Fundamenta Informaticae 85 (2008) 1–17
IOS Press

LDYIS: a Framework for Model Checking Security Protocols

Alessio Lomuscio
Department of Computing
Imperial College London, UK
A.Lomuscio@doc.ic.ac.uk

Wojciech Penczek[*]
Institute of Computer Science, PAS, and
University of Podlasie, Poland
penczek@ipipan.waw.pl

Abstract. We present a formalism for the automatic verification of security protocols based on multi-agent systems semantics. We give the syntax and semantics of a temporal-epistemic security-specialised logic and provide a lazy-intruder model for the protocol rules that we argue to be particularly suitable for verification purposes. We exemplify the technique by finding a (known) bug in the traditional NSPK protocol.

1. Introduction

In protocol analysis it is significantly important to be able to capture the concepts of what information a participant *has* throughout an exchange, what a participant can and cannot *deduce*, and whether or not particular sequences of moves exist resulting in the system *reaching* a particular state. Indeed, in some specialised areas of security (such as all the ones rooted in the BAN proposal [3]), knowledge of the participants is explicitly and symbolically represented. Of course, the area of Artificial Intelligence has a long and successful tradition in the development of formal tools for the representation of knowledge, including the formalisation of the temporal evolution of agents' epistemic states [8] as well as further refinements for security [10, 9]. Crucially, recent developments in the verification of some of these logics by means of symbolic model checking techniques [20, 18, 13], as well as the implementation of these tools in prototype systems [16, 6, 13] have provided the area with a set of tools to attempt the analysis of security protocols by means of efficient and automatic techniques.

[*]Address for correspondence: Institute of Computer Science, PAS ul. Ordona 21, 01-237 Warsaw, Poland

In our own work in this area we have successfully verified by means of our own specialised model checkers [14, 15] the correctness of the dining cryptographer protocol [5] and the TESLA protocol [15] in terms of appropriate specifications expressed as temporal/epistemic formulas. However in doing so we have also found that our analysis cannot be extended to deal with many complex security protocols. In particular if we were to consider a protocol in which the intruder is allowed to operate in line with the full Dolev-Yao model [7] we would quickly have to consider a number of states/transitions higher than what any model checker could ever handle. For example, at any step of any security protocol a principal could theoretically compose and send an unbounded number of messages to all other principals, thereby causing the state space to diverge.

This limitation is not related to the knowledge-based approach we pursued but applies just as well to more traditional model checking approaches. Indeed, while model checking approaches in security are typically concerned with checking reachability properties only [11, 12] (and not temporal/epistemic specifications) the same considerations apply here too. One of the most promising approaches to tackle this problem is the lazy intruder model developed by Basin, Mödersheim and Viganó in [4] and (to the best of our understanding) used together with a planning model checker [2] as a part of the IST Project AVISPA [1].

Here, in the representation of the runs that may take place in the system, messages may be routed to the channel *only when* both the sender and the receiver are in a state in which the protocol permits this message to be sent and received. In this way the model checker implementing this semantics does not have to consider transitions clearly not leading to protocol termination. Furthermore, in the construction of a run many details are abstracted and left to the model checker's unifying mechanisms to instantiate.

In this paper we set out to define a semantics for temporal and epistemic logic based on ideas similar to the ones cited above. We aim to introduce a lazy intruder model, integrate it with a temporal/epistemic logic and pair this with a highly-efficient bounded model checking algorithm. While our approach is directly inspired by the lazy-intruder work cited above, there are also considerable differences resulting from our objective to work on a fully-fledged specification language involving temporal/epistemic operators as opposed to simply checking reachability of states.

The scheme of the rest of the paper is as follows: in Section 2 and 3 we give a semantics to our approach. In Section 4 we define the logic and satisfaction for the language. Section 5 covers the basic bounded model checking set up in the present case. In Section 6 we exemplify the analysis in the case of a particular authentication protocol (NSPK). In Section 7 we show how our formalisation would produce an attack to NSPK.

2. Semantics

Since our intention is to bring together model checking with protocol analysis to check explicitly what epistemic properties participants have (i.e., what information they possess) we work on an extension of the framework of interpreted systems [8]. Interpreted systems are a transition-based semantics where (global) states represent explicitly a snapshot of all components (or *agents*) in the system. Transitions between states represent the result of global joint actions performed simultaneously (in a locked semantics) by all agents at a given global states. The agents select actions to perform following a given local protocol mapping sets of actions for each given local state. In our adaptation of the formalism, the agents and the intruder are the principals in the protocol, their actions are simply communication actions of send and receive (of a given message) and the protocols are explicit representations of the steps of the security protocol under analysis[1].

As discussed in the previous section the key idea of this approach is to employ a symbolic, trace-based semantics for the analysis of security protocols. We use the term "symbolic" to mean a compact, variable-based representation; for instance, a symbolic computational trace is a sequence in which some elements are variables or terms, and therefore represent a set of traces. In contrast to this we employ the term "constant trace" to refer simply to a ground instance of a symbolic trace, i.e., a sequence of concrete computational states. Given the importance in this approach of unification during model checking, the distinction between variables and ground terms is one that we employ throughout the paper for a variety of concepts. More details on this are offered below.

We begin by assuming a finite set of agents, or *principals*, Ag including a special agent called the intruder $\iota \in \mathrm{Ag}$. Note that the principals are *ground* elements and *uniquely* correspond to real entities, not to be confused with the roles they play; so, for instance, if an intruder is impersonating a principal we only need to use one principal, the intruder, in our model. To each principal $i \in \mathrm{Ag}$ we associate a number of security specialised concepts: an ordered set of fresh nonces $\mathcal{N}_i^f$ and old (in the sense of "used" or "seen") set of nonces $\mathcal{N}_i^o$, a set of keys $\mathcal{K}_i$ known to the agent, an index id_i indicating how many parallel sessions i are running, and an address $@_i$ (in the sense of origin/destination for the messages). Of key importance in the following is that in denoting an element of any of these sets we may use constant or variable terms denoting respectively a particular element of the set or a variable representing *any* element of the set. For clarity we use a lowercase letters to denote constant terms and uppercase letters to denote variable terms. For instance, n_a represents a constant nonce related to the constant principal a, n_A represents a constant nonce related to a variable agent A, N_a represents a variable nonce related to a constant agent a, and, N_A represents a variable nonce related to a variable agent A. Similarly for keys, k_a is a constant key for a constant principal a, K_a is a variable key for the constant principal a, and K_A is a variable key for the variable agent A. Ultimately we build traces of global states in which each global state is a tuple of local states for the principals. The local states contain all the information the principals have been exposed to, i.e., the messages they have witnessed and sent; in epistemic logic terminology we say we assume perfect recall.

Each message is represented by a tuple specifying origin, destination, and content. We formally proceed as follows.

Definition 2.1. (Messages)
A message msg is defined by the following grammar:

$$msg ::= i \mid I \mid n \mid N \mid k \mid K \mid (msg)_k \mid (msg)_K \mid msg \cdot msg,$$

where $i \in \mathrm{Ag}$ ($I \in \mathrm{Ag}$) is a constant (respectively, variable) principal, $n \in \mathcal{N}^f \cup \mathcal{N}^o$ ($N \in \mathcal{N}^f \cup \mathcal{N}^o$) is a constant (respectively, variable) nonce, and $k \in \mathcal{K}$ ($K \in \mathcal{K}$) is a constant (respectively, variable) key. The symbol $\cdot$ denotes the concatenation between messages.

