Automatic Verification of Temporal-Epistemic Properties of Cryptographic Protocols

I. Boureanu, M. Cohen, A. Lomuscio

Department of Computing
Imperial College London
London, UK
{I.Boureanu07,Mika.Cohen,A.Lomuscio}@imperial.ac.uk

ABSTRACT. *We present a technique for automatically verifying cryptographic protocols specified in the mainstream specification language CAPSL. We define a translation from CAPSL models into interpreted systems, a popular semantics for temporal-epistemic logic, and rewrite CAPSL goals as temporal-epistemic specifications. We present a compiler that implements this translation. The compiler links to the symbolic model checker MCMAS. We evaluate the technique on protocols in the Clark-Jacobs library and in the SPORE repository against custom secrecy and authentication requirements.*

KEYWORDS: *security protocols, model-checking, epistemic logic*

DOI:10.3166/JANCL.18.1–24

1. Introduction

Logic and formal methods can play a useful role to reason about security protocols. In this context logic can be adopted as a non ambiguous language to express properties of protocols. Formal methods, especially theorem proving and model checking, can be employed as semi-automatic methodologies to attempt to verify that security protocols meet their specifications.

Currently specifications of security protocols (authentication, non-repudiation, etc.) are most often given in terms of reachability properties only, or, in more advanced approaches, in linear temporal logic, e.g., (Corin *et al.*, 2006). It has long been recognised though, including in the security literature, that richer specification languages would enable *more expressive and intuitive* specifications. The seminal work in security on BAN logics (Burrows *et al.*, 1990), where the case for the use of logics for knowledge and related concepts was put forward, is a case in point. Indeed, certain

security specifications, such as anonymity (Syverson *et al.*, 1999), seem only appropriately expressed in terms of knowledge (or lack of it) of protocol participants and intruders. Secrecy and other concepts such as authentication seem also naturally expressed by means of temporal epistemic specifications.

While epistemic, i.e., knowledge-based, specifications in the original BAN approach lacked a concrete semantics, this was rectified in later research (Fagin *et al.*, 1995; Abadi *et al.*, 1991; Cohen *et al.*, 2007). This has enabled several well-founded *theoretical* approaches employing epistemic logic in a security setting to be put forward (Halpern *et al.*, 2004; Halpern *et al.*, 2002; Pucella *et al.*, 2002). In parallel to this there has been considerable progress (Kacprzak *et al.*, 2008; Meyden *et al.*, 2004a; Lomuscio *et al.*, 2009) in solving efficiently the model checking problem for temporal-epistemic logics. This makes it possible, at least in principle, to develop symbolic model checking toolkits, based on temporal-epistemic formalisms and optimised for the verification of security protocols.

Of course model checking (Clarke *et al.*, 1999) of security protocols is an active area of research in security verification with a wealth of considerable results (Armando *et al.*, 2005; Basin *et al.*, 2005; Lowe, 1998). However, the traditional analysis typically focuses only on either reachability or trace properties of the protocols. Instead, in this line of work, our aim is to provide techniques and tools for the automatic verification of rich, epistemic-based specifications of security protocols.

Model checking of epistemic security requirements has been investigated before (Meyden *et al.*, 2004b; Lomuscio *et al.*, 2007; Lomuscio *et al.*, 2009). However, the consequent state space explosion was not studied there, thus making the approaches not viable for general deployment. Also, these works analysed specific protocols in an ad-hoc way by modelling them in existing model checkers, and no general methodology was put forward. Closer to our line of work is (Lomuscio *et al.*, 2008) where an optimised, trace-based semantics for temporal-epistemic logic based on interpreted systems (Fagin *et al.*, 1995) was put forward and a basic bounded model checking algorithm presented. However, no details were given for the automatic generation of the models to be checked, thereby limiting its possible impact.

This paper attempts to provide a fully-automatic methodology for checking authentication protocols given in CAPSL (Common Authentication Protocol Specification Language) by means of temporal epistemic specifications. Our main contribution is a mapping and a compiler from protocol descriptions given in CAPSL into ISPL (Interpreted Systems Programming Language), the input language for MCMAS (Lomuscio *et al.*, 2009), a BDD-based model checker for multi-agent systems supporting temporal-epistemic specifications. The translation is especially optimised to limit the state explosion and benefit from MCMAS's various optimisations. In line with most security verification formalisms and tools (Armando *et al.*, 2005; Basin *et al.*, 2005; Lowe, 1998; Blanchet, 2004; Boreale, 2001; Corin *et al.*, 2006; Meadows, 1996; Paulson, 1999), the methodology we present assumes a bounded number of concurrent protocol sessions instantiated to run concurrently.

The rest of the paper is organised as follows. In Section 2 we present background concepts and introduce the notation used in the rest of the paper. Section 3 presents the core principles of the translation from CAPSL security protocols descriptions into the semantics for temporal-epistemic logic that we employ here. In Section 4 we give details of the compilation from security requirements expressed as CAPSL goals into temporal-epistemic specifications. In Section 5 we present the implementation of the toolkit and discuss the experimental results obtained for some well known authentication and key-establishment protocols. We conclude in Section 6 by discussing related work and future research.

2. Preliminaries

In this section we introduce the necessary background material and fix the notation used in the rest of the paper. Specifically, we first recall some basic notions for high-level protocol description languages, security goals, and protocol scenarios. We then summarise basic notions in temporal-epistemic logic and model checking.

2.1. *High Level Descriptions of Security Protocols*

A security protocol, or a cryptographic protocol, is a communication protocol required to ensure certain security-sensitive properties. Such requirements are usually referred to as *security protocol goals* or, simply, *goals*. Security protocols can be classified according to the goals they aim to establish. In this paper, we will only be concerned with authentication and key-establishment protocols (Clark *et al.*, 1999).

Over the last decade, several high-level description languages for security protocols, e.g., HLPSL (Oheimb, 2005), CAPSL (Millen, 2001), etc., have been designed and employed in protocol analysis. These languages provide a formal methodology for describing protocols by specifying the participants behaviour, the data exchanged, the communication flow, and the security goals to be achieved. Such high-level specifications are usually called *protocol descriptions* or, simply, *descriptions*. Over the years, repositories and libraries of protocol descriptions have been created (Clark *et al.*, 1997; Laboratoire Spécification et Vérification ENS Cachan, 2003; AVISPA-project, 2005).

CAPSL (Common Authentication Protocol Specification Language) (Millen, 2001) is a widely-employed high-level protocol description language. We illustrate the basic CAPSL features by means of the Needham-Schroeder Public Key (NSPK) security protocol (Needham *et al.*, 1978).

EXAMPLE 1 (NEEDHAM-SCHROEDER PUBLIC KEY DESCRIPTION IN CAPSL). —

```
PROTOCOL Needham-Schroeder Public Key;
VARIABLES
        A, B: Principal;
```

```
        Na, Nb: Nonce;
        Ka, Kb: Skey;
DENOTES
        Ka = pk(A); Kb = pk(B);
ASSUMPTIONS
        HOLDS A: Na; HOLDS B: Nb;
MESSAGES
        1. A -> B: {A,Na}Kb;
        2. B -> A: {Na,Nb}Ka;
        3. A -> B: {Nb}Kb;
GOALS
        PRECEDES A: B | Na; PRECEDES B: A | Nb;
        AGREE A,B : Na,Nb,A,B;
        SECRET Na;
        SECRET Nb;
END;
```

□

In a CAPSL description the VARIABLES section denotes the data used in the communication together with their cryptographic type: here, Ka and Kb are public encryption keys, A and B are generic parties engaged in the protocol, and Na, Nb are nonces. The DENOTES and ASSUMPTIONS sections encode the initial knowledge of the principals. In the example above A knows the nonce Na, B knows the nonce Nb, A knows B's public key Kb, and B knows A's public key Ka. The MESSAGES section specifies the *rules* of the protocol, i.e., the stream of messages to be exchanged. Specifically, the *sender* of each message is encoded together with its intended *receiver* and the *step* in the protocol execution at which the communication should take place. For instance, at step 1 A sends to B a message containing her identity and her nonce Na, all encrypted with B's public key. The GOALS section encodes the *security goals*. For the case under analysis A should hold Na before B learns it, B should hold Nb before A learns it, both should agree upon these values and upon their identities, and both Na and Nb should always remain secret to any other party.

