*Fundamenta Informaticae 79 (2007) 1–14*
*IOS Press*


# Verification of the TESLA protocol in MCMAS-X

**Alessio Lomuscio**
*Department of Computing, Imperial College London*
*180 Queen's Gate, London SW7 2AZ, UK*
*A.Lomuscio@imperial.ac.uk*

**Franco Raimondi***
*Department of Computer Science, University College London*
*Gower Street, London WC1E 6BT, UK*
*f.raimondi@cs.ucl.ac.uk*

**Bożena Woźna**†‡
*Institute of Mathematics and Computer Science, Jan Długosz University*
*Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland*
*b.wozna@ajd.czest.pl*

**Abstract.** We use MCMAS-X to verify authentication properties in the TESLA secure stream protocol. MCMAS-Xis an extension to explicit and deductive knowledge of the OBDD-based model checker MCMAS a verification tool for multi-agent systems.

## 1. Introduction

Model checking has traditionally been used for the verification of reactive systems whose properties are specified in one of the many variants of temporal logic. But autonomous and open systems, such as multi-agent systems [27] are best described and reasoned about by richer formalisms whose study is often pursued in frameworks studied in Artificial Intelligence (AI). One of the richer logics used in AI for this task is epistemic logic, or logic for knowledge [8], often combined with temporal logic [18, 10, 19, 16]. Epistemic logic has been shown useful in the modelling of a variety of scenarios from robotics, communication, etc., all sharing the need to represent formally the knowledge of the agents. Also of great interest is the use of temporal-epistemic formalisms to represent and analyse

*This research was carried out while Franco Raimondi was supported by EPSRC via grant CNA 04/04.
†This research was carried out while Bozena Wozna was supported by EPSRC grant GR/S49353.
‡Address for correspondence: Institute of Mathematics and Computer Science, Jan Długosz University, Al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland

formally security protocols. While the original BAN logic [6] lacked computational grounding, more recent attempts [11, 17] provide a full trace-based semantics to interpret the epistemic modalities as well as standard temporal modalities. Key to these approaches is the use of not only a modality for implicit knowledge, representing the knowledge that can be ascribed to a principal from an external point of view, but also one for explicit knowledge [8, 23, 15], representing knowledge the agent has immediate access to (for instance facts present in the agent's internal database).

While model checkers for standard temporal (implicit) knowledge have recently been made available [9, 21, 13], they currently do not support explicit knowledge and derivable notions and so their applicability to an "epistemically-oriented" verification of authentication protocols has not been pursued yet[1].

The aim of this research note is twofold: first we present a model checker that supports modalities for explicit and deductive knowledge; second we report on the use of these techniques to validate the correctness of TESLA [22], a protocol for secure real-time streaming.

The work presented here builds upon our earlier analysis of TESLA[17] and our engineering of MCMAS [13], a symbolic model checker for multi-agent systems. The rest of the paper is organised as follows. In Section 2 we present syntax and semantics of the logic formalism used throughout the paper. In Section 3 we briefly present MCMAS-X. In Section 4 we introduce the TESLA protocol and in Section 5 we model check some of its key properties. We conclude in Section 6 by discussing experimental results.

## 2. A Temporal Epistemic Logic

We shortly present the syntax and semantics of TDL [17], a multi-modal temporal epistemic logic with security-specialised primitives; we assume familiarity with the intuitive meaning of basic cryptographic primitives like *keys*, *nonces*, *pseudo-random functions*, and *MAC functions*. This section summarises material in [17].

**Syntax.** We begin with the definition of *messages*, which constitute a base for the security-specialised part of TDL.

Assume the following disjoint sets: a set $\mathbb{K} = \{k_1, k_2, \ldots\}$ of symmetric and asymmetric keys, a set $\mathbb{N}$ of nonces, a set $\mathbb{T} = \{t_1, t_2, \ldots\}$ of plain-texts, and a set $\mathbb{F}$ of commitments to keys defined by $\{f(k) \mid k \in \mathbb{K}$ where $f : \mathbb{K} \rightarrow \{0, 1, \ldots\}$ is a pseudo-random function$\}$; the commitment to a key $k$ is an integer value that is computed by applying a pseudo-random function $f$ to key $k$. It is assumed that $f^{-1}$ cannot be computed from $f$, so the key $k$ cannot be computed from the commitment to $k$. The set of *messages* $\mathbb{M}$ is defined by the following grammar:

$$m := t \mid k \mid n \mid f(k) \mid m \cdot m \mid \{m\}_k \mid \mathtt{MAC}(k, m)$$

where $t \in \mathbb{T}$, $k \in \mathbb{K}$, $n \in \mathbb{N}$, $f(k) \in \mathbb{F}$, $m$ is a generic message, and $\mathtt{MAC} : \mathbb{K} \times \mathbb{M} \rightarrow \{0, 1, \ldots\}$ is a *message authentication code* function. Again, we assume that the inverse of MAC cannot be computed (so the key $k$ cannot be inferred from the MAC value).

We write $m \cdot m'$ for the concatenation of $m$ and $m'$, $\{m\}_k$ for the encryption of $m$ with the key $k$, and $\mathtt{MAC}(k, m)$ for the message authentication code of $m$ and $k$. We assume that the set $\mathbb{K}$ is closed under

[1] Anonymity protocols (such as the dining cryptographers) can successfully be analysed by using implicit knowledge only [9, 26, 12].

inverses, i.e., for a given key $k \in \mathbb{K}$ there is an inverse key $k^{-1} \in \mathbb{K}$ such that $\{\{m\}_k\}_{k^{-1}} = m$. If the cryptosystem uses symmetric keys, then $k = k^{-1}$; in a public cryptosystem $k$ and $k^{-1}$ are different. We also define a *submessage* binary relation $\sqsubseteq$ on $\mathbb{M}$ as the smallest reflexive and transitive relation satisfying the following conditions: (1) $m \sqsubseteq m \cdot m'$, (2) $m \sqsubseteq m' \cdot m$, (3) $m \sqsubseteq \{m\}_k$.