Messages represent the *content* that is being exchanged. We use *letters* to represent the content, the sender, and the receiver of a message. Due to possible impersonations by Intruder we use the *address of a participant not the participant himself* in the fields of sender/receiver.

[1]Note the different use of the term "protocol" in interpreted systems semantics and as in "security protocol".

Definition 2.2. (Letters)
A *letter* is a tuple $\mathrm{lt} = ((@_s, @_r), msg)$ where $@_s$ is the sender's address, $@_r$ is the receiver's address, and msg is the content of the letter lt. We call $(@_s, @_r)$ the *header* of $\mathrm{lt} = ((@_s, @_r), msg)$.

The above defines a constant letter. Like for any other component in the framework we may need to use variable letters as well. To do this and retain the structure of the letter we simply use variables appropriately in any of letter's terms. For instance $((@_A, @_b), n_A)$ represents a (variable) letter referring to a message from a variable sender A to a constant principal b in which the content is a variable nonce that depends on the value of the sender.

We are now ready to give definitions for the global states of a system. The global states are tuples of local states, which represent the states of a computation principals may be in.

Definition 2.3. (Local states)
A local state for an agent $i \in Ag$ is a 6-tuple $l_i = (\mathrm{Ag}_i, \mathcal{N}_i^o, \mathcal{N}_i^f, \mathcal{K}_i, id_i, \mathrm{lt}_i)$ where

- $\mathrm{Ag}_i \subseteq Ag$ is a set of agents known to i,
- $\mathcal{N}_i^o$ is an ordered set of nonces that have been seen by agent i,
- $\mathcal{N}_i^f$ is an ordered set of fresh nonces available to agent i,
- $\mathcal{K}_i$ is a set of keys known to agent i,
- id_i is the number of sessions either completed or currently running in which i has participated,
- $\mathrm{lt}_i \subset (\mathrm{lt}, id)^+$ is a sequence of pairs of letters and sessions identifiers for the protocols sessions the agent has actively participated in. Each nonce in $\mathcal{N}_i^o$ is present in lt_i.

We will use L_i to denote a set of the possible local states for agent i, and $G \subseteq \Pi_{i=1}^n L_i$ for the set of all possible global states. We will also exploit the operator $First$ such that $First\,(\mathcal{N})$ returns the first element of a non-empty ordered set $\mathcal{N}$ and modifies $\mathcal{N}$ by removing this element.

Definition 2.4. A global state $g = (l_1, \ldots, l_n)$ is a n-tuple of local states for all agents under consideration. An initial global state is a tuple $g^0 = (l_1, \ldots, l_n)$, where $l_i = (\mathrm{Ag}_i, \emptyset, \mathcal{N}_i^f, \mathcal{K}_i, 0, \epsilon)$, for all $i = 1, \ldots n$ with the assumption that $\bigcap_{i=1}^n \mathcal{N}_i^f = \emptyset$ (i.e., the sets of fresh nonces are disjoint).

We assume each agent i performs send/receive actions Act_i according to a *protocol*, i.e., a function $L_i \to 2^{Act_i}$ from local states to actions Act_i ($\epsilon \in Act_i$ is the empty action). We assume all agents perform their actions synchronously at a given global state; so we have transitions of the form $T \subseteq G \times Act_1 \times \cdots \times Act_n \times G$, where we assume agents non-deterministically choose an action at any step from the set of actions offered to them by the protocol. We write $(g, g') \in T$ if $(g, (a_1, \ldots, a_n), g') \in T$ for some $(a_1, \ldots, a_n)$. The states, the actions, and the transitions as above define a branching time semantics. A *path* $\pi = (g_0, g_1, \ldots, g_j)$ is a sequence of global states such that $(g_i, g_{i+1}) \in T$ for each $0 \leq i < j$. 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 $g \in G$. A global state g is called *reachable* from g^0 if there is a path $\pi = (g_0, g_1, \ldots)$ such that $g_0 = g^0$ and $g_i = g$ for some $i \geq 0$.

3. Lazy D-Y Interpreted Systems

The previous section is quite liberal in terms of what traces we allow in a system. In this section we introduce constraints in the executions to model a particular set of assumptions known as Dolev-Yao (D-Y, for short) [7] assumptions. Specifically we assume all participants have perfect recall and that the intruder has complete control of the channel, i.e., it can block/resend/route messages on the communication channel. We also assume that encryption is perfect, i.e., encrypted messages may only be decrypted with the correct key and encryption/decryption of messages is instantaneous.

We now formalise the assumptions above by restricting the possible transitions, thereby defining Lazy Dolev-Yao Interpreted Systems (*LDYIS* for short). While LDYIS model the whole class of D-Y protocols, each particular security protocol will define specific rules specifying the sequence of messages to be sent/received. We use the term "lazy" in the sense of [4] (see below).

To specify any protocol we give *state transformer rules*. These are rules that specify preconditions and postconditions on global states for a particular step in the protocol. For efficiency reasons (further discussed in the next section), a state transformer rule is given in a compact form specifying *sets* of possible transitions in a protocol. Given this, variables specifying particular components in the local states will generally appear in these rules. Following the "lazy" approach in LDYIS for any rule to be triggered we need both the sender and the receiver to be in the appropriate local state corresponding to a particular protocol step. In this way a considerable number of irrelevant transitions (i.e., messages that would be discarded by the receiver) is saved thereby increasing the efficiency of the model checking method applied. More precisely our state transformer rules are defined as follows.

Definition 3.1. (State transformer rules)
For each step t of a protocol under analysis, we consider state transformer rules $G \xrightarrow{t} G'$ of the form $(pre(t), post(t))$, where G, G' are sets of global states, $pre(t)$ are preconditions on G, and $post(t)$ are postconditions on G'.

The preconditions are constraints that must be satisfied for the transition to be enabled; the postconditions specify updates to the local states occurring as a result of the triggering of the transition. Given that we use a lazy semantics $pre(t)$ always specifies *matched moves between Sender and Receiver*, i.e., the sender only sends messages to receivers who are ready to execute the corresponding step in the protocol.

In the preconditions we often write $c \in L_A$ to denote that the component c is an element of each of the local states for the variable agent A. Similarly, in the postconditions we write $L'_A = L_A \circ c$ to denote the update of the set of local states for the variable agent A by means of a component c. Typically c is a letter, a nonce, a key, or a session identifier and the test or the update is intended to be carried out on the relevant subcomponent of the local states; we do not write this explicitly to simplify the reading of the rules.

We further assume that after every move (send/receive), the instantaneous decoding of all messages sent is executed (provided a key is in possession of the intruder and/or principals). Clearly, it is possible to generate state transformer rules in an automatic way, but this requires a syntactical analysis of the protocol steps. We describe here the main idea of the algorithm and present a detailed case study analysis of NSPK, based on this algorithm, in Section 6. For each protocol the state transformer rules can be generated automatically according to the following principles. An "honest send" rule represents a message being sent from A to B. A "fake send" rule corresponds to a message sent by the intruder to B. An "ι-forward" rule corresponds to the forwarding by the intruder of a message, previously intercepted, to another principal. For the above, bearing in mind the semantics chosen assumes synchronous moves, we obtain the following transitions. We give further details of this in Section 6.