2.2. *Security Goals in CAPSL*

The security goals in the GOALS section of a CAPSL protocol description are specified in a simple, high-level language built from atomic facts and belief and knowledge constructs. For the purposes of this paper we analyse the following CAPSL subset.

The atomic facts are defined by:

$$\alpha ::= \texttt{agree}\,\texttt{A} : \texttt{B} : \overline{\texttt{Var}} \mid \texttt{holds}\,\texttt{A} : \overline{\texttt{Var}} \mid \texttt{secret} : \overline{\texttt{Var}}$$

where A and B are CAPSL variables of type Principal and $\overline{\texttt{Var}}$ is a list of CAPSL variables. Informally, agree A : B : $\overline{\texttt{Var}}$ expresses that A agrees with B on $\overline{\texttt{Var}}$, i.e., A and B have the same values for the variables in $\overline{\texttt{Var}}$. The atomic goal holds A : $\overline{\texttt{Var}}$ states that A has the values for the variables in $\overline{\texttt{Var}}$. Finally, the atomic goal

$\texttt{secret} : \overline{\texttt{Var}}$ represents the situation where the values of the variables in $\overline{\texttt{Var}}$ are not in possession of any unintended party.

Complex goals in CAPSL are formed with belief and knowledge operators:

$$\gamma ::= \alpha \mid \texttt{Knows}\, A : \gamma \mid \texttt{Believes}\, A : \gamma$$

where α is an atomic fact as above. Informally, the goal $\texttt{Knows}\, A : \gamma$ expresses that A knows that γ holds, and the goal $\texttt{Believes}\, A : \gamma$ states that A is justified in believing that γ holds. The CAPSL documentation does not provide a full description of the precise meaning of the belief and knowledge operators. However the following example may serve as a guideline.

EXAMPLE 2 (DOXASTIC GOAL). — The goal

$$\texttt{Believes}\, B : \texttt{holds}\, A : \texttt{K} \tag{1}$$

is interpreted in (Denker *et al.*, 2000), page 3, as "if the B-session completes, B is justified in believing that A holds K. The belief assertion means that $\texttt{holds}\, A : \texttt{K}$ is interpreted in the context of B's values for A and K." . □

EXAMPLE 3 (ACKNOWLEDGED AUTHENTICATION). — In repeated authentication protocols, e.g. KSL (Kehne *et al.*, 1992), the ISO standard requires the acknowledgement of the other party's knowledge. This can be translated into CAPSL goals with two levels of nesting for the knowledge operators, as in:

$$\texttt{Knows}\, A : \texttt{Knows}\, B : \texttt{holds}\, A : \texttt{Ma} \tag{2}$$

which expresses that if A terminates its run of the protocol, then it knows that B knows that A has the value of Ma. □

EXAMPLE 4 (BAN GOAL). — The CAPSL language supports BAN logic specifications with several levels of nested knowledge and belief operators. As an illustration, according to the analysis in (Burrows *et al.*, 1990), if protocol parties are honest, the NSPK protocol establishes that B believes that A believes that B believes that the nonce Nb is secret. In CAPSL this is expressed by means of the following goal.

$$\texttt{Believes}\, B : \texttt{Believes}\, A : \texttt{Believes}\, B : \texttt{secret} : \texttt{Nb}$$

□

2.3. *Protocol Scenarios*

A CAPSL protocol description specifies the messages exchanged during a protocol session. The description implicitly associates each variable A of type Principal to a *protocol role*, denoted as A–$role$, specifying the messages A sends and receives in the protocol.

A protocol *participant* is an entity (e.g., a user, or a long-term process, such as a server) having an identity (e.g., *alice*) and some security related data such as public keys, long-term symmetric keys etc. While a participant may take different roles in different concurrent or sequential sessions, the public keys and identity she employs are the same across sessions. A *role instance*, or simply an *instance*, is a process that follows the steps of some specific protocol role; each such instance represents a particular protocol participant, i.e., uses the name of the participant as her identity, and uses the passwords, cryptographic keys, etc., belonging to that participant. We write $(alice, A\text{–}role)$ to represent an instance of participant *alice* playing an $A\text{–}role$. In general, we may have more than one instance of $(alice, A\text{–}role)$; in these cases we use labels to indicate which instance we refer to.

A role instance can be viewed as an *instantiated* protocol role; an *instantiation* is a function mapping variables in the protocol description to concrete values over domains (i.e., mapping principal variables to participant identities *alice*, *bob*, etc., and nonce variables to numbers, etc.). Instantiations are homomorphically extended to messages, rules and protocol roles.

In line with the Dolev-Yao threat model (Dolev *et al.*, 1983) we model an *intruder* (or "attacker") as a further participant in the communication exchange. In addition to the capabilities of any principal, the intruder is assumed to be able to eavesdrop all communication, compose and replay messages into any protocol session, and perform cryptographic operations when in possession of the correct keys. We assume the intruder has a public identity (with associated public and private keys) trusted by the honest principals, and may use these to communicate with other principals.

For most classes of protocols, checking (secrecy) goals in the presence of an attacker and under an unbounded number of role instances is an undecidable problem (Ramanujam *et al.*, 2003b). In practice, therefore, one usually verifies only a specific *protocol scenario*, given as a finite set of role instances. For instance, a possible scenario for Example 1 is the collection of instances $\{(alice, A\text{–}role), (bob, B\text{–}role), (alice, B\text{–}role)\}$, where the participant *alice* is engaged in a session as an $A\text{–}role$ instance and in another session as a $B\text{–}role$, while another participant named *bob* is part of a session as a $B\text{–}role$ instance. Experience has shown that, in most cases, if an attack exists on a protocol, the attack can be found in a small protocol scenario, comprising only a few instances (Lowe, 1998).

2.4. *Temporal-epistemic logic and model checking with MCMAS*

We here fix the notation on some basic notions related to temporal-epistemic logic and symbolic model checking which constitute the building blocks of our framework.

The *interpreted systems* (IS) formalism (Parikh *et al.*, 1985; Fagin *et al.*, 1995) describes a multi-agent system as follows. We assume a set $Ag = \{1, \ldots, n\}$ of agents and a special agent called the *Environment*, abbreviated E. We associate to each agent $i \in Ag$, a set L_i of possible *local states*, a set Act_i of *local actions*, and a *local protocol*

$P_i : L_i \rightarrow 2^{Act_i}$. In the following, we assume that any action performed by an agent in a state is enabled, i.e., that it follows its local protocol. For the environment we associate similar sets: L_E, Act_E, and a protocol function P_E. The transition relation for agent i is defined by the *evolution function* $t_i : L_i \times Act_1 \times \ldots \times Act_n \times Act_E \rightarrow L_i$ returning the resulting local state for agent i given the actions performed by all agents at the present round. The *evolution function* t_E of the environment is defined in a similar way. To describe the evolution of the whole system we consider a set of possible points or *global states* $G \subseteq \prod_{1 \leq i \leq n} L_i \times L_E$, a set of *joint actions* $Act = Act_1 \times \ldots \times Act_n \times Act_E$, a *joint protocol* $\overline{P} = (P_1, \ldots, P_n, P_E)$, and a *global evolution function* $\bar{t} = (t_1, \ldots, t_n, t_E)$ operating on global states by composing the $n + 1$ local evolution functions. A *path* $\pi = (g_0, g_1, \ldots)$ is an infinite sequence of global states such that $\bar{t}(g_k, act_k) = g_{k+1}$, where $act_k \in Act$ is a joint action whose components are all enabled, for each $k \geq 0$. For a path $\pi = (g_0, g_1, \ldots)$, we take $\pi(k) = g_k$. By $\Pi(g)$ we denote the set of all the paths starting at $g \in G$.