Let $\mathcal{PV}$ be a set of propositional variables, $\mathcal{AG}$ a finite set of agents, $p \in \mathcal{PV}$, $i \in \mathcal{AG}$, and $m \in \mathbb{M}$. The set $\mathcal{WF}(\mathrm{TDL})$ of well-formed TDL formulae is defined by the following grammar:

$$\varphi := p \mid has_i(m) \mid sent_i(m) \mid received_i(m) \mid faked_i(m) \mid dropped_i(m) \mid \\ \neg\varphi \mid \varphi \vee \varphi \mid \mathrm{E}X\varphi \mid \mathrm{E}(\varphi \mathcal{U} \varphi) \mid \mathrm{A}(\varphi \mathcal{U} \varphi) \mid \mathcal{K}_i\varphi \mid \mathcal{X}_i\varphi \mid \mathcal{A}_i\varphi$$

The terms $has_i(m)$, $sent_i(m)$, $received_i(m)$, $dropped_i(m)$, and $faked_i(m)$ are security-specialised propositional variables, which, as one would expect, are read as "*agent $i$ has message $m$*", "*agent $i$ sent message $m$*", "*agent $i$ received message $m$*", "*agent $i$ dropped message $m$*, and "*agent $i$ faked message $m$*", respectively. The proposition $has_i(m)$ means that agent $i$ is in explicit possession of message $m$. The meaning of "explicit possession" depends on the application, the capabilities of the principals, and the protocol the principals are running. The interpretation of temporal and epistemic operators is standard. We use the shortcut $\mathcal{D}_i\alpha$ to represent $\mathrm{E}(\mathcal{K}_i\alpha \mathcal{U} \mathcal{X}_i\alpha)$. The formula $\mathcal{D}_i\alpha$ is read as "*agent $i$ may deduce $\alpha$ (by some computational process)*". For more details we refer to [17, 15]. The other temporal modalities $\mathrm{A}G, \mathrm{E}G, \mathrm{A}F, \mathrm{E}F$ can be defined as abbreviations of the ones above as standard.

**Interpreted Systems.** In this section we briefly summarise the multi-agent framework [8], over which a semantics for TDL will be given. In particular, we will focus on a specific class of multi-agent systems appropriate to modelling security protocols. These are message-passing systems in which one or more of the agents is an adversary controlling the communication channel.

A multi-agent system (MAS) consists of an environment and $n$ agents, each of which is in some particular local state at a given point in time. We assume that an agent's local state encapsulates all the information the agent has access to, and the local states of the environment describe information that is relevant to the system but that is not included in any local agent's state; the environment can be viewed as just another agent, as we will do here.

Given the security setting, we assume that the local state of an agent is a sequence of events of the form $(e_0, \ldots, e_m)$, where $e_0$ is the initial event, and for $i \in \{1, \ldots, m\}$, $e_i$ is a term of the form $sent(i, m)$ or $recv(m)$, where $m$ is a message and $i$ is an agent. The term $sent(i, m)$ stands for the agent has sent message $m$ to agent $i$. Similarly the term $recv(m)$ represents that the agent has received message $m$. Note that in $recv(m)$ the sender is not specified. This is because the receiver will not in general be able to determine the sender of a message he has received.

A multi-agent system is not a static entity. Its computations are usually defined by means of runs [8]. Thus, in these settings, an *interpreted system* for a multi-agent system is defined as a set of all possible runs together with a valuation function for the propositional variables of the language under consideration. We interpret TDL on an extension of interpreted systems augmented to include awareness sets; for more details we refer to [8, 17].

**Definition 2.1. (Interpreted system)**
Let $\mathcal{AG}$ be a finite set of $n$ agents, and let each agent $i \in \mathcal{AG}$ be associated with a set of local states $L_i$, and the environment be associated with a set of local states $L_e$. An *interpreted system* is a tuple $M = (S, T, \sim_1, \ldots, \sim_n, \mathcal{V}, \mathbb{A}_1, \ldots, \mathbb{A}_n)$ such that $S \subseteq \prod_{i=1}^{n} L_i \times L_e$ is a set of global states, $T \subseteq S \times S$ is a serial (temporal) relation on $S$, for each agent $i \in \mathcal{AG}$, $\sim_i \subseteq S \times S$ is an equivalence (epistemic)

relation defined by: $s \sim_i s'$ iff $l_i(s') = l_i(s)$, where $l_i : S \to L_i$ is a function that returns the local state of agent $i$ from a global state, $\mathcal{V} : S \to 2^{\mathcal{PV}}$ is a valuation function, and $\mathbb{A}_i : L_i \to 2^{\mathcal{WF}(\mathrm{TDL})}$ is an awareness function assigning a set of formulae to each state, for each $i \in \mathcal{AG}$.

Awareness sets represent facts (expressed as TDL formulae) an agent is aware of at a given state; we refer to [8, 17] for more details.

**Satisfaction.** A *path* in $M$ is an infinite sequence $\pi = (s_0, s_1, \ldots)$ of global states such that $(s_i, s_{i+1}) \in T$ for each $i \in \mathbb{N}$. For a path $\pi = (s_0, s_1, \ldots)$, we take $\pi(k) = s_k$. By $\Pi(s)$ we denote the set of all the paths starting at $s \in S$.

**Definition 2.2. (Satisfaction)**
Let $M$ be an interpreted system, $s$ a state, and $\alpha$, $\beta$ TDL formulae. The satisfaction relation $\models$, indicating truth of a formula in M at state $s$, is defined inductively as follows:

$(M, s) \models p$ if $p \in \mathcal{V}(s)$,
$(M, s) \models \neg\alpha$ if $(M, s) \not\models \alpha$,
$(M, s) \models \alpha \vee \beta$ if $(M, s) \models \alpha$ or $(M, s) \models \beta$,
$(M, s) \models \mathrm{E}X\alpha$ if $(\exists \pi \in \Pi(s))(M, \pi(1)) \models \alpha$,
$(M, s) \models \mathrm{E}(\alpha \mathcal{U} \beta)$ if $(\exists \pi \in \Pi(s))(\exists m \geq 0)[(M, \pi(m)) \models \beta$ and $(\forall j < m)(M, \pi(j)) \models \alpha]$,
$(M, s) \models \mathrm{A}(\alpha \mathcal{U} \beta)$ if $(\forall \pi \in \Pi(s))(\exists m \geq 0)[(M, \pi(m)) \models \beta$ and $(\forall j < m)(M, \pi(j)) \models \alpha]$,
$(M, s) \models \mathcal{X}_i\alpha$ if $(M, s) \models \mathcal{K}_i\alpha$ and $(M, s) \models \mathcal{A}_i(\alpha)$,
$(M, s) \models \mathcal{A}_i\alpha$ if $\alpha \in \mathbb{A}_i(l_i(s))$,
$(M, s) \models \mathcal{K}_i\alpha$ if $(\forall s' \in S)$ $(s \sim_i s'$ implies $(M, s') \models \alpha)$.