- Honest-send-i-$A \longrightarrow B$
  - Preconditions:
    If $i = 1$, then A has not yet sent a message of step 1 to B.
    If $i \geq 2$, then A has received a message from B of step $i-1$ and has not yet replied to B.
  - Postconditions:
    The local states of A and ι (B if $A = \iota$) are updated according to the message sent by A. If $B = \iota$, then $id_B := id_B + 1$. If $i = 1$, then $id_A := id_A + 1$.
- Fake-send-i-$\iota(A) \longrightarrow B$
  - Preconditions:
    A message of step i is composable by ι and acceptable by B, i.e., B has sent a message of step $i-1$ (if $i \geq 2$) to ι and has not received a reply.
  - Postconditions:
    The local states of ι and B are updated according to the message sent by ι. If $i = 1$, then $id_B := id_B + 1$.
- ι-forward (step i: $A \longrightarrow B$)
  - Preconditions:
    A message of step i sent by A was intercepted by ι and not yet received by B.
  - Postconditions:
    The local state of B is updated according to the message intercepted by ι.

The rules given in Section 7 can be produced automatically by a compiler. However in the example discussed in Section 7 the specific rules are computed by hand simply from the D-Y assumptions and the protocol description.

Definition 3.2. (Lazy D-Y Interpreted Systems)
Given a security protocol P and a set of propositional variables PV. An LDYIS M_P for P, or simply a model (for P), is a $(n+4)$-tuple $M_P = (G, g^0, \mathcal{P}, \sim_1, \ldots, \sim_n, V)$, where:

- $g^0 \in \Pi_{i=1}^n L_i$ is the initial global state of the system,
- G is the set of global states reachable from g^0,
- $\mathcal{P} = \bigcup_{g \in G} \mathcal{P}(g)$, where $\mathcal{P}(g) \subset \Pi(g)$ is the set of all paths starting at g compliant with the Lazy D-Y conditions above,
- $\sim_i \subseteq G \times G$ is an epistemic relation 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,
- $V : G \times PV \rightarrow \{true, false\}$ is an interpretation for the propositional variables PV.

The structure above satisfies also the following conditions:

- Agents have perfect recall: following receipt of a message agents add the message to their local state by pairing it with an appropriate session identifier.
- Every message sent by a principal is intercepted by the intruder, who records it in its local state.
- Upon receipt of messages all principals and intruders immediately decode all messages and sub-messages providing they have the key to do so.

We do not give the conditions above formally as they are rather intuitive and will be presented in the example below. It is clear that giving the conditions is not technically difficult although it is rather cumbersome.

Intuitively M_P will be used to interpret a logic defined in the next section. Also note the relations $\sim_i$ are epistemic accessibility relations between states to be used to interpret an epistemic language as defined in the next section.

4. Temporal Logic of Knowledge

In this section we introduce a logical language to be interpreted on the semantics of the previous section. The language we use is a standard combination of epistemic logic and branching time temporal logic. Extensions are possible and worth considering but not pursued here.

Definition 4.1. (Logical Language)
The logical language $\mathcal{L}$ is defined by the following BNF expression:

$$\phi ::= sends_i(msg) \,|\, receives_i(msg) \,|\, has_i(k) \,|\, has_i(n) \,|\, \neg\phi \,|\, \phi \wedge \phi \,|\, \overline{\mathrm{K}}_i\phi \,|\, EX\phi \,|\, E(\phi U \phi) \,|\, EG\phi,$$

where $sends_i(msg)$, $receives_i(msg)$, $has_i(k)$, $has_i(n) \in PV$, msg is a message, $k \in \mathcal{K}_i$ is a key, $n \in \mathcal{N}_i^o$ is a nonce, $i \in \{1, \ldots, n\}$, and PV a set of propositional variables.

The language above includes specialised propositional letters of the obvious meaning, negation, conjunction, branching time operators (EX, EU, EG) and epistemic operators ($\overline{\mathrm{K}}_i$). $\overline{\mathrm{K}}_i\phi = \neg K_i \neg\phi$ where $K_i\phi$ is read as "Agent i knows that ϕ". We use the dual $\overline{\mathrm{K}}_i$ as the model checking technique presented below is based on bounded model checking. We interpret $\mathcal{L}$ on LDYISs as follows.

Definition 4.2. (Satisfaction)
Let M be a model, $g = (l_1, \ldots, l_n)$ a global state, and ϕ, ψ formulas in $\mathcal{L}$. The satisfaction relation $\models$, denoting truth of a formula in the model M^2 at the global state g, is defined inductively as follows:

- $g \models sends_i(msg)$ iff $(((@_i, @_j), msg), id)$ is an element of the sequence lt_i in the local state l_i in g, for some address $@_j$ and session number id,
- $g \models receives_i(msg)$ iff $(((@_j, @_i), msg), id)$ is an element of the sequence lt_i in the local state l_i in g, for some address $@_j$ and session number id,

[2] M is omitted when understood.

- $g \models has_i(n)$ iff $n \in \mathcal{N}_i^o$, $\quad g \models has_i(k)$ iff $k \in \mathcal{K}_i$,
- $g \models \neg\phi$ iff not $g \models \phi$, $\quad g \models \phi \wedge \psi$ iff $g \models \phi$ and $g \models \psi$,
- $g \models \overline{\mathrm{K}}_i\phi$ iff $(\exists g' \in G)\ g \sim_i g'$ and $g' \models \phi$, $\quad g \models EX\phi$ iff $(\exists \pi \in \mathcal{P}(g))$ s.t. $\pi(1) \models \phi$,
- $g \models EG\phi$ iff $(\exists \pi \in \mathcal{P}(g))$ s.t. $(\forall k \geq 0)\ \pi(k) \models \phi$,
- $g \models E(\phi U \psi)$ iff $(\exists \pi \in \mathcal{P}(g))\ (\exists k \geq 0)$ s.t. $\pi(k) \models \psi$ and $(\forall 0 \leq j < k)\ \pi(j) \models \phi$.

Note that the special propositions are interpreted according to their intuitive meaning on their respective logical states and temporal and epistemic operators are as standard.

5. Bounded Model Checking for $\mathcal{L}$

In this section we adapt an algorithm for bounded model checking (BMC) for $\mathcal{L}$. BMC works by translating both the model and the formula to be checked into propositional formulas. The satisfaction of their conjunction is then checked by an efficient SAT-solver. BMC is particularly efficient when the analysis involves looking for faults in protocols whose runs are finite and key properties are expressed as formulas in the existential form.

BMC was originally introduced for verification of the existential fragment of the logic CTL [19], and then extended to ECTLK [18]. BMC is based on the observation that some properties of a system can be checked over a part of its model only. We present the main definitions of BMC for $\mathcal{L}$, but refer the reader to the literature cited above for more details. In order to restrict the semantics to a part of the model we define k-models, where the paths of $\mathcal{P}$ are replaced by their prefixes of length k.

Model checking over models can be reduced to model checking over k-models. The main idea of BMC for $\mathcal{L}$ is that we can check φ over M_k by checking the satisfiability of the propositional formula $[M, \varphi]_k := [M^{\varphi, g^0}]_k \wedge [\varphi]_{M_k}$, where the first conjunct represents (a part of) the model under consideration and the second a number of constraints that must be satisfied on M_k for φ to be satisfied. Once this translation is defined, checking satisfiability of an $\mathcal{L}$ formula can be done by means of a SAT-checker.

