Verification of the TESLA protocol in MCMAS-X

Alessio Lomuscio[1]*, Franco Raimondi[1] and Bożena Woźna[2]**

[1] Department of Computer Science, University College London
Gower Street, London WC1E 6BT, United Kingdom
email: {A.Lomuscio,F.Raimondi}@cs.ucl.ac.uk
[2] IMCS, Jan Długosz University
Armii Krajowej 13/15, 42-200 Częstochowa, Poland.
B.Wozna@mp.ajd.czest.pl

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

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 [25] 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 [7] often combined with temporal logic [16, 9, 17, 14]. 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 formally security protocols. While the original BAN logics [5] lacked computational grounding, more recent attempts [10, 15] 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 of explicit knowledge [7, 21, 13].

While model checkers for standard temporal (implicit) knowledge have recently been made available [8, 19, 12], they currently do not support explicit knowledge and derivable notions and so their applicability to an "epistemically-oriented" verification of security protocols has not been pursued yet[1].

* The authors acknowledge support from the EPSRC (grant GR/S49353 and grant CNA 04/04) and the Nuffield Foundation (grant NAL/690/G).

** The work presented here was prepared while B. Woźna was at University College London working on EPSRC grant GR/S49353.

[1] Anonymity protocols (such as the dining cryptographers) can successfully be analysed by using implicit knowledge only[8, 24, 11].

The aim of this research note is twofold: first we present a model checker that supports modalities for explicit and deductive knowledge; second we on the use of these techniques to validate the correctness of Tesla [20], a protocol for secure real-time streaming.

The work presented here builds upon our earlier analysis of Tesla[15] and our engineering of Mcmas [12], 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 [15], 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 [15].

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 $f : \mathbb{K} \to \{0, 1, \ldots\}$ 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 $\mathtt{MAC} : \mathbb{K} \times \mathbb{M} \to \{0, 1, \ldots\}$ 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 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 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 $k = k^{-1}$; for the 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 $\mathcal{WF}(\text{TDL})$ of well-formed TDL formulas is defined by the following grammar:

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

The meaning of temporal and epistemic operators is standard. We shall further use the shortcut $\mathcal{D}_i\alpha$ to represent $\mathrm{E}(\mathcal{K}_i\alpha\mathcal{U}\mathcal{X}_i\alpha)$. The formula $\mathcal{D}_i\varphi$ is read as "*agent i may deduce α (by some computational process)*". For more details we refer to [15, 13].

Interpreted Systems. In this section we will briefly summarise the multi-agent framework [7], 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 n agents and an environment, 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.

In the security settings, in particular here, 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 (see [7]), where a *run* is a sequence of all the possible global states. Thus, in these settings, an *interpreted system* for a multi-agent system is defined as a set of runs together with a valuation function for the propositional variables of the language under consideration. We interpret TDL on an extension of interpreted systems representing awareness sets; for more details we refer to [7, 15].

Definition 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. Then, 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 formulas to each state, for each $i \in \mathcal{AG}$.*

Awareness sets represent facts (expressed as TDL formulas) an agent is aware of at a given state; we refer to [7, 15] 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 (Satisfaction). *Let M be an interpreted system, s a state, and α, β* TDL *formulas. The satisfaction relation $\models$, indicating truth of a formula in M at state s, is defined inductively as follows:*
$(M, s) \models p$ *iff* $p \in \mathcal{V}(s)$, $\quad (M, s) \models \alpha \vee \beta$ *iff* $(M, s) \models \alpha$ *or* $(M, s) \models \beta$,
$(M, s) \models \neg\alpha$ *iff* $(M, s) \not\models \alpha$, $\quad (M, s) \models \mathrm{E}\bigcirc\alpha$ *iff* $(\exists\pi \in \Pi(s))(M, \pi(1)) \models \alpha$,
$(M, s) \models \mathrm{E}(\alpha \mathcal{U} \beta)$ *iff* $(\exists\pi \in \Pi(s))(\exists m \geq 0)[(M, \pi(m)) \models \beta$ *and* $(\forall j < m)(M, \pi(j)) \models \alpha]$,
$(M, s) \models \mathrm{A}(\alpha \mathcal{U} \beta)$ *iff* $(\forall\pi \in \Pi(s))(\exists m \geq 0)[(M, \pi(m)) \models \beta$ *and* $(\forall j < m)(M, \pi(j)) \models \alpha]$,
$(M, s) \models \mathcal{X}_i\alpha$ *iff* $(M, s) \models \mathcal{K}_i\alpha$ *and* $(M, s) \models \mathcal{A}_i(\alpha)$,
$(M, s) \models \mathcal{A}_i\alpha$ *iff* $\alpha \in \mathbb{A}_i(l_i(s))$, $(M, s) \models \mathcal{K}_i\alpha$ *iff* $(\forall s' \in S)$ $(s \sim_i s'$ *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:
$(M, s) \models sent_i(m)$ iff $(\exists m' \in \mathbb{M})(\exists j \in \mathcal{AG})$ such that $m \sqsubseteq m'$ and $sent(j, m') \in l_i(s)$,
$(M, s) \models received_i(m)$ iff $recv(m) \in l_i(s)$.