Note that since $\mathcal{D}_i\alpha$ is a shortcut for $\mathrm{E}(\mathcal{K}_i\alpha \mathcal{U} \mathcal{X}_i\alpha)$, as defined on page 3, we have that $(M, s) \models \mathcal{D}_i\alpha$ iff $(M, s) \models \mathrm{E}(\mathcal{K}_i\alpha \mathcal{U} \mathcal{X}_i\alpha)$.

Henceforth, we will only consider models with a fixed interpretation for the security-specialised propositional variables $sent_i(m)$ and $received_i(m)$; in particular, we take $\models$ to be defined for these propositions as follows:

$(M, s) \models sent_i(m)$ if $(\exists m' \in \mathbb{M})(\exists j \in \mathcal{AG})$ such that $m \sqsubseteq m'$ and $sent(j, m') \in l_i(s)$,
$(M, s) \models received_i(m)$ if $recv(m) \in l_i(s)$.

We leave the definitions of the other security-specialised propositions open; their interpretation will depend on the protocol under consideration. They are not needed for the analysis of TESLA presented below.

Let $M$ be an interpreted system. We say that a TDL formula $\varphi$ is *valid on* $M$ or $M$ *is a model for* $\varphi$ (written $M \models \varphi$), if $M, s \models \varphi$ for all states $s \in S$.

# 3. The model checkers MCMAS and MCMAS-X

## 3.1. Overview of MCMAS

MCMAS is a symbolic model checker developed for the automatic verification of multi-agent systems. In particular, MCMAS permits the verification of specifications involving time, knowledge, correct behaviour, and strategies of agents. MCMAS employs the formalism of interpreted systems and its extension [8, 26] as the underlying semantics for all these operators.

```
Agent SampleAgent
   Lstate = {s0,s1};  -- the local states
   Lgreen = {s0,s1};  -- the "correct" local states
   Action = {a1,a2};  -- actions
 Protocol:
   s0: {a1};          -- the actions permitted in a local state
   s1: {a1, a2};
 end Protocol
 Ev:
   -- the evolution function lists (on the right) the conditions
   -- causing a transition to the local state on the left.
   s1 if (Lstate=s0 and Action=a1 and AnotherAgent.Action=a7);
   s0 if (Lstate=s1 and Action=a1);
 end Ev
end Agent
```

Figure 1. An agent's definition using ISPL.

MCMAS reduces the problem of model checking a formula on a model to the problem of comparing two Ordered Binary Decision Diagrams (OBDDs, see [5] for more details) representing appropriate Boolean formulae. The idea behind OBDD-based model checking is to represent the set of states $[\![\varphi]\!]$ satisfying $\varphi$ as a Boolean formula built recursively on the structure of $\varphi$. The Boolean formula representing $[\![\varphi]\!]$ is encoded as an OBDD and this is compared to the OBDD representing the Boolean formula encoding the set of reachable states of $M$. If these are equal, then it is the case that $\varphi$ holds in $M$. OBDDs are used because they offer an efficient and compact representation for most Boolean formulae. MCMAS supports not only temporal logic but also epistemic logic [8], ATL [1], and modalities for correctness/violation [14].

An input to MCMAS is a program written in ISPL (Interpreted Systems Programming Language) representing all possible evolution of the system under analysis. ISPL is an SMV-like programming language for the description of interpreted systems. An ISPL program contains a list of agents, each of which is declared by reserved keywords:

```
Agent <AgentID>   <AgBody>  end Agent
```

where `<AgentID>` is any string uniquely identifying an agent, and `<AgBody>` contains the declarations of the local states, the actions, the protocols, and the evolution function for the agent. Following the agents' declaration, an ISPL file includes sections to declare the set of initial states, the evaluation function, and the set of formulae to be verified. Figure 1 reports the definition of a simple agent; we refer to the documentation available [25] for more details about the ISPL language.

MCMAS is available under the terms of the GNU General Public License (GPL) and it has been compiled on a number of platforms. MCMAS is run either from the command line, a graphical interface or a web interface, and it accepts various input parameters to inspect and fine-tune its performance.

### 3.2. MCMAS-X: an extensions of MCMAS

MCMAS-X extends MCMAS to support the verification of the operators $\mathcal{X}_i$, $\mathcal{A}_i$, and $\mathcal{D}_i$ (see Section 2).The

```
Agent SampleAgent
 Lstate = {s0,s1,s2,s3};
 Lgreen = {s0,s1,s2};
 Action = {a1,a2,a3};
 Protocol:
   [...]
 end Protocol
 Ev:
   [...]
 end Ev
 -- This is the new additional section for Awareness
 Aware:
   s0 : {p1,p2};  -- SampleAgent is aware of p1 and p2 in s0
   s1 : {p2};     -- ... and of p2 in s1
 end Aware
end Agent
```

Figure 2. An agent's definition using ISPL in MCMAS-X.

verification of the additional operators is performed by applying the same methodologies used for epistemic operators. Specifically, given an interpreted system $M$, let $[\![\varphi]\!]$ denote the set of global states of $M$ is which $\varphi$ holds. By the definition of satisfiability given in Section 2, we have:

$$[\![\mathcal{A}_i(\varphi)]\!] = \{s \in S | \varphi \in \mathbb{A}_i(l_i(s))\}.$$

Using standard procedures (e.g., see [7, 26]) the definition of $[\![\mathcal{A}_i(\varphi)]\!]$ can be re-casted in terms of OBDDs, and this definition can be inserted in the recursive procedure presented in [26] to compute the set $[\![\varphi]\!]$ for any TDL formula $\varphi$. The sets of states $[\![\mathcal{X}_i(\varphi)]\!]$ and $[\![\mathcal{D}_i(\varphi)]\!]$ can similarly be expressed using OBDDs.

We have implemented software procedures to perform the computation of these sets automatically in a tool called MCMAS-X (Model Checking eXplicit knowledge), available for download [24].

