appears with different signs in multiple quantifications in the effect (for instance, $w(c,d)$ in $\forall x.+w(c,x), \forall y.-w(y,d)$), then only the sign of the last quantification is considered for the atom. id indicates the identifier of the rule.

Let $a(\boldsymbol{v}) : E \leftarrow L$ be an action rule. The *free variables* of the logical formula L are denoted by $\mathbf{fv}(L)$ and are defined in the standard way. We also define $\mathbf{fv}(E) = \bigcup_{e \in E} \mathbf{fv}(e)$ where $\mathbf{fv}(\pm w(\boldsymbol{x})) = \boldsymbol{x}$ and $\mathbf{fv}(\forall x.W) = \mathbf{fv}(W) \backslash x$. We stipulate: $\mathbf{fv}(E) \cup \mathbf{fv}(L) \subseteq \boldsymbol{v}$. If $r(\boldsymbol{v}) : w(\boldsymbol{u}) \leftarrow L$ is a read rule, then $\mathbf{fv}(\boldsymbol{u}) \cup \mathbf{fv}(L) \subseteq \boldsymbol{v}$.

Let Σ be a finite set of objects. A *ground atom* is a variable-free atom; i.e. atoms with the variables substituted with the objects in Σ. For instance, if reviewer$\in Pred$ and Bob,Paper$\in \Sigma$, then reviewer(Bob,Paper) is a ground atom. In the context of this paper, we call the ground atoms as (atomic) *propositions*, since they only evaluate to true and false.

An *action* $\alpha : \varepsilon \leftarrow \ell$ contains an identifier α together with the *evolution rule* $\varepsilon \leftarrow \ell$, which is constructed by instantiating all the arguments in an action rule $a(\boldsymbol{v}) : E \leftarrow L$ with the objects in Σ. We refer to the whole action by its identifier α.

In an asynchronous multi-agent system, it is crucial to know the agent that performs an action. As the convention and for the rest of this paper, we consider the first argument of the action to be the agent performing that action. Therefore, in the action assignReviewer(Alice,Bob,Paper), Alice is the one that assigns Bob as the reviewer of Paper. If α is an action, then $\mathbf{Ag}(\alpha)$ denotes the agent that performs α.

A *read permission* $\rho : p \leftarrow \ell$ is constructed by substituting the arguments in read permission rule $r(\boldsymbol{v}) : w(\boldsymbol{u}) \leftarrow L$ with the objects in Σ. ρ is the identifier, p is the proposition and ℓ is the condition for reading p. As for the actions, we assume the first argument in ρ to be the agent that reads the proposition p, which is denoted by $\mathbf{Ag}(\rho)$.

Definition 6 (Policy). *An* access control policy *is a finite set of actions and read permissions derived by instantiating a set of rules with a finite set of objects.*

6 Building an interpreted system from a policy

In access control systems, when a read permission to a resource is granted, the resource will become a part of agent's local state which means agents knows the information. When the permission is denied, it will be removed from agent's directly accessible information. Therefore, we need to simulate this dynamic behaviour of the local states (temporary read permissions) by introducing extra variables into the model. This knowledge is called *knowledge by readability* of information. Moreover, it is a realistic approach to model access control systems in *asynchronous* manner. This is because in general and in real systems, different requests are held in a queue and processed one at a time asynchronously. An interpreted system is *asynchronous* if all joint actions contain at most one non-Λ agent action where Λ denotes no-operation.

Given a policy, we build an access control system based on interpreted systems framework by considering the requirements above. Incorporating temporary read permissions requires introducing some information into the local states. We say the proposition p is local to the agent i if its value only depends on the local state of i. In the other words, for all $s, s' \in S$ where $s \sim_i s'$ we have $\gamma(s,p) = \gamma(s',p)$.

Definition 7 (Local interpretation). *Let L_i be the set of local states of agent i in interpreted system I and Φ_i be the set of local propositions. We define the* local interpretation *for agent i as a function $\gamma_i : L_i \times \Phi_i \to \{\top, \bot\}$ such that $\gamma_i(l,p) = \gamma(s,p)$ where $l_i(s) = l$ for some global state s. We require the set of local propositions to be pairwise disjoint.*

The following lemma provides the theoretical background of modelling knowledge by readability in an interpreted system.

Lemma 1. *Let I be an interpreted system, G the set of reachable states, i an agent, Φ the set of propositions and $p \in \Phi$. Suppose that $p', p'' \in \Phi_i$. If for all $s \in G$:*

$$\textit{if } \gamma_i(l_i(s), p'') = \top \textit{ then } (I,s) \models p \;\Leftrightarrow\; \gamma_i(l_i(s), p') = \top$$

Then we have: $\quad \gamma_i(l_i(s), p'') = \top \quad \Rightarrow \quad (I,s) \models K_i p \vee K_i \neg p.$

We extend the interpreted systems to model knowledge by readability by incorporating all the atomic propositions that appear in the policy into the environment e. We call those propositions *policy propositions*. Now for each policy proposition p and for each agent, we introduce two local atomic propositions: p_{read} (p'' in Lemma 1) as the read permission of proposition p, and p_{loc} (p' in Lemma 1) as the local copy of p. We modify the transition function in order to satisfy the following property: for all reachable states, if p_{read} is true (agent has read access to p) in a state, then p_{loc} is assigned the same value as p. This property guarantees agent's knowledge of proposition p whenever his access to p is granted. The procedure to build the set of local propositions Φ_i and upgrading the set of actions in policy $\mathcal{C}$ into a new set $\mathcal{A}^u_{\mathcal{C}}$ which allows updating local propositions according to the Lemma 1 is presented in [10].