Given the above, an interpreted systems is a tuple $I = (G, I_0, \bar{t}, \sim_1, \ldots, \sim_n, V)$, where G is the set of the global states reachable from any initial global state in I_0 via the evolution function $\bar{t}$; $\sim_i \subseteq G \times G, i \in Ag$, are epistemic relations for agent i defined by $g \sim_i g'$ iff $l_i(g) = l_i(g')$, where $l_i : G \rightarrow L_i$ returns the local state of agent i given a global state; and $V : G \times PV \rightarrow \{true, false\}$ is an interpretation for the propositional variables PV in the language.

Interpreted systems are a standard semantics for branching time temporal-epistemic logic. The language we will use is defined by the following BNF syntax:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid D_\Gamma\varphi \mid AX\varphi \mid AG\varphi \mid A(\varphi U \varphi)$$

where $p \in PV, i \in Ag, \Gamma \subseteq Ag$.

As a simplification we will often refer to the language above as CTLK (Computational Tree Logic for Knowledge) even if it contains a distributed knowledge modality. The readings of the modalities above are as usual. Specifically, $AX\varphi$ stands for at all possible next steps φ holds, $AG\phi$ represents that in all possible computational paths ϕ always holds, and $A(\varphi U \psi)$ expresses that in all possible computational paths at some point ψ holds and φ holds at all states before then from the present state. The formula $K_i\varphi$ stands for the knowledge of agent i with respect to φ and $D_\Gamma\phi$ represents that the agents in the group $\Gamma \subseteq Ag$ have distributed knowledge of φ. We refer to (Fagin *et al.*, 1995) for more details.

Satisfactions for the language above is defined inductively on interpreted systems at a global point as follows. Let I be an interpreted system, $g = (l_1, \ldots, l_n, l_E)$ a global state, and φ, ψ formulas:

$(I, g) \models p$ iff $V(g, p) = true$,
$(I, g) \models \neg\varphi$ iff it is not the case that $(I, g) \models \varphi$,
$(I, g) \models \varphi \wedge \psi$ iff $(I, g) \models \varphi$ and $(I, g) \models \psi$,
$(I, g) \models K_i\phi$ iff for all $g' \in G$ if $g \sim_i g'$, then $(I, g') \models \phi$,
$(I, g) \models D_\Gamma\phi$ iff for all $g' \in G$, $g \sim_i g'$ for all $i \in \Gamma$, implies that $(I, g') \models \phi$,

$(I, g) \models AX\phi$ iff for all $\pi \in \Pi(g)$ we have $(I, \pi(1)) \models \phi$,
$(I, g) \models AG\phi$ iff for all $\pi \in \Pi(g)$ and for all $k \geq 0$ we have $(I, \pi(k)) \models \phi$,
$(I, g) \models A(\phi U \psi)$ iff for all $\pi \in \Pi(g))$ there exists a $k \geq 0$ such that
$(I, \pi(k)) \models \psi$ and for all $0 \leq j < k$ we have $(I, \pi(j)) \models \phi$.

Temporal-epistemic logic is a well-explored subject in applied logic. We refer to the references above for a survey on existing results.

MCMAS (Lomuscio *et al.*, 2009) is a BDD-based symbolic model checker for the verification of epistemic and ATL properties on systems described by means of variants of interpreted systems. MCMAS takes as input systems descriptions given in ISPL (Interpreted Systems Programming Language), a set of CTLK specifications to be checked, and returns whether or not the specifications are satisfied, giving, in most cases, a counter-model if they are not. An ISPL file uniquely denotes an interpreted system. In Example 5 an excerpt of an ISPL file is reproduced, denoting an agent of an interpreted system.

EXAMPLE 5 (SAMPLE ISPL CODE FOR AN AGENT). —

```
Agent sample_agent

Vars:
   state : {val1,val2,val3,val4};
end Vars
...
Actions = {action1,action2};

Protocol:
   (state=val1 or state=val4) : {action1};
   (state=val2): {action2};
end Protocol

Evolution:
   ...
   (state=val3)
   if
   (Action=action2) and (Environment.Action=action-e);
   ...
end Evolution
end Agent
```

□

The above `sample_agent` has four possible local states corresponding to the variable `state` (more complex types are supported). The possible actions of `sample_agent` are `action1` and `action2`, the former is enabled in states where `state=val1` and `state=val4`, the latter when `state=val2`. The agent moves to the local state `state=val3` when the action `action2` is performed locally together with action `action-e` for the environment. Several other functionalities are offered. We refer the reader to (Lomuscio *et al.*, 2009) for further details.

3. From protocol scenarios to interpreted systems

In this section we map authentication and key-establishment protocol scenarios into interpreted systems. We operate under some restrictions. Specifically, we assume that the runs generated by the protocols have a predetermined maximum number of interleaved sessions, the length of any message generated is bounded, and so are the initial data types considered (nonces, keys, etc).

Given a protocol scenario, our starting point is to map each role instance in the protocol scenario into an agent of an interpreted system and the intruder role into the environment agent. We assume a typed signature and a (free) term algebra formalising the cryptographic data in the protocol description underlying the scenario. We begin by describing how we obtain the local states, local actions and local protocol of an agent ag_A, the mapping of an A–$role$ instance. We will sometimes refer to such a generic agent ag_A as an A-agent.

Local States of agent ag_A.

An instantaneous possible local state of agent ag_A is a pair $(nr, view_{ag_A})$, where nr is a counter for the number of protocol steps executed by the agent and $view_{ag_A}$ is the *view* the agent has of its protocol session, i.e., an instantiation of some of the protocol variables for the session. Formally, $view_{ag_A}$ is a type-respecting map from variables in a *store* to cryptographic data; the store is an ordered list of (typed) protocol variables appearing in the A–$role$, mapped to agent ag_A.

EXAMPLE 6. — In the NSPK protocol description in Example 1, the A–$role$ and B–$role$ include the same variables. So, the store is the same for an A–agent and a B–agent: $(A\colon Node, B\colon Node, k_A\colon Skey, k_B\colon Skey, n_a\colon Nonce, n_b\colon Nonce)$. □

EXAMPLE 7. — A possible view for the NSPK store of an A–agent in the example above is the following assignment: $(A = alice, B = bob, k_A = pvk_{alice}, k_B = pbk_{bob}, n_A = r_1, n_b = \bot)$, where nb is unassigned at the time. If the context is clear, we omit the variables and write simply $(alice, bob, pvk_{alice}, pbk_{bob}, r_1, \bot)$ for the A–agent view. Note that although several agents may have the same store, in general their views of the stores are different. □

Actions of agent ag_A.

The actions available to agent ag_A are: $send\, M$, $receive\, T$ and the empty action ε, where M is an instantiated message and T is a symbolic (i.e., uninstantiated) message. Informally, the action $send\, M$ represents dispatching the message M to the network, while the action $receive\, T$ means waiting for an incoming message of the form T.

Local Protocol of agent ag_A.