MCMAS-X extends MCMAS's syntax in two ways: first, it supports the verification of all the formulae introduced in Section 2; second, it augments the description of an agent with the definition of the function $\mathbb{A}_i$. This latter step is achieved by introducing the new keywords

```
Aware: <definitions> end Aware
```

as exemplified in Figure 2. In this example, the agent `SampleAgent` is aware of propositions `p1` and `p2` in local state `s0`, and of proposition `p2` in local state `s1` (notice that, following the definitions of Section 2 no consistency checks are made when defining awareness sets).

## 4. The TESLA protocol

In this section we introduce the *timed efficient stream loss-tolerant authentication* (TESLA) protocol [22]. TESLA provides secure authentication of the source of each packet in multicast or broadcast data streams. Five schemes of the protocol exist; each assumes a single sender (**S**) broadcasting a continuous

stream of packets to receivers (**R**) acting independently of one another; below we will describe the first variant of the TESLA protocol, and we will take into consideration one receiver only.

In order to provide security, in TESLA it is assumed that: (1) the sender and the receiver must be loosely time-synchronised; this can be done via a simple two-message exchange using, for example, the NTP protocol [20]; (2) the protocol must be bootstrapped through a regular data authentication system; this can be done using any secure session initiation protocol; (3) the protocol uses cryptographic primitives including MAC values and pseudo-random functions (PRFs); MAC is computed by a *message authentication code* function that takes as input a message and a secret key, whereas PRF provides *commitments* to keys. It is assumed that **S** and **R** know the PRF as well as the message authentication code function to be used in the session.

Following [2, 4], we now outline a TESLA scheme assuming that the protocol uses one pseudo-random function only, the participants are initially synchronised, **R** knows the disclosure schedule of the keys, and **S** sends packets at regular intervals that are agreed with **R** during the synchronisation process. More details are in [22].

Assuming that **S** has a digital signature key pair, with private key $k_{\mathbf{S}}^{-1}$ and public key $k_{\mathbf{S}}$ known to **R**, and that **R** chooses a random and unpredictable nonce, the initial $n$ steps, for $n > 1$, of the protocol for one sender and one receiver are the following:

**( -1)** $\mathbf{R} \rightarrow \mathbf{S} : n_{\mathbf{R}}$

**(0)** $\mathbf{S} \rightarrow \mathbf{R} : \{f(k_1), n_{\mathbf{R}}\}_{k_{\mathbf{S}}^{-1}}$

**(1)** $\mathbf{S} \rightarrow \mathbf{R} : P_1 \cdot \mathtt{MAC}(k_1, P_1)$, for $P_1 = t_1 \cdot f(k_2)$

**(2)** $\mathbf{S} \rightarrow \mathbf{R} : P_2 \cdot \mathtt{MAC}(k_2, P_2)$, for $P_2 = t_2 \cdot f(k_3) \cdot k_1$

...

**(n)** $\mathbf{S} \rightarrow \mathbf{R} : P_n \cdot \mathtt{MAC}(k_n, P_n)$, for $P_n = t_n \cdot f(k_{n+1}) \cdot k_{n-1}$

As one can see from the above, with the exception of the two initial packets, which are used to bootstrap the broadcasting process, and the third packet that contains only the message $t_1$ to be delivered, a *commitment* $f(k_2)$ to the key to be used to encode the MAC of the next packet, and the $\mathtt{MAC}(k_1, P_1)$ of the first packet, each packet contains: (1) the message $t_i$ to be delivered; (2) a *commitment* $f(k_{i+1})$ to the key to be used to encode the MAC of the next packets; (3) the key $k_{i-1}$ that was used to encode the MAC of the previous sent packet; (4) the $\mathtt{MAC}(k_i, P_i)$ of the current packet.

TESLA guarantees, among others, the following security property: "*the receiver does not accept any message unless it was actually sent by the sender*". We verify this and other properties by means of MCMAS-X in the next section. We have checked other variants of the TESLA protocol in a similar fashion but, given the procedure is similar, we only report here on the one above.

## 5. The TESLA protocol and MCMAS-X

In the section we model check the TESLA protocol by means of MCMAS-X. To do this we define and encode an interpreted system $M = (S, T, \sim_{\mathbf{S}}, \sim_{\mathbf{R}}, \sim_{\mathbf{I}}, \mathcal{V}, \mathbb{A}_{\mathbf{S}}, \mathbb{A}_{\mathbf{R}}, \mathbb{A}_{\mathbf{I}})$ representing TESLA's executions. Given our state space needs be finite we set a limit $n$ to the number of packets that can be broadcast during one session; obviously this assumption does not affect the analysis as no attack depends on the number of broadcasted packets.

As defined in Section 4, the TESLA protocol involves two participants: a sender (**S**) and a receiver (**R**), communicating through an unreliable channel that is under complete control of an intruder (**I**). In the interpreted system framework it is convenient to see the principals as agents, and the intruder as the environment. While specifying the agents (i.e., defining a set of local states, a set of actions, a protocol, and an evolution function), we assume that **S** has all the information he needs to compose a packet, i.e., he has a complete set of messages $M_{\mathbf{S}} \subseteq \mathbb{M}$. We also assume that $M_{\mathbf{S}}$ constitutes **S**'s initial database that remains accessible to him throughout the run. Moreover, we assume that **I** has all the information needed to compose well-formed packets, with $M_{\mathbf{I}} \subseteq \mathbb{M}$ such that $M_{\mathbf{I}} \cap M_{\mathbf{S}} = \emptyset$, and we assume that $M_{\mathbf{I}}$ can grow during the run. We work with a Dolev-Yao intruder in control of the channel and able to encrypt and decrypt messages if he has the appropriate key. We assume the intruder sends (resends and fakes) well-formed packets only, i.e., any packet contains a message body, a key commitment, a key, and an appropriate `MAC` value. Finally, we assume that **S**, **R**, and **I** use a shared PRF and a shared MAC function, **R** and **I** know the public key of **S**, **S** and **I** begin with disjoint sets of keys, and that **R** knows the precise schedule of packets, and that this information is incorporated into the first packet $P_0$, which cannot be dropped or faked.