Symbolic transition function Given a policy which contains a set of actions, we provide the details for calculating the symbolic transition function we use for traversing over a path in our system. Symbolic transition function applies on a set of states and returns the result of performing an action over the states of that set.

As a convention, we use $s[p \mapsto m]$ where $s \in S$ to denote the state that is like s except that it maps the proposition p to the value m. Let $st \subseteq S$ be a set of states. When performing the action $\alpha : \varepsilon \leftarrow \ell$ in the states of st, the transition is only performed in the states that satisfy the permission ℓ. In the resulting states, the propositions that do not appear in ε remain the same as in the states that the transition begins. Therefore, we define:

$$\Theta_\alpha(st) = \Big\{ s[p \mapsto \top \mid +p \in \varepsilon][p \mapsto \bot \mid -p \in \varepsilon] \;\Big|\; s \in st, (I,s) \models \ell \Big\}$$

Definition 8 (Derived interpreted system). *Let $\mathcal{C}$ be a policy with Σ_{Ag} as the set of agents, $\Phi_{\mathcal{C}}$ the set of policy propositions, Φ_i, $i \in \Sigma_{Ag}$ and $\mathcal{A}^u_{\mathcal{C}}$ the local propositions and updated set of actions in $\mathcal{C}$ constructed to modify local propositions based on Lemma 1. Let $\Omega = \{e\} \cup \Sigma_{Ag}$ and $\Phi = \bigcup_{i\in\Omega} \Phi_i$ where $\Phi_e = \Phi_{\mathcal{C}}$. Then the interpreted system derived from policy $\mathcal{C}$ is:*

$$I_{\mathcal{C}} = \langle (L_i)_{i\in\Omega}, (P_i)_{i\in\Omega}, (ACT_i)_{i\in\Omega}, S_0, \tau, \gamma \rangle$$

where (1) L_i is the set of local states of agent i, where each local state is a valuation of the propositions in Φ_i. The set of global states is defined as $S = L_e \times L_1 \times \cdots \times L_n$ (2) $ACT_i = \{\alpha \in \mathcal{A}^u_{\mathcal{C}} \mid \boldsymbol{Ag}(\alpha) = i\} \cup \{\Lambda\}$ where Λ denotes no operation*, and a joint action is a $|\Omega|$-tuple such that at most one of the elements is non-Λ (asynchronous interpreted system). For simplicity,* we denote a joint action with its non-Λ element *(3) $S_0 \subseteq S$ is the set of initial states (4) γ is the interpretation function over S and Φ. If $p \in \Phi_i$ then we have $\gamma(s, p) = \gamma_i(l_i(s), p)$ (5) P_i is the protocol for agent i where for all $l \in L_i$: $P_i(l) = ACT_i$ (6) τ is the transition function that is defined as follows: if α is a joint action (or simply, an action) and $s \in S$, then $\tau(\alpha, s) = s'$ if $\Theta_\alpha(\{s\}) = \{s'\}$.*

The system derived from policy $\mathcal{C}$ is a special case of interpreted systems where the local states are the valuation of local propositions. In the derived model, the state of the system that is specified by policy $\mathcal{C}$ is simulated in environment e and local states store the information that are accessible to the agents.

7 Abstraction technique

In an interpreted system, the state space exponentially increases when extra propositions are added into the system. Considering a fragment of CTLK properties known as ACTLK as the specification language, we are able to verify the properties over an over-approximated abstract model instead of the concrete one. ACTLK is defined as follows:

Definition 9. *Let Φ be the set of atomic propositions and Ω set of agents. If $p \in \Phi$ and $i \in \Omega$, then ACTLK formulae are defined by:*

$$\phi ::= p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid K_i\phi \mid AX\phi \mid A(\phi U \phi) \mid A(\phi R \phi)$$

where the symbol A is universal path quantifier which means "for all the paths".

To provide a relation between the concrete model and the abstract one, we extend the *simulation relation* introduced in [17] to cover the epistemic relation between states. Using the abstraction technique that preserves simulation relation between the concrete model and the abstract one, we are able to verify ACTLK specification formulas over the model. In this paper and for abstraction and refinement, we focus on safety properties expressed in ACLK. The advantages of safety properties are first, they are capable of expressing *policy invariants*, and second, the generated counterexample contains finite sequence of actions (or transitions). We can extend the abstraction and refinement method to the full ACTLK by unfolding the loops in the counterexamples into finite transitions as described in [7], which is outside the scope of this paper.