The *local protocol* of agent ag_A formalises the moves described in the role instance which is mapped into this agent ag_A. Thus, for each protocol step numbered nr specified for A, we have: $P_{ag_A}((nr, view_{ag_A})) = \{send\, M\}$, if "$nr.\, A \rightarrow Y : M$" is a rule[1] in the role instance mapped into ag_A. The local protocol is implemented similarly for receiving actions. If there is no rule in the A–role with step nr, the agent ag_A is silent at step nr, i.e., $P_{ag_A}((nr, view_{ag_A})) = \{\varepsilon\}$.

The local evolution function of the agent ag_A will be defined below, once we have mapped the intruder role into the environment agent E of the interpreted systems.

Local state of the environment agent.

An instantaneous possible local state of the environment agent is a pair (X, h), where X is a set of cryptographic data observed and deduced in the protocol execution and h is list representing the history of observed actions.

EXAMPLE 8. — In an execution of the NSPK protocol, the intruder might acquire the knowledge set $X = \{\{A, na\}_{k_B} = \{alice, r_1\}_{pbk_{bob}}, \{A, na\}_{k_B} = \{joe, r_2\}_{pbk_{greg}}, A = alice, A = joe, B = bob, DATA = \{n_3, n_5, charlie, \ldots\}\}$ □

Knowledge sets can be considered extensions of agents' views in that they may contain any data of a complex cryptographic type not only values of atomic variables from the protocol description. Notice also that, unlike in the case of views, knowledge-sets may contain data from several (concurrent) sessions of the protocol. For instance, the knowledge set in Example 8 contains two different instances of $\{A, na\}_{k_B}$ corresponding to the different sessions the intruder participated in.

In line with the Dolev-Yao model, we assume that the intruder records every send-action performed by any agent.

EXAMPLE 9. — In an execution of the NSPK protocol, the intruder might have recorded the history $h = [ag_A.send\,\{alice, r_1\}_{pbk_{bob}}, ag_B.send\,\{r_1, r_2\}_{pbk_{alice}}, \ldots]$. □

Actions of the environment agent.

The actions available to the environment agent are: $intercept\, M$ and $transmit\, M$, where M are instantiated messages. Informally, the action $intercept\, M$ reflects the Dolev-Yao surveillance of the network: every action $send\, M$ performed by an honest agent synchronises with an $intercept\, M$ from the Environment. The action $transmit\, M$ represents the intruder inserting M into the network.

Local Protocol of environment agent.

The local protocol is defined such that when the intruder has message M, i.e., M is an entry in the current knowledge set X, the corresponding action $transmit\, M$ is

1. Y is an arbitrary principal in the description.

potentially triggered. At any local state, the Environment agent can perform the action $intercept\, M$, reflecting the continuous Dolev-Yao surveillance of the network.

Local Evolution Function of the environment agent.

The Environment's local evolution function formalises the interception of messages, decomposing, composing and sending of messages. Specifically, when the environment intercepts a message M from an agent ag_A, the environment records the observed *transmit* action in the history H, adds M to its knowledge set X, and then closes the set X under decompositions (decryption, unpairing) and compositions (encryption, pairing). Formally, if $\tilde{a}_E = interceptM$ and $\tilde{a}_{ag_A} = sendM$, then $t_E((X,H),\tilde{a}) = (X \cup M \cup \{M' | \{X \cup M\} \vdash M'\}, H \cup ag_A.send\, M)$, where $\vdash$ is the standard Dolev-Yao message inference. The notation $X \vdash M$ represents that the message M can be obtained from the set X of messages by successive decompositions and compositions. Given our systems are finite and bounded, $X \vdash M$ is decidable and the procedure for closing the set under $\vdash$ terminates.

Local Evolution Function of agent ag_A.

Having provided the actions of the environment, we can now describe the local evolution function for an arbitrary agent ag_A. We adopt the "matching receive" semantics used in other security protocols models, e.g. (Ramanujam *et al.*, 2003a; Rusinowitch *et al.*, 2001; Lomuscio *et al.*, 2008). Recall that the local view of ag_A reflects the protocol execution for ag_A so far. Informally, when the agent ag_A performs the action $receive\, T$, the agent awaits for an instantiation M for the symbolic message T, compares the atomic parts of M to her local state and accepts or drops M depending on whether the match is successful. More precisely, when the environment transmits a message M, obtained by instantiating the variables $\overline{X'}$ in T, the agent checks if the values of the variables $\overline{X} \subseteq X'$ in her local view $view_{ag_A}$ agree with the values of $\overline{X}$ in the message M, i.e., $view_{ag_A}.\overline{X} = M.\overline{X}$, abbreviated $out\_match(\overline{X})$. If they match, then the agent accepts the message, i.e., increments her counter nr and and sets the yet unassigned variables in $\overline{X}$ to their values in M. We write $set(X, view_{ag_A})$ for the result of setting the unassigned variable X in the local view $view_{ag_A}$ of the agent.

To illustrate the resulting local evolution function, consider an agent ag_A in the local state $(1, view_{ag_A})$, where the local view $view_{ag_A}$ is as in Example 7. Assume a joint action $\tilde{a}$, in which the environment action is $\tilde{a}_E = transmit(\{r'_1, r_2\}_{pbk_{alice}})$ and the action of the agent ag_A is $\tilde{a}_{ag_A} = receive(\{na, nb\}_{K_A})$. Then, assuming that $out\_match(na)$ holds, i.e., $r_1 = r'_1$, we have that $t_{ag_A}((1, view_{ag_A}), \tilde{a}) = (2, set(nb, view_{ag_A}))$ = $(2, (A = alice, B = bob, k_A = pvk_{alice}, k_B = pbk_{bob}, n_a = r_1, n_b = r_2)$ However, if $out\_match(na)$ fails, we get $t_{ag_A}((1, view_{ag_A}), \tilde{a}) = (1, view_{ag_A})$.

Initial states.

The *initial states* of the system are given by instantiating stores into views. The instantiation reflects the initial conditions for the agents to engage in the protocol,

including their choice of communication partners. Some variables in the local states can be initially left unassigned, or, equivalently, assigned to $\perp$ (cf. Example 7).

The environment's variables representing the intruder are initialised similarly.

This concludes our translation from scenarios to interpreted systems. We will describe an implementation of this in Section 5.

4. From CAPSL security goals to CTLK specifications

In this section we provide a translation ρ from the security goals (Subsection 2.2) found in a CAPSL protocol description to the corresponding CTLK property in the mapped IS.

Recall from Section 2.2 that a CAPSL atomic goal `agree A : B :` $\overline{\texttt{VAR}}$ refers to principals A and B sharing the value of the variables in $\overline{\texttt{VAR}}$. To account for the inherent multi-session instantiation of the goal, we formalise this with a specification stating that *any* A-agent i needs to agree with *some* B-agent j on the variables in $\overline{\texttt{VAR}}$. Intuitively, given this is the goal of the protocol, the agreement may not happen before the A-agent i has terminated its protocol run. Given the above, and also in light of existing CAPSL examples, we translate the CAPSL `agree` goal above as:

$$\rho(\texttt{agree A : B :}\ \overline{\texttt{VAR}}) \quad = \quad \bigwedge_{i \in A} AG(end(i) \rightarrow \bigvee_{j \in B} agree(i, j, \overline{\texttt{VAR}}))$$

where i ranges over A-agents, i.e., mappings of A–$role$ instances, j ranges over B-agents, i.e., mappings of B–$role$ instances, $agree(i, j, \overline{\texttt{VAR}})$ and $end(i)$ are helper predicates generated in the IS in order to express the final formula, $agree(i, j, \overline{\texttt{VAR}})$ abbreviates $\bigwedge_{Var \in \overline{\texttt{VAR}}} (i.Var = j.Var)$ with $i.Var$ (respectively $j.Var$) denoting variable Var of agent i (respectively agent j), and $end(i)$ abbreviates $i.Nr = n$, where Nr is agent i's variable referring to its current protocol step and n is the number of steps for role A in the given protocol description. Other atomic goals are translated similarly.