We introduce the following sets of local states for **S**, **R** and **I**, respectively:

$$L_{\mathbf{S}} = \{[\cdot], [recv(n_{\mathbf{R}})], [sent(\mathbf{R}, P_0)]\} \cup \{[sent(\mathbf{R}, P_{i-1}), sent(\mathbf{R}, P_i)] \mid 0 < i \leq n\} \cup \{[sent(\mathbf{R}, P_{i-1}), sent(\mathbf{R}, P_i), sent(\mathbf{R}, P_{i+1})] \mid 0 < i \leq n\}.$$

$$\begin{aligned} L_{\mathbf{R}} = {} & \{[\cdot], [sent(\mathbf{S}, n_{\mathbf{R}})], [stop], [recv(P_0)]\} \cup \{[recv(P_0), recv(P_2)]\} \cup \\ & \{[recv(P_i), recv(P_{i+1})] \mid 0 \leq i \leq n\} \cup \{[recv(P_0), recv(P'_1), recv(P_2)]\} \cup \\ & \{[recv(P_{i-1}), recv(P_i), recv(P_{i+1})] \mid 0 < i \leq n\} \cup \\ & \{[recv(P_{i-1}), recv(P_i), recv(P_{i+2})] \mid 0 < i \leq n\} \cup \\ & \{[recv(P_i), recv(P_{i+1}), recv(P'_{i+2})] \mid 0 \leq i \leq n\} \cup \\ & \{[recv(P_0), recv(P'_1)]\} \cup \{[recv(P_0), recv(P'_1), recv(P'_2)]\}. \end{aligned}$$

$$\begin{aligned} L_{\mathbf{I}} = {} & \{[\cdot], [recv(n_{\mathbf{R}})], [recv(P_0)]\} \cup \{[recv(P_i), recv(P_{i+1})] \mid 0 \leq i \leq n\} \cup \\ & \{[recv(P_{i-1}), recv(P_i), recv(P_{i+1})] \mid 0 < i \leq n\} \cup \\ & \{[recv(P_0), recv(P_1), send(\mathbf{R}, P'_1)]\} \cup \\ & \{[recv(P_0), recv(P_1), send(\mathbf{R}, P'_1), recv(P_2)]\} \cup \\ & \{[recv(P_0), recv(P_1), send(\mathbf{R}, P'_1), recv(P_2), send(\mathbf{R}, P'_2)]\} \cup \\ & \{[recv(P_{i-1}), recv(P_i), recv(P_{i+1}), send(\mathbf{R}, P'_{i+1})] \mid 0 < i \leq n\}. \end{aligned}$$

and the following sets of actions, performed in compliance with the description in Section 4:

- $Act_{\mathbf{S}} = \{\lambda\} \cup \{sendP_i, acceptP_i \mid 0 < i \leq n\}$,
- $Act_{\mathbf{R}} = \{\lambda, nonce, stop\} \cup \{acceptP_i \mid 0 < i \leq n\}$,
- $Act_{\mathbf{I}} = \{\lambda\} \cup \{dropP_i, fakeP_i, acceptP_i \mid 0 < i \leq n\}$.

The intuitive meaning of **S**'s local states is the following: $[\cdot]$ represents **S**'s initial state in the protocol; $[recv(n_{\mathbf{R}})]$ represents the message sent by **R** in order to establish communication; $[sent(\mathbf{R}, P_0)]$ represents the fact that **S** has just sent packet $P_0$ to **R**; $[sent(\mathbf{R}, P_{i-1}), sent(\mathbf{R}, P_i)]$ and $[sent(\mathbf{R}, P_{i-1}), sent(\mathbf{R}, P_i), sent(\mathbf{R}, P_{i+1})]$ represent the fact that **S** has sent packets $P_j$, where $j \leq i+1$ and $0 < i \leq n$.

With regards to **S**'s actions, action $\lambda$ is the null-action, $sendP_i$ stands for **S** sending packet $P_i$, and $acceptP_i$ represents that **S** recognises packet $P_i$ as accepted by the receiver.

**R**'s local states above stand for the following: $[\cdot]$ represents **R**'s initial state in the protocol; $[sent(\mathbf{S}, n_{\mathbf{R}})]$ represents the fact that **R** has just sent the nonce $n_{\mathbf{R}}$ to **S** and is waiting for packets; $[stop]$ represents the fact that **R** has just stopped collecting packets; $[recv(P_0)]$, $[recv(P_0), recv(P_2)]$, $[recv(P_i),\ recv(P_{i+1})]$, $[recv(P_{i-1}), recv(P_i),\ recv(P_{i+2})]$ and $[recv(P_{i-1}),\ recv(P_i),\ recv(P_{i+1})]$ represent the packets **R** has received from **S**; $[recv(P_0), recv(P'_1)]$, $[recv(P_0), recv(P'_1),\ recv(P'_2)]$, $[recv(P_0), recv(P'_1), recv(P_2)]$, and $[recv(P_i),\ recv(P_{i+1}), recv(P'_{i+2})]$ represent the faked packets **R** has received. As regards to **R**'s actions, $acceptP_i$ represents **R** accepting packet $P_i$ as authentic; the other action names have intuitive correspondences.

For what concerns **I**, $[\cdot]$ represents **I**'s initial state in the protocol; $[recv(n_{\mathbf{R}})]$ stands for **I**'s state following the interception of **R**'s initial message to **S**; $[recv(P_0)]$, $[recv(P_i),\ recv(P_{i+1})]$ and $[recv(P_{i-1}),\ recv(P_i), recv(P_{i+1})]$ represent the packets intercepted by **I**; $[recv(P_0), recv(P_1), send(\mathbf{R}, P'_1)]$, $[recv(P_0), recv(P_1),\ send(\mathbf{R}, P'_1), recv(P_2)]$, $[recv(P_0), recv(P_1), send(\mathbf{R}, P'_1), recv(P_2),\ send(\mathbf{R}, P'_2)]$, and $[recv(P_{i-1}), recv(P_i), recv(P_{i+1}), send(\mathbf{R}, P'_{i+1})]$ represent the packets intercepted by **I** and their faked versions. The action $acceptP_i$ denotes the fact that intruder is not able to fake or drop the packet $P_i$; $dropP_i$ (respectively $fakeP_i$) encodes the action of **I** dropping (respectively faking) packet $P_i$.