7.1 Existential abstraction

The general framework of existential abstraction is first introduced by Clark et. al in [17]. Existential abstraction partitions the states of a model into clusters, or equivalence classes. The clusters form the states of the abstract model. The transitions between the clusters in the abstract model give rise to an over-approximation of the original (or concrete) model that *simulates* the original one. So, when a specification in ACTL (or in the context of this paper, ACTLK) logic is true in the over-approximated model, it will be true in the concrete one. Otherwise, a counterexample will be generated which needs to be verified over the concrete model.

Notation 1 *For simplicity, we use the same notation ($\sim_i$) for the epistemic accessibility relation in both the concrete and abstract interpreted systems.*

Definition 10 (Simulation). *Let I and $\widetilde{I}$ be two interpreted systems, Ω be the set of agents in both systems, and Φ and $\widetilde{\Phi}$ the corresponding set of propositions where $\widetilde{\Phi} \subseteq \Phi$. The relation $H \subseteq S \times \widetilde{S}$ is* simulation relation *between I and $\widetilde{I}$ if and only if:*

1. *For all $s_0 \in S_0$, there exists $\widetilde{s}_0 \in \widetilde{S_0}$ st. $(s_0, \widetilde{s}_0) \in H$.*

   and for all $(s, \widetilde{s}) \in H$:

2. *For all $p \in \widetilde{\Phi}$: $\gamma(s, p) = \widetilde{\gamma}(\widetilde{s}, p)$*
3. *For each state $s' \in S$ such that $\tau(s, \alpha) = s'$ for some $\alpha \in ACT$, there exists $\widetilde{s}' \in \widetilde{S}$ and $\widetilde{\alpha} \in \widetilde{ACT}$ such that $\widetilde{\tau}(\widetilde{s}, \widetilde{\alpha}) = \widetilde{s}'$ and $(s', \widetilde{s}') \in H$.*
4. *For each state $s' \in S$ such that $s \sim_i s'$, there exists $\widetilde{s}' \in \widetilde{S}$ such that $\widetilde{s} \sim_i \widetilde{s}'$ and $(s', \widetilde{s}') \in H$.*

The above definition for simulation relation over the interpreted systems is very similar to the one for Kripke model [7], except that the relation for the epistemic relation is introduced. If such simulation relation exists, we say that $\widetilde{I}$ *simulates* I (denoted by $I \preceq \widetilde{I}$). If H is a function, that is, for each $s \in S$ there is a unique $\widetilde{s} \in \widetilde{S}$ such that $(s, \widetilde{s}) \in H$, we write $h(s) = \widetilde{s}$ instead of $(s, \widetilde{s}) \in H$.

Proposition 1. *For every ACTLK formula φ over propositions $\widetilde{\Phi}$, if $I \preceq \widetilde{I}$ and $\widetilde{I} \models \varphi$, then $I \models \varphi$* [14].

Variable hiding abstraction Variable hiding is a popular technique in the category of existential abstraction. In our methodology, we consider factorizing the concrete state space into equivalence classes that act as abstract states by abstracting away a set of system propositions. In our approach, the states in each equivalence class are only different in the valuation of the hidden propositions. The actions in the abstract model are the equivalence classes of the actions in the concrete model. All the actions in each equivalence class have the visible propositions with the same sign in the effect of the evolution rule, and the semantically equivalent permissions when invisible propositions are existentially quantified. The abstract system simulates the concrete one (see [10] for technical details).

Definition 11. *We define $h_A : ACT \to \widetilde{ACT}$ as the surjection that maps the actions in the concrete model to the actions in the abstract one.*

8 Automated refinement

Our counterexample based abstraction refinement method consists of three steps: (1) *Generating the initial abstraction* by building the simplest possible initial abstract model by retaining only the propositions that appear in specification φ which we aim to verify (2) *Model-checking the abstract structure*. If the abstract model satisfies φ, then it can be concluded that the concrete model also satisfies φ. If the abstract model checking generates a counterexample, it should be checked if the counterexample is an actual counterexample for the concrete model. If it is spurious, the abstract model should be refined (3) *Refining the abstraction* by partitioning the states in the abstract model in such a way that the refined model does not admit the same counterexample. For the refinement, we turn some of invisible variables into visible. After each refinement, step 2 will be proceeded.

The process of abstraction and refinement will eventually terminate, as in the worst case, the refined model becomes the same as the finite state concrete one.

8.1 Generating the initial abstraction

For automatic abstraction refinement, we build the initial model as simple as possible. For an ACTLK formula φ, we keep all the atomic propositions that appear in φ visible in the abstract model and hide the rest.

8.2 Validation of counterexamples

The structure of a counterexample created by the verification of an ACTLK formula is different from the counterexample generated in the absence of knowledge modality. In an ACTLK counterexample, we have epistemic relations as well as temporal ones. Analysis of such counterexamples is more complicated than the counterexamples for temporal properties.

A counterexample for a safety property in ACTLK is a loop-free tree-like graph with states as vertices, and temporal and epistemic transitions as edges (figure 1). Every counterexample has an initial state as the root. A temporal transition in the graph is labelled with its corresponding action and epistemic transition is labelled with the corresponding epistemic relation. We define a *temporal path* as a path that contains only temporal transitions. An *epistemic path* contains at least one epistemic transition. Every state in the counterexample is *reachable from an initial state*, which may differ from the root. For any state s, we write s for the empty path which starts and finishes in s.