Recall from Section 2.2 that an epistemic goal `Knows A :` γ states that the A-agent knows that γ holds. Following the consideration above we represent this in CTLK as:

$$\rho(\texttt{Knows A :}\ \gamma) \quad = \quad \bigwedge_{i \in A} AG(end(i) \rightarrow K_i\, \rho^i(\gamma))$$

where $\rho^i(\gamma)$ is an appropriate translation of γ from the perspective of agent i, i.e., in the context of agent i's values for the variables appearing in γ (cf. Example 2).

More precisely, the relativised translation function ρ^i is defined inductively by:

$$\rho^i(\texttt{holds A :}\overline{\texttt{VAR}}) \quad = \quad \bigvee_{j \in A} (i.PartnerA = j.Id \wedge agree(i, j, \overline{\texttt{VAR}})) \tag{3}$$

$$\rho^i(\texttt{Knows A : } \gamma) \quad = \quad \bigvee_{j \in A} (i.PartnerA = j.Id \wedge K_j\, \rho^j(\gamma)) \tag{4}$$

and similarly for other atomic sub-formulae. In (3) and (4), `j.Id` is the identity of the participant that agent j represents, and $i.PartnerA$ is the identity of the participant that agent i regards as having the A–$role$ in the current session. In detail, here `j.Id` stands for `j.A`, since any A–agent j stores its identity in the variable $j.A$. Similarly, $i.PartnerA$ stands for `i.A`, as agent i stores the identity of the communication partner of an A–$role$ in his local variable A. Thus, according to (3), $\rho^i(\texttt{holds A : } \overline{\texttt{VAR}})$ states that there is some A-agent j, representing participant $i.PartnerA$, who agrees with agent i on the variables $\overline{\texttt{VAR}}$. According to (4), $\rho^i(\texttt{Knows A : } \gamma)$ states that there is some A-agent j, representing participant $i.PartnerA$, who knows $\rho^j(\gamma)$. We translate CAPSL's belief goals in the same way as we do for the epistemic ones. This is a reasonable approximation, since belief in CAPSL is introduced to refer to "justified belief" (Denker *et al.*, 2000).

$$\rho(\texttt{Believes A : } \gamma) \quad = \quad \bigwedge_{i \in A} AG(end(i) \rightarrow K_i\, \rho^i(\gamma))$$

$$\rho^i(\texttt{Believes A : } \gamma) \quad = \quad \bigvee_{j \in A} (i.PartnerA = j.Id \wedge K_j\, \rho^j(\gamma))$$

EXAMPLE 10 (ATOMIC GOAL). — According to the inductive definition given above the NSPK CAPSL goal:

```
agree A : B : B,Na
```

translates into:

$$\bigwedge_{i \in A} AG(i.Nr = 3 \rightarrow \bigvee_{j \in B} (i.\texttt{PartnerB} = j.\texttt{Id} \wedge i.\texttt{Na} = j.\texttt{Na}))$$

stating that whenever an A-agent i has completed three protocol steps, participant $i.PartnerB$ is represented by some B-agent j who agrees with i on the variable `Na`. □

EXAMPLE 11 (COMPLEX GOAL). — Consider the goal (1) from Example 2. Applying ρ on (1) yields:

$$\bigwedge_{i \in B} AG(i.Nr = 3 \rightarrow K_i\, \rho^i(\texttt{holds A : K}))$$

In turn, $\rho^i(\texttt{holds A : K})$ is rewritten as:

$$\bigvee_{j \in A} (i.PartnerA = j.Id \wedge i.\texttt{K} = j.\texttt{K})$$

Thus, the goal (1) translates to:

$$\bigwedge_{i \in B} AG(i.Nr = 3 \rightarrow K_i \bigvee_{j \in A} (i.PartnerA = j.Id \wedge i.\mathtt{K} = j.\mathtt{K}))$$

stating that whenever a B-agent i, e.g., representing a participant *bob*, has completed three protocol steps, agent i knows that the participant $i.PartnerA$, e.g., $alice$, is represented by some A-agent j which agrees with i on the variable `K`. □

EXAMPLE 12 (ACKNOWLEDGED AUTHENTICATION). — The acknowledged authentication goal (2) from Example 3 translates into:

$$\bigwedge_{i \in A} AG(end(i) \rightarrow K_i\, \rho^i(\mathtt{Knows\ B:\ holds\ A:\ Nb}))$$

where $\rho^i(\mathtt{Knows\ B:\ holds\ A:\ Nb})$ in turn expands to:

$$\bigvee_{j \in B} (i.PartnerB = j.Id \wedge K_j \bigvee_{k \in A} (j.PartnerA = k.Id \wedge j.\mathtt{Nb} = k.\mathtt{Nb}))$$

Thus, the CTLK translation of goal (2) states that whenever an A-agent i terminates its protocol run, she knows that participant $i.PartnerB$ is represented by some B-agent j who knows that participant $j.PartnerA$ is represented by some A-agent k who agrees with agent j on `Nb`. □

In the above, in line with much of the security literature, we focused on the properties of individual agents representing protocol participants (*alice*, *bob*, etc.). This is useful as agents are the actual parties in the protocol runs. However, even if this is normally not tackled in the mainstream security literature, it also seems of interest to attempt to analyse the epistemic properties of the participants themselves. The epistemic setting employed here makes this particularly straightforward. If we assume that, in as far as protocol information is concerned, participants have at their disposal all and only the information acquired by the agents representing them, the participant's knowledge is then the combined information of all agents representing her. The relevant epistemic notion here is the one of *distributed knowledge* of the group of agents representing a certain participant (see Section 2).

If we map the identity of the epistemic concepts in CAPSL goals to the actual participants we obtain:

$$\begin{aligned} \rho(\mathtt{Knows\ A\ :}\ \gamma) &= \bigwedge_{i \in A} AG(end(i) \rightarrow K_{i.Id}\, \rho^i(\gamma)) \\ \rho^i(\mathtt{Knows\ A\ :}\ \gamma) &= \bigvee_{j \in A} (i.PartnerA = j.Id \wedge K_{j.Id}\, \rho^j(\gamma)) \end{aligned}$$

where $K_{i.Id}$ (respectively $K_{j.Id}$) is an epistemic modality referring to participant $i.Id$ ($j.Id$ respectively). Since all information available to the participant $i.Id$, e.g., $alice$,

is information coming from the various agents representing $i.Id$, i.e. information from all sessions that $alice$ participates in, we interpret $K_{i.Id}\varphi$ as $D_\Gamma\varphi$ where Γ is the set of all agents representing $i.Id$.

EXAMPLE 13. — Returning to Example 11, the alternative mapping ρ translates the CAPSL goal (1) into:

$$\bigwedge_{i\in B} AG(i.Nr = 3 \rightarrow K_{i.Id} \bigvee_{j\in A} (i.PartnerA = j.Id \wedge i.\mathtt{K} = j.\mathtt{K}))$$

stating that whenever a B-agent i has completed three protocol steps, participant $i.Id$ knows that participant $i.PartnerA$ is represented by some A-agent j which agrees with i on the variable K. □

In this section we have provided a direct map from CAPSL goals to CTLK specifications. We stress that the translations provided here are simply one of the many that may be given depending on the interpretation of CAPSL's primitives. Indeed, the security literature displays a multitude of subtly different interpretations of atomic goals. We do not take a view in this paper on which is the "correct" interpretation, if one exists. Instead we aim to support several so that the user can choose the one he or she finds most appropriate. For this reason the tool presented in the next section produces a relatively large number of CTLK translations for each goal supplied in the CAPSL file.