We have now defined the set of states and set of actions for the multi-agent system representing TESLA, so we can describe how the protocol evolves. In the multi-agents settings this is defined by means of an evolution function $t : S \times Act \rightarrow 2^{L_{\mathbf{S}} \times L_{\mathbf{R}} \times L_{\mathbf{I}}}$, where $Act \subseteq Act_{\mathbf{S}} \times Act_{\mathbf{R}} \times Act_{\mathbf{I}}$ and $S \subseteq (L_{\mathbf{S}} \times L_{\mathbf{R}} \times L_{\mathbf{I}})$. The function $t$ gives the transition relation $T$; namely, for all the $s,\ s' \in S$, $(s, s') \in T$ if there exists an $act \in Act$ such that $t(s, act) = s'$. We do not report here the full evolution function for TESLA; this can be found in [17].

To finalise the description of the interpreted system $M$ for TESLA, we have to define a valuation function $\mathcal{V} : S \rightarrow 2^{\mathcal{PV}}$ and the awareness functions $\mathbb{A}_X : L_X \rightarrow 2^{\mathcal{WF}(TDL)}$, for $X \in \{\mathbf{S}, \mathbf{R}, \mathbf{I}\}$. We first introduce the following set $\mathcal{PV}$ of propositional variables:

$$\mathcal{PV} = \{has_{\mathbf{R}}(m), sent_{\mathbf{S}}(m), received_{\mathbf{R}}(m), dropped_{\mathbf{I}}(m), faked_{\mathbf{I}}(m) \mid m \in \mathbb{M}\}.$$

We define $\mathcal{V} : S \rightarrow 2^{\mathcal{PV}}$ as follows:

- $has_{\mathbf{R}}(t_i) \in \mathcal{V}(s)$ if there exist packets $P_{i-1}$, $P_i$ and $P_{i+1}$ such that $f(k_i) \sqsubseteq P_{i-1}$, $t_i \sqsubseteq P_i$, $k_i \sqsubseteq P_{i+1}$, $recv(P_{i-1}) \in l_{\mathbf{R}}(s)$, $recv(P_i) \in l_{\mathbf{R}}(s)$ and $recv(P_{i+1}) \in l_{\mathbf{R}}(s)$,
- $sent_{\mathbf{S}}(m) \in \mathcal{V}(s)$ if there exists packet $P_i$ such that $m \sqsubseteq P_i$ and $sent(\mathbf{R}, P_i) \in l_{\mathbf{S}}(s)$, for any $m \in M_{\mathbf{S}}$,
- $received_{\mathbf{R}}(m) \in \mathcal{V}(s)$ if $recv(m) \in l_{\mathbf{R}}(s)$, for any $m \in M_{\mathbf{S}} \cup M_{\mathbf{I}}$,
- $dropped_{\mathbf{I}}(m) \in \mathcal{V}(s)$ if $recv(m) \notin l_{\mathbf{R}}(s)$ and $recv(m) \in l_{\mathbf{I}}(s)$, for any $m \in M_{\mathbf{S}}$,
- $faked_{\mathbf{I}}(m) \in \mathcal{V}(s)$ if there exist packets $P_j$ such that $m \sqsubseteq P_j$ and $send(\mathbf{R}, P_j) \in l_{\mathbf{I}}(s)$, for any $m \in M_{\mathbf{S}} \cup M_{\mathbf{I}}$.

For **R** we take the following awareness function $\mathbb{A}_{\mathbf{R}} : L_{\mathbf{R}} \rightarrow 2^{\mathcal{WF}(\mathrm{TDL})}$. Let $l \in L_{\mathbf{R}}$ and $\alpha$ be a TDL formula. Then, $\alpha \in \mathbb{A}_{\mathbf{R}}(l)$ if:

- $\alpha = received_{\mathbf{R}}(m)$ and $recv(m) \in l$ and $m \in M_{\mathbf{S}} \cup M_{\mathbf{I}}$,
- $\alpha = faked_{\mathbf{I}}(m)$ and $l = [stop]$ and $m \in M_{\mathbf{S}} \cup M_{\mathbf{I}}$,

- $\alpha = dropped_{\mathbf{I}}(m)$ and $l = [stop]$ and $m \in M_{\mathbf{S}}$,
- $\alpha = has_{\mathbf{R}}(m)$ and ($recv(m) \in l$ or $\exists m'$ such that $m \sqsubseteq m'$ and $recv(m') \in l$) and $m \in M_{\mathbf{S}} \cup M_{\mathbf{I}}$.

For $X \in \{\mathbf{S}, \mathbf{I}\}$, the awareness function $\mathbb{A}_X : L_X \rightarrow 2^{\mathcal{WF}(\mathrm{TDL})}$ is the following: for any $l \in L_X$, $\mathbb{A}_X(l) = \emptyset$.

To generate automatically the above interpreted system representing TESLA we developed a program in C++ that for an input $n$ of packets used generates the corresponding ISPL code (see Figure 3) to be used with MCMAS-X. In this way we can automatically produce not just one but a number of instances of the protocol. This helps us evaluate the performance of MCMAS-X.

```
Agent Receiver
 Lstate={empty,send_s_nr,stop,recv_p0,recv_p0_recv_p1,recv_p0_recv_p2,...};
 Action = {nothing,nonce,stop,accept_p1,accept_p2}; Protocol:
 empty : {nonce};              recv_p0 : {nothing};
 send_s_nr : {nothing};        stop : {stop};
 recv_p0_recv_p2 : {stop};     recv_p0_recv_p1 : {nothing};
end
Protocol Ev:
  stop if ((Lstate=stop and Action=stop and Sender.Action=nothing and
    Intruder.Action=nothing) or (Lstate=recv_p0_recv_p2 and Action=stop
    and Sender.Action=nothing and Intruder.Action=nothing) or ...);
 ...
end Ev Aware:
 recv_p0 : {received_r_p0,has_r_p0};
 recv_p0_recv_p1 : {received_r_p0,received_r_p1,has_r_p0,has_r_p1};
 recv_p0_recv_p2 : {received_r_p0,has_r_p0,received_r_p2,has_r_p2};
 ...
end Aware
end Agent
```

Figure 3. A fragment of **R**'s definition in the ISPL format for $n = 2$.