Counterexample formalism: A tree is a finite set of temporal and epistemic paths with an initial state as the root. Each path begins from the root and finishes at a leaf. For an epistemic transition over a path, we use the same notation as the epistemic relation while we consider the transition to be from left to the right. For instance, the tree in the figure 1 is formally presented by $\{\widetilde{s}_0 \xrightarrow{\widetilde{\alpha}_1} \widetilde{s}_1 \xrightarrow{\widetilde{\alpha}_2} \widetilde{s}_3,\ \widetilde{s}_0 \xrightarrow{\widetilde{\alpha}_1} \widetilde{s}_1 \sim_a \widetilde{s}'_2 \xrightarrow{\widetilde{\alpha}'_3} \widetilde{s}'_3\}$.

To verify a tree-like counterexample, we traverse the tree in a *depth-first* manner. An abstract counterexample is valid in the concrete model if a real counterexample in the concrete model corresponds to it. We use the notation $s \rightarrow s'$ when the type of the transition from s to s' is not known.

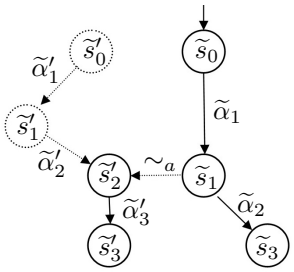

Fig. 1. A tree-like counterexample generated by the verification of an ACTLK safety property over the abstract model. In the diagram, $\widetilde{s}_0, \widetilde{s}'_0 \in S_0$ and $\widetilde{s}_1 \sim_a \widetilde{s}'_2$. As reachability is a requirement for $\widetilde{s}_1 \sim_a \widetilde{s}'_2$ and $\widetilde{s}_1$ is already reachable, the temporal path $\widetilde{s}'_0 \xrightarrow{\widetilde{\alpha}'_1} \widetilde{s}'_1 \xrightarrow{\widetilde{\alpha}'_2} \widetilde{s}'_2$ provides the witness for the reachability of $\widetilde{s}'_2$. Considering this witness is required in counterexample checking.

Definition 12 (Vertices, root). *Let $\widetilde{ce}$ be a counterexample. Then **Vert**$(\widetilde{ce})$ denotes the set of all the states that appear in $\widetilde{ce}$. **Root**$(\widetilde{ce})$ denotes the root of $\widetilde{ce}$. For a path $\widetilde{\pi}$, **Root**$(\widetilde{\pi})$ denotes the state that $\widetilde{\pi}$ starts with.*

Definition 13 (Corresponding paths). *Let $\widetilde{I}$ be an abstract model of the interpreted system I, h be the abstraction function, and h_A be the function that maps the actions in I to the ones in $\widetilde{I}$. The concrete path $\pi = s_1 \to \cdots \to s_n$ in the concrete model corresponds to the path $\widetilde{\pi} = \widetilde{s}_1 \to \cdots \to \widetilde{s}_n$ in the abstract model, if*

- *For all $1 \leq i \leq n : \widetilde{s}_i = h(s_i)$*
- *If $\widetilde{s}_i \xrightarrow{\widetilde{\alpha}_{i+1}} \widetilde{s}_{i+1}$ is a temporal transition, we have $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ where $h_A(\alpha_{i+1}) = \widetilde{\alpha}_{i+1}$.*
- *If $\widetilde{s}_i \sim_a \widetilde{s}_{i+1}$ is an epistemic transition, then $s_i \sim_a s_{i+1}$ and s_{i+1} is reachable in the concrete model.*

Definition 14 (Concrete counterexample). *Let $\widetilde{ce}$ be a tree-like counterexample in the abstract model where **Root**$(\widetilde{ce}) \in \widetilde{S}_0$. A concrete counterexample ce corresponds to $\widetilde{ce}$ if **Root**$(ce) \in S_0$ and there exists a one-to-one correspondence between the states and the paths of the counterexamples ce and $\widetilde{ce}$ according to the definition 13.*

To *verify a path* in a counterexample, we define two transition rules TEMPORALCHECK and EPISTEMICCHECK denoted by $\Rightarrow_t$ and $\Rightarrow_e$ as in figure 2. For a path with the transition $\widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}'$ as the head and for the concrete states st, the rule $\Rightarrow_t$ finds all the successors of the states in st which reside in $h^{-1}(\widetilde{s}')$. If the head of the path is the epistemic transition $\widetilde{s} \sim_a \widetilde{s}'$, then the rule $\Rightarrow_e$ extracts all the *reachable states* in $h^{-1}(\widetilde{s}')$ corresponding to π' as the witness of reachability of $\widetilde{s}'$, which has common local states with some states in $st \subseteq h^{-1}(\widetilde{s})$. Both the temporal and epistemic rules are deterministic. We write $\Rightarrow_t^*$ to denote a sequence of temporal transitions $\Rightarrow_t$. We use $\Rightarrow^*$ to denote a sequence of the transitions $\Rightarrow_t$ or $\Rightarrow_e$.