We leave definitions of the other security-specialised propositions open. Namely, for distinct protocols these propositions will be defined differently. They are not needed for the analysis of Tesla presented below.

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

3 The model checkers Mcmas and Mcmas-x

Overview of Mcmas. Mcmas is a symbolic model checker for multi-agent systems developped for the automatic verification of temporal and epistemic modalities in interpreted systems [7] as well as other modalities to reason about strategies and correct behaviour of agents [24]. Mcmas implements efficient verification algorithms based on ordered binary decision diagrams (Obdds, see [4] for more details).

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
```

Above `<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 [22] for more details about the ISPL language.

```
Agent SampleAgent
   Lstate = {s0,s1};  Lgreen = {s0,s1};  Action = {a1,a2};
 Protocol:
   s0: {a1};          s1: {a1, a2};
 end Protocol
 Ev:
   s1 if (Lstate=s0 and Action=a1 and AnotherAgent.Action=a7);
   s0 if (Lstate=s1 and Action=a1);
 end Ev
end Agent
```

Fig. 1. An agent's definition using ISPL.

Mcmas is available under the terms of the GNU General Public License (GPL) and it has been compiled on a number of platform. Mcmas is run from the command line and it accepts various input parameters to inspect and fine-tune its performance.

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). Given an interpreted system M, let $[\![\varphi]\!]$ denote the set of global states of M is which φ 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 [6, 24]) this definition can be re-casted in terms of Obdds, so that the set $[\![\mathcal{A}_i(\varphi)]\!]$ can be expressed as an Obdd. Consequently, the sets of states $[\![\mathcal{X}_i(\varphi)]\!]$ and $[\![\mathcal{D}_i(\varphi)]\!]$ can be expressed using Obdds, too.

We have implemented software procedures to perform the computation of these sets automatically in a tool called Mcmas-x available for download [23].

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 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`. Note that, following the definitions of Section 2 no consistency checks are made when defining awareness sets.