Given the interpreted system $M$ of TESLA as defined above, we now set out to check by means of MCMAS-X all the properties examined in [17]. First we would like to establish whether or not TESLA satisfies the desired security property: *"the receiver does not accept any message unless it was actually sent by the sender"*, i.e., whether or not $M$ is a model for the following TDL formula: for any $0 < i < n$,

$$has_{\mathbf{R}}(t_i) \Rightarrow (sent_{\mathbf{S}}(P_{i-1}) \wedge sent_{\mathbf{S}}(P_i) \wedge sent_{\mathbf{S}}(P_{i+1})). \quad (1)$$

Next we would like to check whether or not TESLA satisfies the stronger property *"the receiver does not accept any message unless he knows that it was actually sent by the sender"*. This is expressed by the following TDL formula: for any $0 < i < n$,

$$has_{\mathbf{R}}(t_i) \Rightarrow \mathcal{K}_{\mathbf{R}}(sent_{\mathbf{S}}(P_{i-1}) \wedge sent_{\mathbf{S}}(P_i) \wedge sent_{\mathbf{S}}(P_{i+1})). \quad (2)$$

Further, we would like to check whether TESLA meets the following properties: (3) "*it is always the case that the receiver does not accept any message unless he knows that it was actually sent by the*

*sender"*. (4) *"the principals know about the presence of the intruder"*. (5) the receiver is able to check the source of messages, i.e., *"if a packet is faked, then the receiver would deduce this"*. (6) *"if the receiver receives some packets $P_{i-1}$, $P_i$, and $P_{i+1}$ with a message $t_i \sqsubseteq P_i$, and he does not accept $t_i$, then he knows that at least one of the packets was not sent by the intended sender"*. In other words, if a packet was indeed faked, the receiver is able to deduce this fact. (7) *"if the intruder drops a packet, the receiver will deduce this fact"*.

The properties above can be expressed in a temporal-epistemic language by means of the formulae below.

$$\mathrm{A}G(has_{\mathbf{R}}(t_i) \Rightarrow \mathcal{K}_{\mathbf{R}}(sent_{\mathbf{S}}(P_{i-1}) \wedge sent_{\mathbf{S}}(P_i) \wedge sent_{\mathbf{S}}(P_{i+1}))) \tag{3}$$

$$K_{\mathbf{S}}\mathrm{E}F(sent_{\mathbf{S}}(P_i) \wedge \neg received_{\mathbf{R}}(P_i)) \tag{4}$$

$$faked_{\mathbf{I}}(P_i) \Rightarrow \mathcal{D}_{\mathbf{R}}(faked_{\mathbf{I}}(P_i)) \tag{5}$$

$$\begin{gathered}(received_{\mathbf{R}}(P_{i-1}) \wedge received_{\mathbf{R}}(P_i) \wedge received_{\mathbf{R}}(P_{i+1}) \wedge \neg has_{\mathbf{R}}(t_i)) \Rightarrow \\ \big(\mathcal{K}_{\mathbf{R}}(\neg sent_{\mathbf{S}}(P_{i-1}) \vee \neg sent_{\mathbf{S}}(P_i) \vee \neg sent_{\mathbf{S}}(P_{i+1})) \wedge \\ (\mathcal{D}_{\mathbf{R}}(fake_{\mathbf{I}}(P_{i-1})) \vee \mathcal{D}_{\mathbf{R}}(fake_{\mathbf{I}}(P_i)) \vee \mathcal{D}_{\mathbf{R}}(fake_{\mathbf{I}}(P_{i+1})))\big)\end{gathered} \tag{6}$$

$$\mathrm{A}G(dropped_{\mathbf{I}}(P_i) \Rightarrow \mathcal{D}_{\mathbf{R}}(dropped_{\mathbf{I}}(P_i)) \tag{7}$$

All formulae above were successfully verified by MCMAS-X thereby demonstrating the correctness of the protocol.

## 6. Experimental results and conclusions

We have employed the ISPL generator defined in the previous section to create a number of instances of the TESLA protocol, from 5 to 320 steps. We have verified all formulae above for all steps analysed, demonstrating the correctness of TESLA with respect to the specifications above. While process algebras [4] and Lynch-Vaandrager automata [2] have previously been used to analyse TESLA, our results specifically demonstrate its correctness with respect to the temporal epistemic specifications above.

MCMAS-X uses OBDDs to verify the properties. Consequently most of the computational time spent by the model checker is used to construct a symbolic representation of the model for the system. Table 1 reports some experimental results obtained using a MacBook Pro equipped with a 2.1GHz Intel processor, 2GBytes of RAM, running Mac OS X 10.4.6. The first column reports the number of packets, the second column contains the time required for the verification, while the third and the fourth column provide information about space requirements. In particular, column three lists the number of variables required to encode the example: from this value the size of the model can be deducted. The size of the model can also be estimated by evaluating the number of local states and actions required (see Section 4), as follows:

- The number of local states for **S** is $3 + 2n$, and the number of actions is $1 + 2n$ (where $n$ is the number of steps of the protocol.
- The number of local states for **R** is $8 + 4n$, and the number of actions is $3 + n$.
- The number of local states for **I** is $4 + 4n$, and the number of actions is $1 + 3n$.

Therefore, the size of the model is in the order of $n^{15}$, obtained by multiplying the number of local states (with an additional power of two for "next" variable), by the number of actions. The number of Boolean variables required to encode this model is the logarithm in base two of the size of model as computed above. For instance, 85 Boolean variables are required when $n = 200$, corresponding to a model of size $2^{85} \approx 4 \cdot 10^{25}$.

| N. of packets | Time (sec) | N. of BDD variables | Memory (bytes) |
|---|---|---|---|
| 5 | 2 | 40 | 4612376 |
| 10 | 3 | 48 | 4737832 |
| 20 | 8 | 55 | 5644888 |
| 50 | 25 | 67 | 6562280 |
| 100 | 38 | 76 | 9572968 |
| 150 | 77 | 82 | 9191848 |
| 200 | 92 | 85 | 10674616 |
| 250 | 110 | 91 | 11481224 |
| 320 | 190 | 91 | 15703560 |

Table 1. Experimental results.