Proposition 2 (Soundness of $\Rightarrow^*$). *Let $\widetilde{\pi} = \widetilde{s}_1 \to \cdots \to \widetilde{s}_n$ be a path in the abstract model. If $st_1 \subseteq h^{-1}(\widetilde{s}_1)$ and $(\widetilde{\pi}, st_1) \Rightarrow^* (\widetilde{s}_n, st_n)$ for some $\emptyset \subset st_n \subseteq S$, then there exists a concrete path that starts from a state in st_1 and ends in a state in st_n.*

In the case that $\widetilde{\pi} = \widetilde{s}_0 \to \cdots \to \widetilde{s}_n$ is a path in the counterexample and $(\widetilde{\pi}, S_0 \cap h^{-1}(\widetilde{s}_0)) \Rightarrow^* (\widetilde{s}_n, st_n)$, then there exists a corresponding concrete path starting at some initial state $s_0 \in S_0 \cap h^{-1}(\widetilde{s}_0)$ which ends at some state $s_n \in st_n$.

$$\textsc{TemporalCheck}\ \frac{h_A^{-1}(\widetilde{\alpha}) = \{\alpha_1, \ldots, \alpha_n\}}{(\widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}' \,||\, \pi, st) \Rightarrow_t (\pi, \bigcup_{i=1}^{n} \Theta_{\alpha_i}(st) \cap h^{-1}(\widetilde{s}'))}$$

$$\textsc{EpistemicCheck}\ \frac{\begin{array}{c}\pi' = \widetilde{s}'_0 \xrightarrow{\widetilde{\alpha}'_1} \ldots \xrightarrow{\widetilde{\alpha}'_m} \widetilde{s}' \text{ is a temporal path to } \widetilde{s}' \text{ where } \widetilde{s}'_0 \in \widetilde{S}_0 \\ (\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow_t^* (\widetilde{s}', st') \qquad \hat{st} = \{s \in st' \mid l_a(s) \in L_a(st)\}\end{array}}{(\widetilde{s} \sim_a \widetilde{s}' \,||\, \pi, st) \Rightarrow_e (\pi, \hat{st})}$$

Fig. 2. Temporal and epistemic transition rules. In EPISTEMICCHECK rule, π' is the witness for the reachability of $\widetilde{s}'$ in the abstract model, and st' is the concrete states that are reachable through the concrete paths corresponding to π'. In the case that the model-checker returns all the abstract paths to $\widetilde{s}'$, let us say $\widetilde{\Pi}'$, then st' will be calculated as $st' = \bigcup\{st \mid \pi' = \widetilde{s}'_0 \rightarrow \cdots \rightarrow \widetilde{s}' \in \widetilde{\Pi}', \widetilde{s}'_0 \in \widetilde{S}_0 \text{ and } (\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow_t^* (\widetilde{s}', st)\}$.

Proposition 3 (Completeness of $\Rightarrow^*$). *Let $\widetilde{\pi} = \widetilde{s}_1 \rightarrow \cdots \rightarrow \widetilde{s}_n$ be a path in the abstract model. If there exists a concrete path $\pi = s_1 \rightarrow \cdots \rightarrow s_n$ corresponding to $\widetilde{\pi}$ and $s_1 \in st_1 \subseteq h^{-1}(\widetilde{s}_1)$, then $(\widetilde{\pi}, st_1) \Rightarrow^* (\widetilde{s}_n, st_n)$ for some $\emptyset \subset st_n \subseteq S$.*

Forward transition rules in figure 2 are sufficient to check *linear counterexamples* or equivalently, paths. To extend the counterexample checking to tree-like counterexample, extra procedures are required.

To verify a tree-like counterexample, we introduce two transition rules BACKWARDTCHECK and BACKWARDECHECK denoted by $\Leftarrow_t$ and $\Leftarrow_e$. The transition rules find all the predecessors of the states in st (figure 3) with respect to the temporal or epistemic transitions in a backward manner which reside in the set of reachable states through the path. We write $\Leftarrow^*$ to denote a sequence of backward transitions $\Leftarrow_t$ and $\Leftarrow_e$.

Assume that $\widetilde{\pi} = \widetilde{s}_0 \rightarrow \cdots \rightarrow \widetilde{s}_n$ is a path in the counterexample $\widetilde{ce}$ which $(\widetilde{\pi}, S_0 \cap h^{-1}(\widetilde{s}_0)) \Rightarrow^* (\widetilde{s}_n, st_n)$ for some $\emptyset \subset st_n \subseteq S$. st_n contains all the states in the leaves of the concrete paths corresponding to $\widetilde{\pi}$. The point is that not all the concrete states that are traversed in $\Rightarrow^*$ can reach the states in st_n. If $\widetilde{s} \in \mathbf{Vert}(\widetilde{\pi})$, then $(\widetilde{\pi}, st_n) \Leftarrow^* (\widetilde{s}_0, st_0)$ finds the set of states $r_{\widetilde{s}}$ which contains the reachable states in $h^{-1}(\widetilde{s})$ that lead to some states in st_n along the concrete paths corresponding to $\widetilde{\pi}$. st_0 contains the initial states that lead to the states in st_n. We use the notation $r_{\widetilde{s}}^{\widetilde{\pi}}$ to relate $r_{\widetilde{s}}$ with the path $\widetilde{\pi}$. Note that to find $r_{\widetilde{s}}^{\widetilde{\pi}}$, we first need to find st_n through $\Rightarrow^*$ transition.