We provide here some details of the translation. We begin with the encoding of the transitions in the system under consideration. We assume $L_i \subseteq \{0,1\}^{k_i}$, where $k_i = \lceil \log_2(|L_i|) \rceil$ and we take $k_1 + \ldots + k_n = m$. Moreover, let Ix_i be an $<$-ordered set of the indices of the bits of the local states of each participant i of the global states, i.e., $Ix_1 = \{1, \ldots, k_1\}, \ldots, Ix_n = \{m - k_n + 1, \ldots, m\}$. Then, each global state $g = (l_1, \ldots, l_n)$ can be represented by $w = (w[1], \ldots, w[m])$ (which we shall call a *global state variable*), where each $w[i]$ for $i = 1, \ldots, m$ is a propositional variable. A sequence $w_{0,j}, \ldots, w_{k,j}$ of global state variables is called the j-th symbolic k-path. The propositional formula $[M^{\varphi, g^0}]_k$, representing the k-paths in the k-model, is defined as follows:

$$[M^{\varphi, g^0}]_k := I_{g^0}(w_{0,0}) \wedge \bigwedge_{j=1}^{f_k(\varphi)} \bigwedge_{i=0}^{k-1} T(w_{i,j}, w_{i+1,j}),$$

where $w_{0,0}$ and $w_{i,j}$ for $0 \leq i \leq k$ and $1 \leq j \leq f_k(\varphi)$ are global state variables, and $T(w_{i,j}, w_{i+1,j})$ is a formula encoding the transition relation T. $[M^{\varphi, g^0}]_k$ encodes the initial state g^0 by $w_{0,0}$ and constrains the $f_k(\varphi)$[3] symbolic k-paths to be valid k-paths in M_k.

[3]The function f_k determines the number of k-paths sufficient for checking an $\mathcal{L}$ formula, see [18] for more details.

The next step of the algorithm consists in encoding φ by a propositional formula. Let w, v be global state variables. We use the following propositional formulas: $p(w)$ encodes a proposition p of $\mathcal{L}$, $H(w, v)$ represents logical equivalence between global state encodings (i.e., representing the same global state), $HK_i(w, v)$ represents logical equivalence between i-local state encodings (i.e., representing the same i-local state), $L_{k,j}(l)$ encodes a backward loop connecting the k-th state to the l-th state in the j-th symbolic $k-$path j, for $0 \leq l \leq k$. The translation of φ at state $w_{m,n}$ into the propositional formula $[\varphi]_k^{[m,n]}$ is as follows:
$[p]_k^{[m,n]} := p(w_{m,n})$, for $p \in PV$,
$[\overline{\mathrm{K}}_l\alpha]_k^{[m,n]} := \bigvee_{i=1}^{f_k(\varphi)} \Big(I_{g^0}(w_{0,i}) \ \wedge\ \bigvee_{j=0}^{k} \big([\alpha]_k^{[j,i]} \ \wedge\ HK_l(w_{m,n}, w_{j,i})\big)\Big)$,
$[\mathrm{EX}\alpha]_k^{[m,n]} := \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge\ [\alpha]_k^{[1,i]}\Big)$,
$[\mathrm{EG}\alpha]_k^{[m,n]} := \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge\ (\bigvee_{l=0}^{k} L_{k,i}(l)) \ \wedge\ \bigwedge_{j=0}^{k} [\alpha]_k^{[j,i]}\Big)$,
$[\mathrm{E}(\alpha \mathrm{U} \beta)]_k^{[m,n]} := \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge\ \bigvee_{j=0}^{k} \big([\beta]_k^{[j,i]} \ \wedge\ \bigwedge_{t=0}^{j-1} [\alpha]_k^{[t,i]}\big)\Big)$.
Given the translations above, we can now check φ over M_k by checking the satisfiability of the propositional formula $[M^{\varphi,g^0}]_k \wedge [\varphi]_{M_k}$, where $[\varphi]_{M_k} = [\varphi]_k^{[0,0]}$. The translation above is shown in [18] to be correct and complete. Given that $\mathcal{L}$ is a propositional temporal epistemic language in which the propositions' interpretation depends on the global states only these results apply to $\mathcal{L}$ as well.

6. Needham Schroeder Public-Key Protocol (NSPK)

The approach above is general and provides an abstract framework for the analysis of protocols. We now instantiate the framework by a case study analysis of NSPK [3] by introducing specific NSPK rules. The NSPK protocol is defined by the following three steps:

1 $A \longrightarrow B$: $\{A, N_A\}_{K_B}$

2 $B \longrightarrow A$: $\{N_A, N_B\}_{K_A}$

3 $A \longrightarrow B$: $\{N_B\}_{K_B}$

In the first step A (Initiator) sends to B (Responder) his identity A and a fresh nonce N_A, both encrypted with B's public key K_B. B responds to A with the nonce N_A and a fresh nonce N_B, both encrypted with A's public key K_A. In the third step, A sends back to B the nonce N_B encrypted with B's public key K_B.

Recall that we assume Intruder ι to have full control of the channel. It can stop all messages, and can route messages on the network with any header and with any content that it is able to produce by composing, decrypting, and encrypting messages with keys known to it.

Session identifiers are local to the participants. When starting a new session (or receiving the first message of a new session) each participant increases his session identifier by one and records the message sent together with the header and the new session number. When a participant sends (or receives) another message, we record it in its local state together with the header and the corresponding session number. When Intruder intercepts a message sent in the first step of the protocol, this is recorded with the original session identifier. At any other step the intruder checks his history to use the correct session identifier.

We start by describing the transition rules representing A sending a message to B, as in step 1 of the protocol. We define two rules for each step plus one rule, which is applied to all the steps. For step 1 we have: one rule for an honest send from A to B, one rule for a fake send from $\iota(A)$ to B (impersonation of A by the intruder), and one rule for an ι-forward to B (forward message from the intruder to B). By $N_A = First\left(\mathcal{N}_A^f\right)$ we mean that for each principal $p \in \{a, b\}$ playing the role of A we have $N_p = First\left(\mathcal{N}_p^f\right)$. In all the rules below we assume that $A \neq B$.

Definition 6.1. (Rule T_1: honest-send-1-($A \longrightarrow B$))
Preconditions: $(((@_A, @_B), (A, N_A)_{k_B}), Id_A) \notin L_A$,
Postconditions: If $A \neq \iota$, then $L'_A = L_A \circ (((@_A, @_B), (A, N_A)_{k_B}), Id_A{+}1) \circ \{N_A\} \circ \{Id_A{+}1\}$,
$L'_\iota = L_\iota \circ (((@_A, @_B), (A, N_A)_{k_B}), Id_A + 1)$ if $B \neq \iota$, and
$L'_\iota = L_\iota \circ (((@_A, @_B), (A, N_A)_{k_B}), Id_\iota + 1) \circ \{N_A\} \circ \{Id_\iota + 1\}$ if $B = \iota$, where $N_A = First\left(\mathcal{N}_A^f\right)$.
If $A = \iota$, then $L'_A = L_A \circ (((@_A, @_B), (A, N_A)_{k_B}), Id_A{+}1) \circ \{N_A\} \circ \{Id_A{+}1\}$,
$L'_B = L_B \circ (((@_A, @_B), (A, N_A)_{k_B}), Id_B{+}1) \circ \{N_A\} \circ \{Id_B{+}1\}$, where $N_A \in \{First\left(\mathcal{N}_\iota^f\right)\} \cup \mathcal{N}_\iota^o$.