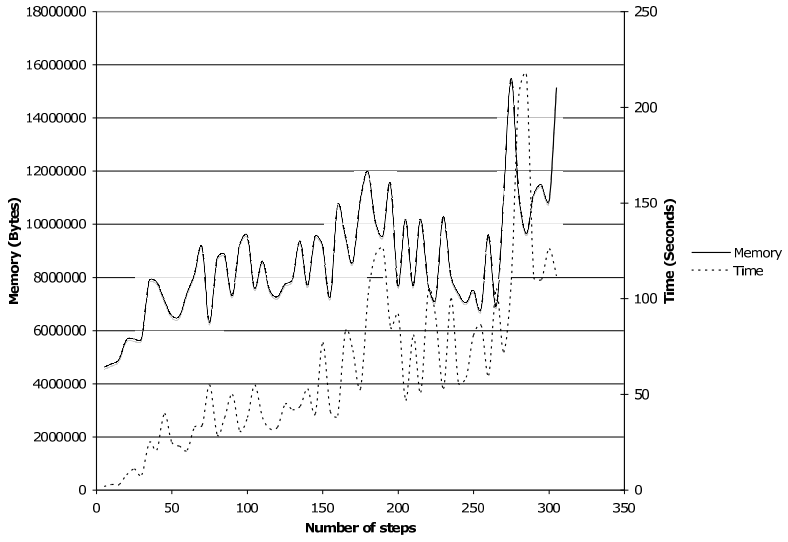


Figure 4. Experimental results.

Figure 4 depicts all the experimental results for time and memory requirements. The oscillating behaviour of the memory requirements shown in the figure is generated by the heuristic techniques employed in the construction of OBDDs (a similar behaviour was observed for a different example in [12]). Nevertheless, an increasing trend is evident, especially for time requirements (dotted line).

Another factor which may impact the results is the kind of heuristics employed by MCMAS-X for variable reordering. Currently, we adopt the default reordering methods provided by [13], which is triggered only when the OBDDs reach a certain size. Moreover, we do not partition the transition relation but we treat it as a monolithic OBDD. We leave the issue of partitioning, variable grouping, and variable reordering open for future investigation.

Given that no other model checker is available to verify explicit knowledge we cannot offer a direct comparison of the results above. Note though that on their own they do seem entirely adequate. Obviously, other specialised model checkers exist to verify temporal only properties (or simply reachability) for security protocols, notably AVISPA [3], but given the different emphasis in the two approaches it would not seem appropriate to compare experimental results.

# References


[1] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[2] M. Archer. Proving correctness of the basic TESLA multicast stream authentication protocol with TAME. In *Proceedings of Workshop on Issues in the Theory of Security*, 2002.

[3] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, J. Mantovani, S. Moedersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The Avispa tool for the automated validation of internet security protocols and applications. In *CAV 2005, 17th Int. Conf. on Computer Aided Verification*, Edinburgh, Scotland, UK, July 2005.

[4] P. J. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS'02)*, pages 146–161. Springer-Verlag, 2002.

[5] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.

[6] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

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

[9] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *LNCS*, pages 479–483. Springer-Verlag, 2004.

[10] J. Halpern, R. van der Meyden, and M. Y. Vardi. Complete axiomatisations for reasoning about knowledge and time. *SIAM Journal on Computing*, 33(3):674–703, 2003.

[11] J. Y. Halpern and R. Pucella. Modeling Adversaries in a Logic for Security Protocol Analysis. In *Proceedings of the Workshop on Formal Aspects of Security (FASec'02)*, volume 2629 of *LNCS*, pages 115–132. Springer-Verlag, 2002.

[12] M. Kacprzak, A. Lomuscio, A. Niewiadomski, W. Penczek, F. Raimondi, and M. Szreter. Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, 63(2,3):221–240, 2006.

[13] A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In H. Hermanns and J. Palsberg, editors, *Proceedings of TACAS 2006, Vienna*, volume 3920, pages 450–454. Springer Verlag, March 2006.

[14] A. Lomuscio and M. Sergot. Deontic interpreted systems. *Studia Logica*, 75(1):63–92, 2003.

[15] A. Lomuscio and B. Woźna. A combination of explicit and deductive knowledge with branching time: completeness and decidability results. In Matteo Baldoni, Ulle Endriss, Andrea Omicini, and Paolo Torroni, editors, *Declarative Agent Languages and Technologies III: Third International Workshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers*, volume 3904 of *LNAI*, pages 188 – 204. Springer Berlin/Heidelberg, 2006.

[16] A. Lomuscio and B. Woźna. A complete and decidable axiomatisation for deontic interpreted systems. In *Proceedings of the 8th International Workshop on Deontic Logic in Computer Science (DEON'06)*, volume 4048, pages 238 – 254. Springer Berlin / Heidelberg, July 2006.

[17] A. Lomuscio and B. Woźna. A complete and decidable security-specialised logic and its application to the tesla protocol. In Peter Stone and Gerhard Weiss, editors, *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS'06)*, pages 145–152, Hakodake, Japan, 2006. ACM Press.

[18] R. van der Meyden. Axioms for knowledge and time in distributed systems with perfect recall. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 448–457, Paris, France, 1994. IEEE Computer Society Press.

[19] R. van der Meyden and K. Wong. Complete axiomatizations for reasoning about knowledge and branching time. *Studia Logica*, 75(1):93–123, 2003.

[20] D. Mills. Network time protocol (version 3) specification, implementation and analysis. Technical Report 1305, RFC, March 1992.

[21] W. Nabialek, A. Niewiadomski, W. Penczek, A. Pólrola, and M. Szreter. VerICS 2004: A model checker for real time and multi-agent systems. In *Proceedings of the International Workshop on Concurrency, Specification and Programming (CS&P'04)*, volume 170 of *Informatik-Berichte*, pages 88–99. Humboldt University, 2004.

[22] A. Perrig, R. Canetti, J. D. Tygar, and Dawn X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, May 2000.

[23] R. Pucella. Deductive Algorithmic Knowledge. In *Proceedings of the 8th International Symposium on Artificial Intelligence and Mathematics (SAIM'04)*, Online Proceedings: AI&M 22-2004, 2004.

[24] F. Raimondi and A. Lomuscio. MCMAS-X - An extension of MCMAS to Explicit knowledge. http://www.cs.ucl.ac.uk/staff/f.raimondi/mcmas-x.tar.gz.

[25] F. Raimondi and A. Lomuscio. `http://www.cs.ucl.ac.uk/staff/f.raimondi/MCMAS/`, 2006.

[26] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 2007. To appear in Special issue on Logic-based agent verification.

[27] M. Wooldridge. *An introduction to multi-agent systems*. John Wiley, England, 2002.