Assume that $\widetilde{\Pi} \subseteq \widetilde{ce}$. If $\widetilde{s} \in \mathbf{Vert}(\widetilde{ce})$ then we define $r_{\widetilde{s}}^{\widetilde{\Pi}} = \cap_{\widetilde{\pi} \in \widetilde{\Pi}} r_{\widetilde{s}}^{\widetilde{\pi}}$. If $\widetilde{s} \notin \mathbf{Vert}(\widetilde{\pi})$, then we stipulate $r_{\widetilde{s}}^{\widetilde{\pi}} = h^{-1}(\widetilde{s})$. We also stipulate $r_{\widetilde{s}_0}^{\emptyset} = S_0 \cap h^{-1}(\widetilde{s}_0)$ where $\widetilde{s}_0 = \mathbf{Root}(\widetilde{ce})$ and $r_{\widetilde{s}}^{\emptyset} = h^{-1}(\widetilde{s})$ for all $\widetilde{s} \in \mathbf{Vert}(\widetilde{ce})$ where $\widetilde{s} \neq \widetilde{s}_0$.

Proposition 4. *A counterexample $\widetilde{ce}$ in the abstract model has a corresponding concrete one if:*

1. *for each path $\widetilde{\pi} \in \widetilde{ce}$, there exists $\emptyset \subset st \subseteq S$ such that $(\widetilde{\pi}, S_0 \cap h^{-1}(\widetilde{s}_0)) \Rightarrow^* (\widetilde{s}', st)$ where $\widetilde{s}_0 = \mathbf{Root}(\widetilde{ce})$ and $\widetilde{\pi}$ ends in $\widetilde{s}'$.*
2. *for all $\widetilde{s} \in \mathbf{Vert}(\widetilde{ce})$: $r_{\widetilde{s}}^{\widetilde{ce}} \neq \emptyset$.*

$$\textsc{BackwardTCheck}\ \frac{\begin{array}{c}(\pi, S_0 \cap h^{-1}(\mathbf{Root}(\pi))) \Rightarrow^* (\widetilde{s}, st') \\ h_A^{-1}(\widetilde{\alpha}) = \{\alpha_1, \ldots, \alpha_n\} \qquad rs = \bigcup_{i=1}^{n} \Theta_{\alpha_i}^{-1}(st) \cap st'\end{array}}{(\pi \,||\, \widetilde{s} \xrightarrow{\widetilde{\alpha}} \widetilde{s}', st) \Leftarrow_t (\pi, rs) \qquad r_{\widetilde{s}} := rs}$$

$$\textsc{BackwardECheck}\ \frac{\begin{array}{c}(\pi, S_0 \cap h^{-1}(\mathbf{Root}(\pi))) \Rightarrow^* (\widetilde{s}, st'') \\ \pi' = \widetilde{s}'_0 \xrightarrow{\widetilde{\alpha}'_1} \ldots \xrightarrow{\widetilde{\alpha}'_m} \widetilde{s}' \text{ is the temporal path to } \widetilde{s}' \text{ where } \widetilde{s}'_0 \in \widetilde{S}_0 \\ (\pi', S_0 \cap h^{-1}(\widetilde{s}'_0)) \Rightarrow^* (\widetilde{s}', st') \\ \hat{st} = \{s \in st'' \mid l_a(s) \in L_a(st \cap st')\}\end{array}}{(\pi \,||\, \widetilde{s} \sim_a \widetilde{s}', st) \Leftarrow_e (\pi, \hat{st}) \qquad r_{\widetilde{s}} := \hat{st}}$$

Fig. 3. Backward temporal and epistemic transition traversal. $\Theta_{\alpha}^{-1}(st)$ computes the set of predecessors of the states in st with respect to the transitions made by action α.

Let $\widetilde{ce}$ be a counterexample. The process of counterexample checking iterates over the paths in $\widetilde{ce}$ and checks if they corresponds to some paths in the concrete model by using proposition 2 and the transition rule $\Rightarrow^*$. If $\widetilde{\pi} \in \widetilde{ce}$ corresponds to some concrete paths, then for each state $\widetilde{s}$ in $\widetilde{\pi}$, the algorithm finds all the concrete states $r_{\widetilde{s}}^{\widetilde{\pi}}$ in $h^{-1}(\widetilde{s})$ that lead to the leaf states of the concrete paths, by applying $\Leftarrow^*$ over $\widetilde{\pi}$. In each loop iteration, the paths in $\widetilde{ce}$ that are processed in previous iterations are stored in the set $\widetilde{\Pi}$. The set $r_{\widetilde{s}}^{\widetilde{\Pi}}$ stores the concrete states that are common between the paths in $\widetilde{\Pi}$ and should remain non-empty during the process of counterexample checking. The procedure returns false if one of the paths in $\widetilde{ce}$ does not have corresponding concrete path or $r_{\widetilde{s}}^{\widetilde{\Pi}} = \emptyset$ for some $\widetilde{s} \in$ **Vert**$(\widetilde{ce})$ and $\widetilde{\Pi} \subseteq \widetilde{ce}$. Otherwise it returns true [10].