By $L_A \circ c$ we denote the update of L_A defined by c. The result of the update consists in the following change of the local state of A: $(((@_A, @_B), (A, N_A)_{k_B}), Id_A + 1)$ is added to the sequence lt_A in the local state of A, the nonce $\{N_A\}$ is added to the set of old nonces of A (i.e., to $\mathcal{N}_A^o$), and the session number of A is increased by 1 for $c = Id_A + 1$. Similar considerations apply to $L_\iota \circ c$. Since this is a symbolic rule, in order for it to be executed it requires unification of all the variables present. Notice that L_B does not change because the letter sent by A is intercepted by the intruder (so L_ι changes) and only later possibly forwarded to B (this is later described by the rule ι-forwards in Definition 6.7).

Note also the rule above covers several cases including a, ι sending to b, as well as b, ι sending to a, and a, b sending to ι. Notice that if $B \neq \iota$, the session number of ι does not change because it represents only the number of sessions initiated or participated in by it (not intercepting messages).

The next rule encodes a fake send from $\iota(A) \text{to} B$ in step 1.

Definition 6.2. (Rule T_2: fake-send-1-($\iota(A) \longrightarrow B$))
Preconditions: $(((@_\iota, @_B), (A, N_\iota)_{k_B}), Id_B) \notin L_B$,
Postconditions: $L'_B = L_B \circ (((@_\iota, @_B), (A, N_A)_{k_B}), Id_B{+}1) \circ \{N_A\} \circ \{Id_B{+}1\}$,
$L'_\iota = L_\iota \circ (((@_\iota, @_B), (A, N_\iota)_{k_B}), Id_\iota{+}1) \circ \{N_\iota\} \circ \{Id_\iota{+}1\}$, where $N_\iota \in \{First\left(\mathcal{N}_\iota^f\right)\} \cup \mathcal{N}_\iota^o$,
$N_A = N_\iota$, and $A \neq \iota$.

The above models a situation in which Intruder, impersonating A, sends a message to B. In doing so it uses any nonce N_ι, either freshly generated or old. The message is directly delivered to B, thereby updating B's local state. Notice that the intruder ι is initiating the session with B, so the session number changes for him as well as for B. The next rule is for an honest send from $B \text{to} A$ in step 2.

Definition 6.3. (Rule T_3: honest-send-2-($B \longrightarrow A$))
Preconditions: If $B \neq \iota$, then $(((@_A, @_B), (A', N_A)_{k_B}), Id_B) \in L_B$,
$(((@_B, @_A), (N_A, N_B)_{k_{A'}}), Id_B) \notin L_B$.
If $B = \iota$, then $(((@_A, @_B), (A, N_A)_{k_B}), Id_B) \in L_B$, $(((@_B, @_A), (N_A, N_B)_{k_A}), Id_B) \notin L_B$.
Postconditions: If $B \neq \iota$, then $L'_B = L_B \circ (((@_B, @_A), (N_A, N_B)_{k_{A'}}), Id_B) \circ \{N_B\}$,
$L'_\iota = L_\iota \circ (((@_B, @_A), (N_A, N_B)_{k_{A'}}), Id_A)$, where $N_B = First\left(\mathcal{N}_B^f\right)$.

If $B = \iota$, then $L'_B = L_B \circ (((@_B, @_A), (N_A, N_B)_{k_A}), Id_B) \circ \{N_B\}$,
$L'_A = L_A \circ (((@_B, @_A), (N_A, N_B)_{k_A}), Id_A) \circ \{N_B\}$, where $N_B \in \{First\left(\mathcal{N}_B^f\right)\} \cup \mathcal{N}_B^o$ or
$\{N_A, N_B\}_{k_A} \in L_B$, and $L'_B = L_B \circ \{N_B\}$ if $N_B = First\left(\mathcal{N}_B^f\right)$.

This rule is split into two parts, each governing whether or not B represents Intruder. When $B \neq \iota$ the rule describes two possibilities, i.e., B replying to an honest send from A or to a fake send from $\iota(A)$. In both cases only L_B and L_ι change as the message is intercepted by Intruder and only later possibly forwarded to A. If B replies to an honest send, then $A' = A$, otherwise $A = \iota$ and A' could be the name of any participant Intruder is impersonating. The conditions $(((@_A, @_B), (A', N_A)_{k_B}), Id_B) \in L_B$ and $(((@_B, @_A), (N_A, N_B)_{k_{A'}}), Id_B) \notin L_B$ guarantee that B has received the message from A according to the first step of the protocol and has not yet sent a reply to A. When $B = \iota$ the rule describes the case where B is replying to an honest send from A. The next rule is for a fake send from $\iota(B)$ to A in step 2.

Definition 6.4. (Rule T_4: fake-send-2-($\iota(B) \longrightarrow A$))
Preconditions: $(((@_A, @_\iota), (A, N_A)_{k_B}), Id_\iota) \in L_\iota$, $(((@_A, @_\iota), (A, N_A)_{k_B}), Id_A) \in L_A$,
$(((@_\iota, @_A), (N_A, N_\iota)_{k_A}), Id_\iota) \notin L_\iota$, and $(((@_\iota, @_A), (N_A, N_\iota)_{k_A}), Id_A) \notin L_A$,
Postconditions: $L'_A = L_A \circ (((@_\iota, @_A), (N_A, N_B)_{k_A}), Id_A) \circ \{N_B\}$,
$L'_\iota = L_\iota \circ (((@_\iota, @_A), (N_A, N_\iota)_{k_A}), Id_\iota) \circ \{N_\iota\}$,
where $N_B = N_\iota$, ($N_\iota \in \{First\left(\mathcal{N}_\iota^f\right)\} \cup \mathcal{N}_\iota^o$ and $N_A \in \mathcal{N}_\iota^o$) or $\{N_A, N_\iota\}_{k_A} \in L_\iota$.

The above rule codes the situation where Intruder, impersonating B, sends a message to A. To do so it replays the nonce N_A generated before by A and any nonce N_ι. Alternatively, ι can send any other message $\{N_A, N_\iota\}_{k_A}$ known to him (without knowing the encrypted nonces). The next rule is for an honest send from A to B in step 3.

Definition 6.5. (Rule T_5: honest-send-3-($A \longrightarrow B$))
Preconditions: If $A \neq \iota$, then $(((@_B, @_A), (N_A, N_B)_{k_A}), Id_A) \in L_A$,
$(((@_A, @_B), (N_B)_{k_{B'}}), Id_A) \notin L_A$.
If $A = \iota$, then $(((@_B, @_A), (N_A, N_B)_{k_A}), Id_A) \in L_A$, $(((@_A, @_B), (N_B)_{k_B}), Id_A) \notin L_A$,
$(((@_B, @_A), (N_A, N_B)_{k_A}), Id_B) \in L_B$.
Postconditions: If $A \neq \iota$, then $L'_A = L_A \circ (((@_A, @_B), (N_B)_{k_{B'}}), Id_A)$,
$L'_\iota = L_\iota \circ (((@_A, @_B), (N_B)_{k_{B'}}), Id_A)$. If $A = \iota$, then $L'_A = L_A \circ (((@_A, @_B), (N_B)_{k_B}), Id_A)$,
$L'_B = L_B \circ (((@_A, @_B), (N_B)_{k_B}), Id_B)$, where $N_B \in \mathcal{N}_A^o$.