```
Agent SampleAgent
 Lstate = {s0,s1,s2,s3};  Lgreen = {s0,s1,s2};  Action = {a1,a2,a3};
 Protocol:   [...]    end Protocol
 Ev:         [...]    end Ev
 Aware:      s0 : {p1,p2};  s1 : {p2}; end Aware
end Agent
```

Fig. 2. An agent's definition using ISPL in Mcmas-x.

4 The Tesla protocol

In this section we introduce the *timed efficient stream loss-tolerant authentication* (Tesla) protocol [20]. 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-synchronized; this can be done via a simple two-message exchange using, for example, the NTP protocol [18]; (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 like `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 [1, 3], 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 [20].

Let $[x, y]$ denote the concatenation of x and y. 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) $\mathbf{S} \rightarrow \mathbf{R} : [P_1, \mathtt{MAC}(k_1, P_1)]$, for $P_1 = [t_1, f(k_2)]$
(2) $\mathbf{S} \rightarrow \mathbf{R} : [P_2, \mathtt{MAC}(k_2, P_2)]$, for $P_2 = [t_2, f(k_3), k_1]$
...
(n) $\mathbf{S} \rightarrow \mathbf{R} : [P_n, \mathtt{MAC}(k_n, P_n)]$, for $P_n = [t_n, f(k_{n+1}), k_{n-1}]$

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

Tesla guarantees, among others, the following security property: "*the receiver does not accept as authentic 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.

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 ($\mathbf{S}$) and a receiver ($\mathbf{R}$), communicating through an unreliable channel that is under complete control of an intruder ($\mathbf{I}$). 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 $\mathbf{S}$ has all the information he needs to prepare 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 $\mathbf{S}$'s initial database that remains accessible to him throughout the run. Moreover, we assume that $\mathbf{I}$ has all the information needed to prepare 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 (resend 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 $\mathbf{S}$, $\mathbf{R}$, and $\mathbf{I}$ use a shared PRF and a shared MAC function, $\mathbf{R}$ and $\mathbf{I}$ know the public key of $\mathbf{S}$, $\mathbf{S}$ and $\mathbf{I}$ begin with disjoint sets of keys, and that $\mathbf{R}$ 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 $\mathbf{S}$, $\mathbf{R}$ and $\mathbf{I}$, respectively:

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

$$\begin{gathered} L_{\mathbf{R}} = \{[\cdot], [sent(\mathbf{S}, n_{\mathbf{R}})], [stop], [recv(P_0)]\} \cup \{[recv(P_0), recv(P_2)]\} \cup \\ \{[recv(P_i), recv(P_{i+1})] \mid 0 \leq i \leq n\} \cup \{[recv(P_0), recv(P_1'), recv(P_2)]\} \cup \\ \{[recv(P_{i-1}), recv(P_i), recv(P_{i+1})] \mid 0 < i \leq n\} \cup \\ \{[recv(P_{i-1}), recv(P_i), recv(P_{i+2})] \mid 0 < i \leq n\} \cup \\ \{[recv(P_i), recv(P_{i+1}), recv(P_{i+2}')] \mid 0 \leq i \leq n\} \cup \\ \{[recv(P_0), recv(P_1')]\} \cup \{[recv(P_0), recv(P_1'), recv(P_2')]\}. \end{gathered}$$

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

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 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 λ 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 he 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 $t : S \times Act \to 2^{L_{\mathbf{S}} \times L_{\mathbf{R}} \times L_{\mathbf{I}}}$, 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 the full evolution function for TESLA; this can be found in [15].

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

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

We define $\mathcal{V} : S \to 2^{\mathcal{PV}}$ 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 $\mathbf{R}$ we take the following awareness function $\mathbb{A}_{\mathbf{R}} : L_{\mathbf{R}} \to 2^{\mathcal{WF}(\mathrm{TDL})}$. Let $l \in L_{\mathbf{R}}$ and α 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 $\mathbb{A}_X : L_X \to 2^{\mathcal{WF}(\mathrm{TDL})}$ is the following: for any $l \in L_X$, $\mathbb{A}_X(l) = \emptyset$.

To generate automatically the above interpreted system of TESLA we have produced a C++ program that given the number n of packets generates the corresponding ISPL code (see Figure 3) to be used with MCMAS-X. In this way we can generate a number of instances of the protocol which can help evaluate the performance of MCMAS-X.

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 [15]. First we would like to establish whether or not TESLA satisfies the desired security property: *"the receiver does not accept as authentic 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 $0 < i < n$,

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

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