8.3 Refinement of the abstraction

For the refinement, we find the *failure state* in the counterexample as a standard terminology [7] by simulating the counterexample in the concrete model. A failure state $\widetilde{s}_f$ is a state along the tree-like counterexample where the concrete transitions cannot follow the transitions from $\widetilde{s}_f$. The concrete states that follow the counterexample and then stop following are *dead-end* states. To refine the abstract model, we split up the dead-end states from the rest of the states in $h^{-1}(\widetilde{s}_f)$ by turning a set of invisible variables into visible so that the same counterexample does not occur in the refined model by finding *conflict clauses*. See [10] for the full technical details.

8.4 Going beyond ACTLK

While this section develops a fully automated abstraction refinement method for the verification of temporal-epistemic properties that reside the category of ACTLK, some important epistemic safety properties does not fall into this category. For instance and in a conference paper review system, it is valuable for policy designers to verify that for all reachable states, an author of a paper, say a, cannot find out ($\neg K_{\mathsf{a}}$) who is the reviewer

of his own paper (see the first property in example 1). Although we are able to verify such properties in the concrete model, we cannot apply automated counterexample-guided abstraction and refinement for such properties.

For the abstraction and refinement, we restrict the formula in scope of the knowledge operators to propositional formulas (see [10] for the technical discussion). Then we use an interactive refinement procedure in the following way: we abstract the interpreted system in the standard way that we described. If the property does not hold in the abstract model, the counterexample will be checked in the concrete model and the abstract model will be refined if it is required. If the property turned to be true in the abstract model as a result of the satisfaction of $\neg K_{\mathsf{a}}$ (for which there is no witness in the abstract model), then we refine the local state of the agent a in an interactive manner. In this way, the tool asks the user to selects a set of invisible *local propositions*, with possibly higher correlation to the agent's knowledge, to be added in the next round if required. This process will continue until a valid counterexample appear while the local state is still abstract, or the local state becomes concretized.

9 Experimental results

We have implemented a tool in F# functional programming language. The front end is a parser that accepts a set of action and read permission rules, a set of objects and a query in the form of $\iota : \varphi$ where ι is the formula representing the initial states and φ is the property we aim to verify. Given the above information, the tool derives an interpreted system based on definition 8 where the initial states of the system are determined by parameter ι in the query. On the back end, we use MCMAS [9] as the model-checking engine. In the presence of abstraction and refinement, the tool feeds MCMAS with the abstract model together with the property φ. If model-checker returns true for an ACTLK property, then the tool returns true to the user. Otherwise, the tool automatically checks the generated counterexample based on proposition 4, and reports if it is a real counterexample, which will be returned to the user, or verification needs a refinement round. The tool performs an automated refinement if it is required. For the properties that are discussed in section 8.4, the tool asks user to select a set of invisible local variables to be added to the abstract model for the refinement when model-checker returns true. This will continue until all the related invisible local variables turn to visible, or a valid counterexample is found.

For this section, we choose one temporal and three epistemic properties for the case study of conference paper review system (CRS) with the information leakage vulnerability described in the introduction. We first verify the query (Query 1) "$\mathsf{author}(\mathsf{p}_1, \mathsf{a}_1) \wedge \neg\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_1) : AG(\neg\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_1))$" which states that if in the initial states, agent a_1 is the author of paper p_1 and not the reviewer of his own paper, then it is not possible for a_1 to be assigned as the reviewer of his paper p_1. Query 2 "$\neg\mathsf{submittedreview}(\mathsf{p}_1, \mathsf{a}_1) \wedge \mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) : AG(K_{\mathsf{a}_1}\mathsf{review}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow AG(\neg\mathsf{submittedreview}(\mathsf{p}_1, \mathsf{a}_1))$" checks if in the initial states, a_2 is the reviewer of paper p_1 and a_1 has not submitted a review for p_1, then a_1 cannot submit a review for p_1 later if he *reads* the review of a_2 (knowledge by readability). Query 3 "$\mathsf{author}(\mathsf{p}_1, \mathsf{a}_1) : AG(\mathsf{AllPapersAssigned} \wedge \mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow \neg K_{\mathsf{a}_1}\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2))$" asks if a_1 is the author of p_1, then it is not

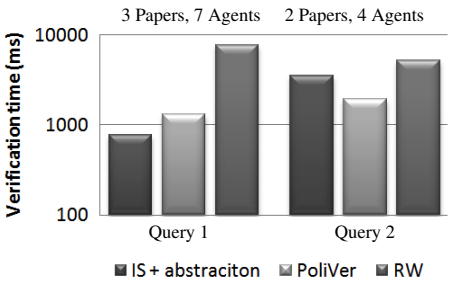

Fig. 4. Comparison of the verification time for the queries 1 and 2 between our tool which uses MCMAS as the model-checking engine, PoliVer and RW.