Similarly to Definition 6.1 (but note that the corresponding pre-/post-conditions and messages are different) this rule is split into two parts, depending on whether $A \neq \iota$ or $A = \iota$. In the first case, two possibilities are covered: in the first, A is replying to an honest send from B, in the second to a fake send from $\iota(B)$. Then, only L_A and L_ι are changed as the message is intercepted by Intruder and only later possibly forwarded to B. If A replies to an honest send, then $B' = B$, otherwise $B = \iota$ and B' could be the name of any participant Intruder impersonates. The conditions $(((@_B, @_A), (N_A, N_B)_{k_A}), Id_A) \in L_A$, $(((@_A, @_B), (N_B)_{k_B}), Id_A) \notin L_A$ guarantee that A has received the message from B according to step 2 and has not yet sent a reply to B. When $A = \iota$ the rule describes the case where A is replying to an honest send from B. The last condition in the preconditions says that B has sent the message $(N_A, N_B)_{k_A}$ in step 2.

The next rule is for a fake send from $\iota(A)\,\text{to}\,B$ in step 3.

Definition 6.6. (Rule T_6: fake-send-3-($\iota(A) \longrightarrow B$))
Preconditions: $(((@_B, @_\iota), (N_A, N_B)_{k_A}), Id_\iota) \in L_\iota$, $(((@_B, @_\iota), (N_A, N_B)_{k_A}), Id_B) \in L_B$, $(((@_\iota, @_B), (N_B)_{k_B}), Id_B) \notin L_B$, and $(((@_\iota, @_B), (N_B)_{k_B}), Id_\iota) \notin L_\iota$.
Postconditions: $L'_B = L_B \circ (((@_\iota, @_B), (N_B)_{k_B}), Id_B)$, $L'_\iota = L_\iota \circ (((@_\iota, @_B), (N_B)_{k_B}), Id_\iota)$.

In the above rule the intruder impersonating A sends a message to B consisting of a nonce N_B encrypted with the key k_B. This message must be composable by Intruder, i.e., N_B has to be in the set $\{First\left(\mathcal{N}_\iota^f\right)\} \cup \mathcal{N}_\iota^o$. Moreover, the message $(N_B)_{k_B}$ must be acceptable by B; so N_B must have previously been sent from B to A and the reply has not yet been received by B. The above is represented by the following condition: $(((@_B, @_\iota), (N_A, N_B)_{k_A}), Id_B) \in L_B$. To avoid to represent repeated sending of the same messages by Intruder, the last condition is also imposed.

Definition 6.7. (ι-forwards (steps 1-3))
Step 1: Preconditions: $((@_A, @_B), (A, N_A)_{K_B}, Id_1) \in L_\iota$, $((@_A, @_B), (A, N_A)_{K_B}, Id_2) \notin L_B$,
Postconditions: $L'_B = L_B \circ ((@_A, @_B), (A, N_A)_{K_B}, Id_B + 1) \circ \{Id_B + 1\} \circ \{N_A\}$

Step 2: Preconditions: $((@_A, @_B), (A, N_A)_{K_B}, Id_1) \in L_A$; $((@_B, @_A), (N_A, N_B)_{K_A}, Id_1) \notin L_A$; $((@_B, @_A), (N_A, N_B)_{K_A}, Id_2) \in L_\iota$.
Postconditions: $L'_A = L_A \circ ((@_B, @_A), (N_A, N_B)_{K_A}, Id_1) \circ \{N_B\}$.

Step 3: Preconditions: $((@_B, @_A), (N_A, N_B)_{K_A}, Id_1) \in L_A$; $((@_A, @_B), (N_B)_{K_B}, Id_1) \notin L_B$; $((@_A, @_B), (N_B)_{K_B}, Id_2) \in L_\iota$.
Postconditions: $L'_B = L_B \circ ((@_A, @_B), (N_B)_{K_B}, Id_1)$.

To conclude the encoding of the D-Y intruder we use the above ι-forward rules to represent Intruder forwarding messages it has previously intercepted. At each step, the precondition specifies the local states of Sender and Intruder at which a forward can take place. Notice that in the above rules, nonces do not need to have the indexes that unify, i.e., $N_A = n_a$ and $A = b$ is a valid unification.

7. An Attack on NSPK found with BMC

We now use the rules of the previous section to show how a previously known attack on NSPK may efficiently be found when the system runs are explored by means of the BMC method of Section 5. We consider 3 agents (2 participants a and b communicating in the presence of an intruder ι). We begin our run at an initial global state $g^0 = (l_a^0, l_b^0, l_\iota^0)$, where $l_j^0 = (\{a, b, \iota\}, \emptyset, \mathcal{N}_j^f, \{k_a, k_b, k_\iota, k_j^{-1}\}, 0, \epsilon)$, for $j \in \{a, b, \iota\}$. We assume to begin the run with a initiating an NSPK exchange with ι believing ι is an honest participant.

1.1 $honest-send-1-a \longrightarrow \iota$. Definition 6.1 applies, where $A = a$, $B = \iota$, and $N_A = n_a$. The resulting updates are computed: $l'_a = l_a \circ ((@_a, @_\iota), (a, n_a)_{k_\iota}, 1) \circ \{n_a\} \circ \{1\}$, $l'_\iota = l_\iota \circ ((@_a, @_\iota), (a, n_a)_{k_\iota}, 1) \circ \{n_a\} \circ \{1\}$, where $n_a \in \mathcal{N}_a'^s$, $n_a = First\left(\mathcal{N}_a^f\right)$. ι performs the corresponding decoding moves, it extracts the nonces, it decomposes the messages, etc. (as in every turn below). ι can now use the message it has received to start a (fresh) second parallel session with b (this is called "impersonating a" by some authors) .

2.1 $fake - send - 1 - \iota(a) \longrightarrow b$. Definition 6.2 applies, where $A = a$, $B = b$, and $N_\iota = n_a$. As a result, b thinks a's address is $@_\iota$. The following updates are computed:
$l'_b = l_b \circ ((@_\iota, @_b), (a, n_a)_{k_b}, 1) \circ \{n_a\} \circ \{1\}$,
$l'_\iota = l_\iota \circ ((@_\iota, @_b), (a, n_a)_{k_b}, 2) \circ \{2\}$.
As a result of this message b responds to ι.

2.2 $honest - send - 2 - b \longrightarrow \iota(a)$ - by means of Definition 6.3 applies, where $A = \iota$, $B = b$, $N_A = n_a$, $N_B = n_b$, $Id_B = 1$, and $A' = a$. $l'_b = l_b \circ ((@_b, @_\iota), (n_a, n_b)_{k_a}, 1) \circ \{n_b\}$, $l'_\iota = l_\iota \circ ((@_b, @_\iota), (n_a, n_b)_{k_a}, 2)$, where $n_b = First\left(\mathcal{N}_b^f\right)$.
The intruder can now simply replay the message received from b to show his credentials to a.