5. Automatic Compilation of Protocol Scenarios into Interpreted Systems

In this section we give details of an implementation of the translations described in earlier sections. Specifically, we present *PD2IS* (Protocol Descriptions to Interpreted Systems), an automatic compiler from CAPSL protocol descriptions to ISPL programs. We evaluate the methodology by discussing experimental results obtained for the verification of formulas specifying secrecy and authentication properties.

5.1. *Implementation details*

PD2IS implements the translations described in Section 3 and Section 4. PD2IS takes as input a CAPSL protocol description and some user-defined parameters. These describe part of the instantiation (e.g., participant names for instantiating the principals, etc.) and bounds on the size of the scenarios to be considered (e.g., the maximum number of A–agents, etc.). With these parameters PD2IS generates all complete instantiations and then either produces the ISPL programs corresponding to the different possible scenarios up to the user-defined bounds, or just a particular ISPL file if this is fully described by the input parameters. For each such instantiated system PD2IS generates a set of ad-hoc propositions and uses these to construct the temporal-epistemic translations for the CAPSL goals. MCMAS is called for each ISPL file produced by

PD2IS. MCMAS returns the calls either by certifying that the specifications are satisfied or by returning detailed counterexamples. These are used by PD2IS to report details of the attack found on the protocol (i.e., the failure of one or more of the goals). PD2IS is open-source and coded in JAVA (Boureanu *et al.*, 2009).

From an architectural point of view PD2IS comprises four modules: *utils*, *parser*, *unmarshaller* and *producer* (see Figure 5.1).

The module *utils* is composed of two submodules: the *description* submodule and the *scenario-generator* submodule. The *description* submodule consists of a collection of XML schemas that encode the protocol signature and the term algebra as given in Section 3 (i.e., variables and their ranges, untyped messages and typed messages principals, and goals). The routines in the *scenario-generator* submodule scan part of the description file and input parameters and generate the data structures for the scenarios and the formulas to be checked. The *parser* module parses the CAPSL description and populates it in the context of the data structures provided by the *scenario-generator*. The parser outputs XML files describing the previously generated scenarios and the specifications to be checked. The *unmarshaller* module then converts these XML files into JAVA objects and populates the data structures describing the interpreted system. Finally, the *producer* module processes the structures created by the *unmarshaller* module into several ISPL files.

Note we use XML as an intermediate language to describe the interpreted systems under generation. Thus, PD2IS is designed as an expandable platform: by implementing another *producer* module we can compile into other languages.

We illustrate below ISPL code snippets generated by PD2IS from an NSPK protocol scenario. The first code sample is a simplified and commented version of the code generated for an A-agent. It can be seen that agents' local variables encode *views*; the agents' actions and local protocols contain instantiated *send* and *receive* actions; the agents' local evolutions are described by appropriate matching preconditions, and setting postconditions. The actual ISPL files produced by PD2IS are less intuitive than the ones presented below as they are heavily optimised to reduce the state-space of the generated model.

EXAMPLE 14. — Simplified ISPL code for an A-agent

```
Agent ag_A -- Encodes an NSPK instance (alice, A-role)
 Vars: --Encodes <VIEWS>
   -- The Id of the agent ag_A is stored in A:
   A: {alice};
   --The communication partner B is fixed:
   B: {bob}
   Na, Nb: {r1,r2,...};
   Step: {0,1,2,3};
 end Vars

 Protocol: --Encodes <A-role>
   --Step 0:
```

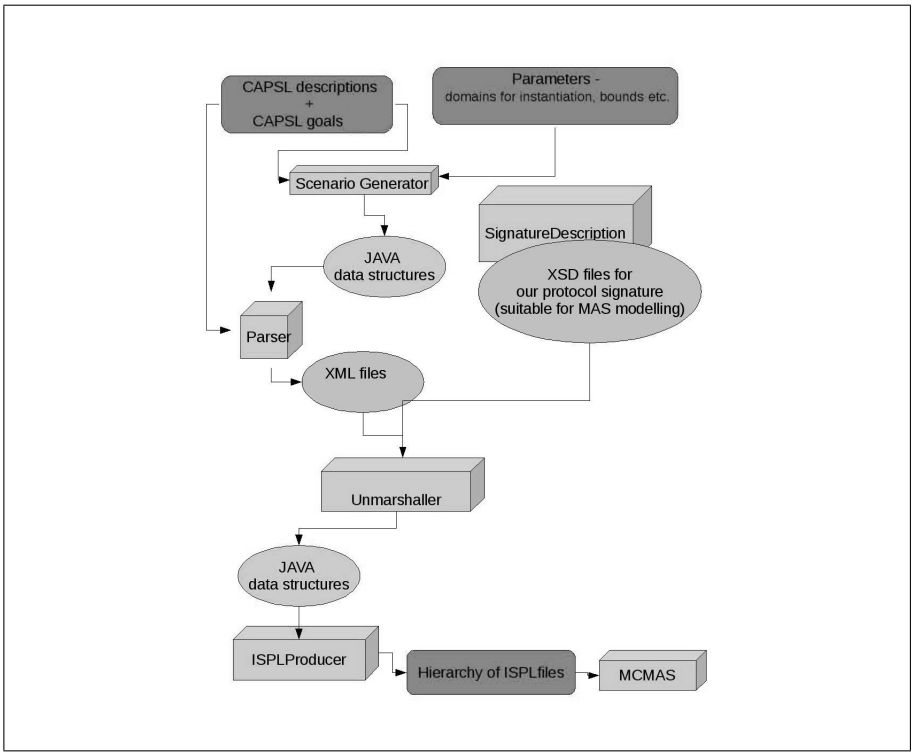

Figure 1. *PD2IS's compilation workflow.*

```
  Na=X and Step=0: {send_enc_alice_X_pubkey_bob};
  --A rule as above for each X in the nonce range {r1,r2,...}

  --Step 1:
  Step=1: {receive_enc_Na_Nb_pubkey_A};

  --Step 2:
  Na=X and Nb=Y and Step=2: {send_enc_X_Y_pubkey_bob};
  --A rule as above for all X,Y in the nonce range {r1,r2,...}

  --Step 3:
   Step=3: {donothing};
end Protocol

Evolution:
  --Step 0 and step 2:
  Step=Step+1
  if
  Action=send_X and Env.Action=intercept_X;
```

```
   --A rule as above for each message X

   --Step 1:
   Step=Step+1  and
   Nb=Y --<SET> assigns nonce Y to X
   if
   Action=receive_enc_Na_Nb_pubkey_A and
   Env.Action=transmit_enc_X_Y_pubkey_alice and
   Na=X; --<OUT_MATCH> checks the consistency of Na
   --A rule as above for all X,Y in the nonce range {r1,r2,...}

   --Step 3
   -- No update to local state
 end Evolution
```

□

The local variables of the Environment agent encode all the actions that the intruder can eavesdrop and execute as well as the messages and parts of messages deduced by him, i.e., the action history H, and the knowledge set X described in Section 3. The Environment's *protocol section* and *evolution section* encode the Dolev-Yao deductions described in in Section 3 by means of the $\vdash$ relation. We give below a simplified version of an ISPL code snippet for the Environment section in a generated NSPK scenario.

EXAMPLE 15. — A simplified fragment for the Environment agent in ISPL

```
Vars:
 knows_X:boolean;   -- Represents whether nonce X is in the knowledge set
 --A line as above for each X in the nonce range {r1,r2,...}
 ...
end Vars

Protocol:
 --Transmit actions enabled when nonce X is in the knowledge-set:
 knows_X: {transmit_enc_alice_X_pubkey_bob};
 --A rule as above for each X in the nonce range {r1,r2,...}
 ...
end Protocol

Evolution:
 --DY decomposition upon intercept of enc_A_Na_pubkey_B:
 knows_X = true
 if
 Action=intercept_enc_alice_X_pubkey_intruder and
 ag_A.Action=send_enc_alice_X_pubkey_intruder
 --A rule as above for each X in the nonce range {r1, r2, ...}
 --and for each A-agent ag_A
 ...
end Evolution
```

□

In summary, from CAPSL files the toolkit produces optimised ISPL code ready to be verified by MCMAS.