	Concrete model		Abstraction and refinement		
	time(s)	BDD vars	time(s)	Max BDD vars	last ref time
Query 3	6576.5	180	148.3	80	3.28
Query 4	6546.4	180	174.1	98	21

Fig. 5. A comparison of query verification time (in second) and runtime memory usage (in MB) between the concrete model and automated abstraction refinement method.

possible for a_1 to find the reviewer of his paper when his paper is assigned to a_2, which is not ACTLK. Query 4 "$\mathsf{author}(\mathsf{p}_1, \mathsf{a}_1) : AG(\mathsf{AllPapersAssigned} \wedge \mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2) \rightarrow K_{\mathsf{a}_1}\mathsf{reviewer}(\mathsf{p}_1, \mathsf{a}_2))$" has ACTLK property, which checks if a_1 can always find who the reviewer of his paper is when all the papers are assigned.

Queries 1 and 2 can be verified in access control policy verification tools like RW and PoliVer, which model knowledge by readability. We compare our tool in the presence of abstraction and refinement with RW and PoliVer from the point of verification time in figure 4. It is important to note that when applying abstraction and refinement, a high percentage of evaluation time is spent on generating abstract models, invoking executable MCMAS which also invokes Cygwin library, and verifying the counterexamples. In most of our experiments, verification of the final abstract model by MCMAS takes less than 10ms.

The novel outcome of our research is the verification of the queries 3 (interactive refinement) and 4 (fully automated refinement) where PoliVer and RW are unable to detect information leakage in CRS policy. In PoliVer and RW, the author never finds a chance to see who the reviewer of his paper is and therefore safety property holds in the system. Modelling in interpreted systems reveals that the author can reason who is the reviewer of his paper. For query 3, the tool also outputs the counterexample which demonstrates the sequence of actions that allows the author to reason about the reviewer of his paper. Figure 5 shows the practical importance of our abstraction method.

10 Conclusion

In this research, we introduced a framework for verifying temporal and epistemic properties over access control policies. In order to verify knowledge by reasoning, we used interpreted systems as the basic framework, and to make the verification practical for medium to large systems, we extended counterexample-guided refinement known as CEGAR to cover safety properties in ACTLK. Case studies and experimental results

show a considerable reduction in time and space when abstraction and refinement are in use. We also apply an interactive refinement for some useful properties that does not reside in ACTLK like the ones that contain the negation of knowledge modality.

Acknowledgement: We would like to acknowledge Microsoft Research and EPSRC project TS/I002529/1 "Trust Domains" for funding this research.

References

1. Becker, M.Y.: Specification and analysis of dynamic authorisation policies. In: Proceedings of 22nd IEEE Computer Security Foundations Symposium (CSF). (2009)
2. Zhang, N., Ryan, M., Guelev, D.P.: Synthesising verified access control systems through model checking. Journal of Computer Security **16**(1) (2008) 1–61
3. Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Specifying and reasoning about dynamic access-control policies. In: LNCS. Volume 4130., Springer (2006) 632–646
4. Mardare, R., Priami, C.: Dynamic epistemic spatial logics. Technical report, The Microsoft Research-University of Trento Centre for Computational and Systems Biology (2006)
5. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (1995)
6. Fagin, R., Halpern, J.Y., Moses, Y., Vardis, M.Y.: Knowledge-based programs. Distributed Computing **10**(4) (1997) 199–225
7. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Computer Aided Verification. (2000) 154–169
8. Clarke, E.M., Lu, Y., Com, B., Veith, H., Jha, S.: Tree-like counterexamples in model checking. In: LICS 2002: Proceedings of the 17 th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society (2002)
9. Lomuscio, A., Raimondi, F.: MCMAS: A model checker for multi-agent systems. In: TACAS 2006: Tools and Algorithms for the Construction and Analysis of Systems, Springer-Verlag (2006) 450–454
10. Koleini, M., Ritter, E., Ryan, M.: Reasoning about knowledge in dynamic policies. Technical report, University of Birmingham, School of Computer Science, Available at: http://www.cs.bham.ac.uk/ mdr/research/papers/pdf/13-mc-knowledge.pdf (2012)
11. Aucher, G., Boella, G., van der Torre, L.: Privacy policies with modal logic: The dynamic turn. In: Deontic Logic in Computer Science. (2010) 196–213
12. Koleini, M., Ryan, M.: A knowledge-based verification method for dynamic access control policies. In: ICFEM 2011: Proceedings of 13th International Conference on Formal Engineering Methods. (2011)
13. Cohen, M., Dam, M., Lomuscio, A., Russo, F.: Abstraction in model checking multi-agent systems. In: AAMAS 2009: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems. (2009) 945–952
14. Zhou, C., Sun, B., Liu, Z.: Abstraction for model checking multi-agent systems. Frontiers of Computer Science in China **5** (2011) 14–25
15. Lomuscio, A., Raimondi, F.: The complexity of model checking concurrent programs against CTLK specifications. In: AAMAS 2006: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, ACM Press (2006) 548–550
16. Cohen, M., Dam, M., Lomuscio, A., Qu, H.: A symmetry reduction technique for model checking temporal-epistemic logic. In: Proceedings of the 21st international jont conference on Artifical intelligence. IJCAI'09 (2009)
17. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5) (1994) 1512–1542