1.2 $honest - send - 2 - \iota \longrightarrow a$ - Definition 6.4 applies, where $B = \iota$, $A = a$, $N_a = n_a$, $N_b = n_b$, $Id_B = 1$, and $A' = a$. $l'_a = l_a \circ ((@_\iota, @_a), (n_a, n_b)_{k_a}, 1) \circ \{n_b\}$, $l'_\iota = l_\iota \circ ((@_\iota, @_a), (n_a, n_b)_{k_a}, 2)$.
ι has successfully impersonated b in its run with a.
Then, a concludes the exchange by:

1.3 $honest - send - 3 - a \longrightarrow \iota$ - Definition 6.5 applies, where $A = a$, $B = \iota$, $N_B = n_b$, $B' = \iota$, $A' = a$, and $Id_A = 1$. $l'_a = l_a \circ ((@_a, @_\iota), (n_b)_{k_\iota}, 1)$, $l'_\iota = l_\iota \circ ((@_a, @_\iota), (n_b)_{k_\iota}, 1)$.
Intruder ι is now in the position to authenticate himself to b by replaying the message.

2.3 $fake - send - 3 - \iota(a) \longrightarrow b$ - Definition 6.6 applies, where $B = b$ and $N_B = n_b$. $l'_b = l_b \circ ((@_\iota, @_b), (n_b)_{k_b}, 1)$, $l'_\iota = l_\iota \circ ((@_\iota, @_b), (n_b)_{k_b}, 2)$,
The above two interleaved sessions define the following execution: $g_0 \xrightarrow{1.1} g_1 \xrightarrow{2.1} g_2 \xrightarrow{2.2} g_3 \xrightarrow{1.2} g_4 \xrightarrow{1.3} g_5 \xrightarrow{2.3} g_6$.

We now aim to show that the run above does not satisfy an intuitive specification in the logic $\mathcal{L}$. We can represent one of the correctness criteria in the authentication protocol by using the following condition: if b completes an execution started by a using nonce n_b, then b and a know that n_b is a secret shared by a and b only. In particular n_b is unknown to the intruder ι; note that, by Dolev-Yao assumptions, a, b are aware an intruder is operating on the channel. This condition can be expressed by the formula $\varphi = AG((has_a(n_b) \wedge has_b(n_a) \wedge sends_a((n_b)_{k_b}) \wedge receives_b((n_b)_{k_b}) \Rightarrow (K_b(\neg has_\iota(n_b)) \wedge K_a(\neg has_\iota(n_b)))$.

Clearly the specification above is not satisfied in the model. In fact it is easy to see that the run we have produced before satisfies the negation of the formula above:
$EF(has_a(n_b) \wedge has_b(n_a) \wedge sends_a((n_b)_{k_b}) \wedge receives_b((n_b)_{k_b}) \wedge (\overline{\mathrm{K}}_b(has_\iota(n_b)) \vee \overline{\mathrm{K}}_a(has_\iota(n_b)))$.

7.1. A translation for BMC

In this section we exemplify how a bounded model checker implementing the lazy approach above would have found the counterexample. We model executions for the following parameters: M - the number of sessions, N - the number of participants including the intruder ι (notice that $1, \ldots, N-1$ are the constant principals while N denotes ι). We show a general encoding of global states, the initial state, and the rule T_1 only. Next, for $M = 2$ and $N = 3$, we show the encoding of φ and some of its constituents propositional variables. To begin with we represent a local state of a participant i by the following vector of vectors of propositional variables $w^i = (w_1^i, w_2^i, w_3^i, w_4^i, w_5^i, w_6^i)$, in which the components are the encodings of the following parameters:

- w_1^i encodes the agents known to i (of length $N\lceil \log_2(N) \rceil$),

- w_2^i encodes the nonces seen by i (of length $2M\lceil\log_2(2MN)\rceil$),
- w_3^i encodes the fresh nonces of i (of length $M\lceil\log_2(NM)\rceil$),
- w_4^i encodes the keys known to i (of length $2N\lceil\log_2(2N)\rceil$),
- w_5^i encodes the number of sessions run by i (of length $\lceil\log_2(M)\rceil$),
- w_6^i encodes the sequence of (lt, id) (of length $3M \times L_D$), where $L_D = 2\lceil\log_2(N)\rceil + \lceil\log_2(2MN + N)\rceil + \lceil\log_2(2MN)\rceil + \lceil\log_2(2N)\rceil + \lceil\log_2(M)\rceil$.

In the following we assume that for each agent i, address $@_i$, nonce n_i, key k_i, and session number id_i we have a corresponding Boolean representation (encoding) $[i]$, $[@_i]$, $[n_i]$, $[k_i]$, and $[id_i]$. This is totally unproblematic and can be done in similar fashion by means of Boolean variables.

Let $w = (w^{1,0}, \ldots, w^{N,0})$ be a global state variable, where $w^{i,0} = (w_1^{i,0}, \ldots, w_6^{i,0})$ represents a local state for agent i. For a vector of propositional variables $w = (w_1, \ldots, w_n)$ by $(i_1, \ldots, i_m)(w)$, where $m \leq n$ and $i_j \in \{0,1\}$, we mean the formula $\bigwedge_{j=1}^{m} b(i_j, w_j) \wedge \bigwedge_{j=m+1}^{n} \neg w_j$ with $b(1, w_j) = w_j$ and $b(0, w_j) = \neg w_j$. The initial state state $g^0 = (l_1^0, \ldots, l_N^0)$ is encoded by $I_{g^0}(w) = \bigwedge_{i=1}^{N} I_{l_i^0}(w^{i,0})$, where $I_{l_i^0}(w^{i,0}) = [1] \cdot \ldots \cdot [N](w_1^{i,0}) \wedge (0)w_2^{i,0} \wedge [n_1^1] \cdot \ldots \cdot [n_i^{2M}](w_3^{i,0})) \wedge [k_1] \cdot \ldots [k_N] \cdot [k_i^{-1}](w_4^{i,0}) \wedge (0)w_5^{i,0} \wedge (0)w_6^{i,0}$. By $[x] \cdot [y]$ we mean the concatenation of the binary encodings of x and y.

The encoding of $T(w, w')$ is equal to $\bigvee_{i=1}^{6} [T_i(w, w')]$, where $[T_i(w, w')]$ is the propositional encoding of the rule T_i.

To generate the Boolean translation representing all the moves in an execution we need to encode into propositional logic each rule T_i from the previous section. The propositional encoding is cumbersome, although, of course, the aim of the method is for these to be computed automatically. This is in line with intermediate representations for SAT-based model checking inputs (often in the tens of thousands of variables).