5.2. *Experimental Results*

To evaluate the tool PD2IS in a systematic way, we ran tests on protocol descriptions from the CAPSL-version of the Clark-Jacob's library (Clark *et al.*, 1999) and the SPORE library (Laboratoire Spécification et Vérification ENS Cachan, 2003). In addition to the atomic CAPSL goals considered by other tools, we added a number of complex CAPSL goals to each CAPSL input file. Specifically, we included complex authentication goals with up to two levels of nesting of knowledge operators (see Example 2 and Example 3). Table 1 reports the experimental results.

Table 1. *Experimental results*

Protocol	Learn	Attacks	Avg. time (secs)	Total time (secs)
ISO1PUCCF	off	none	1	23
	on	1	1	3
ISO2PUCCF	off	none	2	42
	on	2	2	6
ISOSK1PU	off	none	2	46
	on	1	1	2
ISOSK2PU	off	none	3	63
	on	2	2	8
ISOSK3PM	off	none	4	80
	on	4	3	13
AndrewRPC	-	2	4	60
NSPK	-	1	1	19
WideMouthFrog	-	1	7	56
KSL1	-	1	5	55
KSL2	-	1	10	170

The first column in the table specifies the protocol being checked; KSL1 and KSL2 stand for the protocols described in (Kehne *et al.*, 1992) and (Lowe, 1996) respectively. The second column indicates whether we assume that keys can be broken ("learned") by the intruder. The third reports the number of atomic CAPSL goals for which MCMAS found an attack. The forth gives the average verification time that MCMAS took to verify a single ISPL file while searching for possible attacks. Differently from other approaches, in this approach we systematically generate ISPL files corresponding to each protocol scenario considered. In this case we asked PD2IS to generate scenarios with up to 3 agents per protocol role and passed MCMAS the files one at a time

until either an attack on an atomic goal was found or all the generated files had been checked. The last column reports the total time used by PD2IS and MCMAS while performing these checks sequentially. Note that in principle the various ISPL files could be checked in parallel thereby reducing the total verification time.

Observe that the recorded verification times are not too dissimilar to those produced by leading toolkits. Indeed, verification results in the literature are most often reported only for one scenario for each given protocol. We believe the promising results are due to a combination of factors, including the underlying efficiency of MCMAS, the efficient translation optimised for immediate decoding by the receiver of messages with one level of encryption, and the generation of ISPL code in which variable types and ranges are optimised for MCMAS.

Lastly, while we do not employ full-typing, we assume some of the tagging schemes in (Heather *et al.*, 2003), which also help by limiting the size of the DY inference system.

Irrespective of the attractive verification results for atomic goals, as previously remarked, the methodology presented focuses on verifying specifications containing knowledge operators resulting from what are perceived as being natural epistemic translations of complex CAPSL goals. To illustrate this point, Table 2 presents the results for two NSPK scenarios checked against the epistemic CAPSL goals in Example 2 and Example 3. As reported in the table, the epistemic CAPSL goals both hold unless the scenario includes a "corrupt insider", i.e., unless the intruder initially knows the private key of some principal whom other principals trust. This is of course in line with our intuition. The two epistemic CAPSL goals considered are automatically compiled into temporal-epistemic specifications as described in Example 11 and Example 12 respectively.

Also note that adding further levels of nesting of knowledge modalities may falsify an initially true CAPSL goal. For example, the experiments confirm that in the case of the ISO1PUCCF protocol while the first-level epistemic goal from Table 2 holds, the second-level goal from Table 2 fails.

Table 2. *Verification results for NSPK*

Insider	$\texttt{Knows}\, B : \texttt{holds}\, A : \texttt{Na}$	$\texttt{Knows}\, A : \texttt{Knows}\, B : \texttt{holds}\, A : \texttt{Na}$
No	True	True
Yes	False	False

6. Conclusions and related work

In this paper we presented a methodology for verifying temporal-epistemic properties of cryptographic protocols. Specifically we defined a map from protocol scenarios into interpreted systems, a mainstream semantics for temporal-epistemic logic.

The translation has been implemented into the PD2IS toolkit that transforms a CAPSL protocol description into an ISPL program to be checked by MCMAS. In addition to translating the models, PD2IS also compiles CAPSL goals into a range of temporal-epistemic specifications corresponding to different readings of the goals. To evaluate the efficiency of the technique we tested several protocols in the Clark-Jacobs and SPORE libraries. The performance of the toolkit proved to be satisfactory. The objective in this article was not to analyse these specific protocols but to develop an automatic translation and test its implementation. We are not aware of another existing toolkit that enables the seamless verification of temporal-epistemic properties of any protocol described in a mainstream security protocol description language.

Any model checking approach to the verification of security protocols requires a careful handling of the state-explosion problem. The work described in this article is no exception and a number of assumptions have been made. Most notably we employed a "matching receive" semantics, we limited the analysis to a fixed number of interleaved sessions, and the messages generated have finite-length.

This works follows (Lomuscio *et al.*, 2008), where a framework for specifying security protocols as interpreted systems was described. (Lomuscio *et al.*, 2008) put forward LDYIS (Lazy Dolev-Yao Interpreted Systems) that included several abstraction features to reduce the state space. However (Lomuscio *et al.*, 2008) used a SAT-based bounded model checking method for the verification back-end. A further point of difference is that while LDYIS deals with authentication of origin here we focus on entity authentication; to achieve this we relax slightly the tight matched send-receive conditions prescribed in (Lomuscio *et al.*, 2008). In addition to the above while (Lomuscio *et al.*, 2008) focuses on the abstract semantics and deals with an ad-hoc example, here a map and a compiler is defined that in principle can be applied to any protocol description.

While we are not aware of other automatic translations into interpreted systems in the literature, epistemic logic has of course been put forward before as a specification language. Many of these approaches, starting from the well-known BAN logics line (Burrows *et al.*, 1990) either focus on the resulting expressivity (Syverson *et al.*, 2000) to model particular protocols, or integrate it within theorem provers. More recently the specific protocols have been analysed in an ad-hoc basis by means of temporal-epistemic model checkers (Meyden *et al.*, 2004a; Lomuscio *et al.*, 2007; Kacprzak *et al.*, 2008) with anonymity protocols such as Chaum's dining cryptographers (Chaum, 1988) receiving considerable attention (Kacprzak *et al.*, 2006; Meyden *et al.*, 2004b). These radically differ from the one presented here in that no general translation method is defined.

The technique presented only deals with a restricted version of CAPSL. For instance, we do not handle the case of all types of nesting within deeply encrypted messages. It would be of interest to extend it to support more complex protocols displaying this and other features. Also, while the compiler currently only supports CAPSL, it seems possible in principle to devise compilations from other protocol languages.

7. References

Abadi M., Tuttle M., "A semantics for a logic of authentication", *Proceedings of the 10th annual ACM symposium on Principles of distributed computing (PODC'91)*, ACM Press, New York, NY, USA, pp. 201–216, 1991.

Armando A., Basin D., Boichut Y., Chevalier Y., Compagna L., Cuellar J., Drielsma P., Héam P., Mantovani J., Moedersheim S., Oheimb D. v., Rusinowitch M., Santiago J., Turuani M., Viganò L., Vigneron L., "The AVISPA Tool for the automated validation of internet security protocols and applications", *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, vol. 3576 of *Lecture Notes in Computer Science*, Edinburgh, UK, pp. 281–285, 07, 2005.