```
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
```

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

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

Further, we would like to check whether TESLA meets the following properties: (3) *"it is always the case that the receiver does not accept as authentic 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 as authentic, 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) *"the intruder has to send a packet at each interval, which was agreed by the sender and the receiver at the beginning of the transmission under consideration"*.

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

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

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

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

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

$$\big(\mathcal{D}_{\mathbf{R}}(fake_{\mathbf{I}}(P_{i-1})) \vee \mathcal{D}_{\mathbf{R}}(fake_{\mathbf{I}}(P_i)) \vee \mathcal{D}_{\mathbf{R}}(fake_{\mathbf{I}}(P_{i+1})))\big)$$

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

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 formulas above for all steps analysed, demonstrating the correctness of Tesla with respect to the specifications above. While process algebras [3] and Lynch-Vaandrager automata [1] have previously been used to analyse the protocol, our results demonstrate the correctness of it 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. For instance, 85 Boolean variables are required when $n = 200$, corresponding to a model of size $2^{85} \approx 4 \cdot 10^{25}$. The last column reports the actual memory used by Mcmas-x.

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 depicts all the experimental results for time and memory requirements. The oscillating behaviour of the memory requirements shown in the figure is justified by the heuristic techniques employed in the construction of Obdds (a similar behaviour was observed for a different example in [11]). Nevertheless, an increasing trend is evident, especially for time requirements (dotted line).

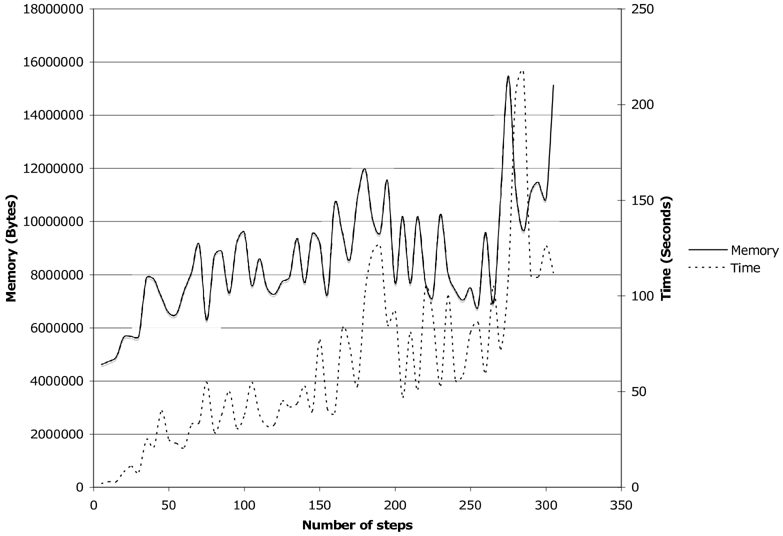

Fig. 4. Experimental results.

Given that no other model checker is available to verify explicit knowledge we cannot offer a direct comparison of the results above. On their own they do seem adequate. Obviously, other specialised model checkers exist to verify temporal only properties (or simply reachability) for security protocols, notably AVISPA [2], but given the different emphasis in the two approaches it would not seem appropriate to compare experimental results.

References

1. 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.
2. 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 *Proceedings of CAV'05*. Springer-Verlag, 2005.
3. P. J. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In *Proceedings of ESORICS'02*, pp. 146–161. Springer-Verlag, 2002.
4. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8):677–691, 1986.
5. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.
6. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

7. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, 1995.
8. P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *Proceedings of CAV'04*, volume 3114 of *LNCS*, pp. 479–483. Springer-Verlag, 2004.
9. 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.
10. J. Y. Halpern and R. Pucella. Modeling Adversaries in a Logic for Security Protocol Analysis. In *Proceedings of FASec'02*, volume 2629 of *LNCS*, pp. 115–132. Springer-Verlag, 2002.
11. 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*, 2006. To appear.
12. A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In *Proceedings of TACAS'06*, volume 3920, pp. 450–454. Springer Verlag, 2006.
13. A. Lomuscio and B. Woźna. A combination of explicit and deductive knowledge with branching time: completeness and decidability results. In *Proceedings of DALT'05*, volume 3904 of *LNAI*, pp. 188 – 204. Springer Berlin/Heidelberg, 2006.
14. A. Lomuscio and B. Woźna. A complete and decidable axiomatisation for deontic interpreted systems. In *Proceedings of DEON'06*, July 2006. To appear.
15. A. Lomuscio and B. Woźna. A complete and decidable security-specialised logic and its application to the TESLA protocol. In *Proceedings of AAMAS'06*, pp. 145–152, Japan, 2006. ACM Press.
16. R. van der Meyden. Axioms for knowledge and time in distributed systems with perfect recall. In *Proceedings of 9th IEEE Symposium on Logic in Computer Science*, pp. 448–457. IEEE Computer Society Press, 1994.
17. R. van der Meyden and K. Wong. Complete axiomatizations for reasoning about knowledge and branching time. *Studia Logica*, 75(1):93–123, 2003.
18. D. Mills. Network time protocol (version 3) specification, implementation and analysis. Technical Report 1305, RFC, March 1992.
19. 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 CS&P'04*, volume 170 of *Informatik-Berichte*, pp. 88–99. 2004.
20. 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*, pp. 56–73, May 2000.
21. R. Pucella. Deductive Algorithmic Knowledge. In *Proceedings of SAIM'04*, Online Proceedings: AI&M 22-2004, 2004.
22. F. Raimondi and A. Lomuscio. MCMAS - A tool for verification of multi-agent systems. http://www.cs.ucl.ac.uk/staff/f.raimondi/MCMAS/.
23. 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.
24. F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via OBDDs. *Journal of Applied Logic*, 2005. To appear in Special issue on Logic-based agent verification.
25. M. Wooldridge. *An introduction to multi-agent systems.* John Wiley, England, 2002.