AVISPA-project, "AVISS – Deliverable D.6.1: List of selected problems", http://www.avispa-project.org/delivs/6.1/d6-1.ps, 2005.

Basin D., Mödersheim S., Viganò L., "OFMC: A symbolic model checker for security protocols", *International Journal of Information Security*, vol. 4, num. 3, pp. 181–208, June, 2005.

Blanchet B., "Automatic Proof of Strong Secrecy for Security Protocols", *IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 86–101, 2004.

Boreale M., "Symbolic trace analysis of cryptographic protocols", *Proceedings of the 28th International Colloquium of Automata, Languages and Programming (ICALP'01)*, vol. 2076 of *Lecture Notes in Computer Science*, Springer, pp. 667–681, 2001.

Boureanu I., Cohen M., Lomuscio A., "Protocol Descriptions to Interpreted Systems, version 0.90", http://pd2is.sourceforge.net, 2009.

Burrows M., Abadi M., Needham R., A Logic of Authentication, Technical report, DEC–SRC, http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-39.html, 1990.

Chaum D., "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability", *Journal of Cryptology*, vol. 1, num. 1, pp. 65–75, 1988.

Clark J., Jacob J., "A Survey of Authentication Protocol Literature", http://citeseer.nj.nec.com/, 1997.

Clark J., Jacob J., "Clark-Jacobs library in CAPSL", http://www.csl.sri.com/projects/capsl/, 1999.

Clarke E., Grumberg O., Peled D., *Model Checking*, MIT Press, 1999.

Cohen M., Dam M., "A Complete axiomatization of Knowledge and Cryptography", *Proceedings of the 22nd Symposium on Logic in Computer Science (LICS'07)*, IEEE Computer Society Press, pp. 77–88, 2007.

Corin R., Etalle S., Saptawijaya A., "A Logic for Constraint-based Security Protocol Analysis", *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, IEEE Computer Society, Washington, DC, USA, pp. 155–168, 2006.

Denker G., Millen J., "CAPSL Integrated Protocol Environment", *In Proceedings of DARPA Information Survivability Conference (DISCEX'00)*, IEEE Computer Society, pp. 207–221, 2000.

Dolev D., Yao A., "On the Security of Public-Key Protocols", *IEEE Transactions on Information Theory*, vol. 29, pp. 198–208, 1983.

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

Halpern J., Pucella R., "Modeling Adversaries in a Logic for Security Protocol Analysis", *Proceedings of the Workshop on Formal Aspects of Security (FASec'02)*, vol. 2629 of *Lecture Notes in Computer Science*, Springer, pp. 115–132, 2002.

Halpern J. Y., Meyden R., "A logical reconstruction of SPKI", *Journal of Computer Security*, vol. 11, num. 4, pp. 581–613, 2004.

Heather J., Lowe G., Schneider S., "How to Prevent Type Flaw Attacks on Security Protocols", *Journal of Computer Security*, vol. 11, num. 2, pp. 217–244, 2003.

Kacprzak M., Lomuscio A., Niewiadomski A., Penczek W., Raimondi F., Szreter M., "Comparing BDD and SAT based techniques for model checking Chaum's Dining Cryptographers Protocol", *Fundamenta Informaticae*, vol. 72, num. 1-3, pp. 215–234, 2006.

Kacprzak M., Nabialek W., Niewiadomski A., Penczek W., Pólrola A., Szreter M., Woźna B., Zbrzezny A., "VerICS 2007 - a Model Checker for Knowledge and Real-Time", *Fundamenta Informaticae*, vol. 85, num. 1-4, pp. 313–328, 2008.

Kehne A., Schönwälder J., Langendörfer H., "A nonce-based protocol for multiple authentications", *SIGOPS Operating Systems Review*, vol. 26, num. 4, pp. 84–89, 1992.

Laboratoire Spécification et Vérification ENS Cachan, "SPORE: Security Protocols Open Repository", `http://www.lsv.ens-cachan.fr/spore/index.html`, 2003.

Lomuscio A., Penczek W., "LDYIS: a framework for model checking security protocols", *Fundamenta Informaticae*, vol. 85, pp. 359–375, 2008.

Lomuscio A., Qu H., Raimondi F., "MCMAS: A model checker for multi-agent systems", *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, vol. 5643 of *Lecture Notes in Computer Science*, Springer, pp. 682–688, 2009.

Lomuscio A., Raimondi F., Wozna B., "Verification of the Tesla protocol in MCMAS-X", *Fundamenta Informaticae*, vol. 79, num. 3-4, pp. 473–486, july, 2007.

Lowe G., "Some new attacks upon security protocols", *Proceedings of the 9th IEEE workshop on Computer Security Foundations (CSFW'96)*, IEEE Computer Society, pp. 162–169, 1996.

Lowe G., "Casper: A Compiler for the Analysis of Security Protocols", *Journal of Computer Security*, vol. 6, num. 1-2, pp. 53–84, 1998.

Meadows C., "The NRL Protocol Analyzer: An Overview", *Journal of Logic Programming*, vol. 26, pp. 113–131, 1996.

Meyden R., Gammie P., "MCK: Model Checking the Logic of Knowledge", *Proceedings of 16th International Conference on Computer Aided Verification (CAV'04)*, vol. 3114 of *Lecture Notes in Computer Science*, Springer, pp. 479–483, 2004a.

Meyden R., Su K., "Symbolic Model Checking the Knowledge of the Dining Cryptographers", *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, IEEE Computer Society, Washington, DC, USA, pp. 280–291, 2004b.

Millen J., "Common Authentification Protocol Specification Language", http://www.csl.sri.com/projects/capsl/, 2001.

Needham R., Schroeder M., "Using Encryption for Authentication in Large Networks of Computers", *Communications of the ACM*, vol. 21, pp. 993–999, 1978.

Oheimb D. v., "The High-Level Protocol Specification Language HLPSL developed in the EU project AVISPA", *Proceedings of International Summer School On Applied Semantics (APPSEM'05)*, 2005.

Parikh R., Ramanujam R., "Distributed Processes and the Logic of Knowledge", *Logic of Programs*, pp. 256–268, 1985.

Paulson L., "Proving Security Protocols Correct", *Proceeding of the IEEE Symposium on Logic in Computer Science (LICS'99)*, IEEE Computer Society Press, 1999, pp. 370–383, 1999.

Pucella R., Weissman V., "A Logic for Reasoning about Digital Rights", *Proceedings of the 15th Computer Security Foundations Workshop (CSFW'02)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 282–294, 2002.

Ramanujam R., Suresh S., "A decidable subclass of unbounded security protocols", *Proceedings of the Workshop on Issues in the Theory of Security (WITS'03)*, IOS Press, pp. 11–20, 2003a.

Ramanujam R., Suresh S., "Undecidability of secrecy for security protocols", *manuscript*, 2003b.

Rusinowitch M., Turuani M., "Protocol Insecurity with Finite Number of Sessions is NP-complete", *Proceedings of the 14th IEEE Workshop on Computer Security Foundations (CSFW'01)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 174–187, 2001.

Syverson P., Cervesato I., "The Logic of Authentication Protocols", *Proceedings of the International School on Foundations of Security Analysis and Design (FOSAD'00)*, vol. 2170 of *Lecture Notes in Computer Science*, Springer, pp. 63–136, 2000.

Syverson P., Stubblebine S., "Group Principals and the Formalization of Anonymity", *Proceedings of the Wold Congress on Formal Methods (FM'99) in the Development of Computing Systems-Volume I*, Springer, London, UK, pp. 814–833, 1999.