We report below an encoding of the postconditions of T_1 for the case where $A \neq \iota$ (the rest can be worked out similarly) simply to show that this can be obtained even by hand, albeit laboriously. The postcondition for T_1 is equal to:
$\bigvee_{i=1}^{N-1} \Big(\bigvee_{j \notin \{i,N\}} ((w_6'^i = w_6^i \circ [((@_i, @_j), (i, (dec(First\,(w_3^i))_{k_j}), dec(w_5^i) + 1)]) \wedge (w_2'^i = w_2^i \circ First\,(w_3^i)) \wedge (w_5'^i = [dec(w_5^i) + 1]) \wedge \bigwedge_{l \in \{1,4\}} (w_l'^i \equiv w_l^i)) \wedge (w_6'^N = w_6^N \circ [((@_i, @_j), (i, (dec(First\,(w_3^i))_{k_j}), dec(w_5^i) + 1)])) \wedge \bigwedge_{l \in \{1,2,4,5\}} (w_l'^N \equiv w_l^N) \wedge \bigwedge_{l \notin \{i,N\}} (w'^l \equiv w^l)) \Big) \bigvee (w_6'^i = w_6^i \circ [((@_i, @_N), (i, (dec(First\,(w_3^i))_{k_N}), dec(w_5^i) + 1)]) \wedge (w_2'^i = w_2^i \circ First\,(w_3^i)) \wedge (w_5'^i = dec(w_5^i) + 1) \wedge \bigwedge_{l \in \{1,4\}} (w_l'^i \equiv w_l^i)) \wedge (w_6'^N = w_6^N \circ [((@_i, @_N), (i, (dec(First\,(w_3^i))_{k_N}), dec(w_5^N) + 1)]) \wedge (w_2'^N = w_2^N \circ First\,(w_3^i)) \wedge (w_5'^N = dec(w_5^N) + 1)) \wedge \bigwedge_{l \in \{1,4\}} (w_l'^N \equiv w_l^N) \wedge \bigwedge_{l \notin \{i,N\}} (w'^l \equiv w^l)),$
where $w_6^i \circ [(lt, id)]$ denotes w_6^i extended with the encoding of (lt, id), i.e., $[lt], [id]$; $dec(w_j^i)$ denotes the value encoded by w_j^i, $w_2^i \circ First\,(w_3^i)$ denotes w_2^i extended with the encoding of the first nonce of w_3^i and at the same time removing that nonce from the encoding w_3^i, and $w' \equiv w$ encodes the equivalence of the corresponding propositions of w and w'.

Encoding for agents ids, nonces and keys can be easily obtained. In fact assume that for $N = 3$ ($a = 1$, $b = 2$, $\iota = 3$), and $M = 2$, we consider the following: $[a] = [@_a] = (0,1)$, $[b] = [@_b] = (1,0)$, $[\iota] = [@_\iota] = (1,1)$, $[n_a] = (0,0,1)$, $[n_a'] = (0,1,0)$, $[n_b] = (0,1,1)$, $[n_b'] = (1,0,0)$, $[n_\iota] = (1,1,0)$, $[n_\iota'] = (1,1,1)$, $[k_a] = (0,0,1)$, $[k_a^{-1}] = (1,1,0)$, $[k_b] = (0,1,0)$, $[k_b^{-1}] = (1,0,1)$, $[k_\iota] = (1,0,0)$, $[k_\iota^{-1}] = (0,1,1)$, as id_i is a number, we take simply its binary encoding.

With the above we can encode k-models exactly in the same way any bounded model checker would do.
To encode the formulas to be checked, let $w = (w^a, w^b, w^\iota)$ be a global state variable. The encoding of the propositional variables $has_a(n_b)$, $has_b(n_a)$, and $has_\iota(n_b)$ is as follows:

$has_a(n_b)(w) = (\neg w^a_{2,1} \wedge w^a_{2,2} \wedge w^a_{2,3}) \vee (\neg w^a_{2,4} \wedge w^a_{2,5} \wedge w^a_{2,6})$,
$has_b(n_a)(w) = (\neg w^b_{2,1} \wedge \neg w^b_{2,2} \wedge w^b_{2,3}) \vee (\neg w^a_{2,4} \wedge \neg w^a_{2,5} \wedge w^a_{2,6})$,
$has_\iota(n_b)(w) = (\neg w^\iota_{2,1} \wedge w^\iota_{2,2} \wedge w^\iota_{2,3}) \vee (\neg w^\iota_{2,4} \wedge w^\iota_{2,5} \wedge w^\iota_{2,6})$.

The encoding of $sends_a((n_b)_{k_b})$ and $receives_b((n_b)_{k_b})$ is similar.
To encode the formula $\neg\varphi = EF((has_a(n_b) \wedge has_b(n_a)) \wedge (\overline{\mathrm{K}}_b(has_\iota(n_b)) \vee \overline{\mathrm{K}}_a(has_\iota(n_b)))$ we need 3 symbolic paths as $f_6(\varphi) = 3$. Let $\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3}$ be three symbolic paths.

$[EF(has_a(n_b) \wedge has_b(n_a)) \wedge sends_a((n_b)_{k_b}) \wedge receives_b((n_b)_{k_b}) \wedge (\overline{\mathrm{K}}_b(has_\iota(n_b)) \vee \overline{\mathrm{K}}_a(has_\iota(n_b)))]^{0,0}_6 :=$

$\bigvee_{i=1}^{3} \Big(H(w_{0,0}, w_{0,i}) \wedge \bigvee_{j=0}^{6}(has_a(n_b)(w_{j,i}) \wedge has_b(n_a)(w_{j,i}) \wedge sends_a((n_b)_{k_b})(w_{j,i}) \wedge$
$receives_b((n_b)_{k_b})(w_{j,i}) \wedge \bigvee_{m=1}^{3}\Big(I_{g^0}(w_{0,m}) \wedge \bigvee_{n=0}^{6}\big(has_\iota(n_b)(w_{n,m}) \wedge HK_b(w_{j,i}, w_{n,m})\big)\Big) \vee$
$\bigvee_{m=1}^{3}\Big(I_{g^0}(w_{0,m}) \wedge \bigvee_{n=0}^{6}\big(has_\iota(n_b)(w_{n,m}) \wedge HK_a(w_{j,i}, w_{n,m})\big)\Big)\Big)$

The translations exemplified above could be fed to a SAT-solver thereby returning satisfaction for the conjunction of the specification formula considered on the sub-run shown.

8. Conclusions

In this paper we have made three contributions. Firstly, we have taken inspiration from the ideas of the lazy-intruder model [4] to define LDYIS, a MAS based semantics for security protocols. Secondly, we have formalised a general approach to transition rules that generate LDYIS runs on which a temporal-epistemic logic can be interpreted. Thirdly, we have proposed a semantics (LDYIS) that is immediately ready to be model checked by means of any SAT-based methods such as bounded model checking. The formalism presented in this paper differs from the one pursued in the Avispa project in that it uses MAS inspired semantics and a fully-fledged temporal/epistemic language to check protocol specifications (as opposed to reachability only). Technically, the approaches hardly resemble each another as the semantics is rather different. We find the expressive power of temporal/epistemic specifications appropriate for security protocols, particularly to express anonymity. The approach presented in this paper can also be seen as an attempt to limit the state explosion in the verification of security protocols. We are currently working on an implementation of this technique to evaluate it experimentally; however, formal considerations point to efficiency savings over non-lazy approaches such as [17].

Acknowledgements: The authors are grateful to Ioana Boureanu, Jonathan Ezekiel, and Maciej Szreter for several suggestions on an earlier version of this paper. Work on this project was supported by an International joint project award to both authors from the Royal society. The first author acknowledges additional support from EPSRC (EP/E035655/1). The second author acknowledges additional support from the Ministry of Science and Higher Education (3 T11C 011 28).

References

[1] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganó, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proc* of CAV'05, *LNCS* 3576: 281–285, 2005.

[2] A. Armando and L. Compagna. An optimized intruder model for SAT-based model-checking of security protocols. *ENTCS*, 125(1):91–108, 2005.

[3] M. Burrows, M. Abadi, R. Needham. A Logic of Authentication, ACM Trans. Comput. Syst. 8(1): 18–36, 1990.

[4] D. A. Basin, S. Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[5] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[6] P. Dembiński, A. Janowska, P. Janowski, W. Penczek, A. Pólrola, M. Szreter, B. Woźna, and A. Zbrzezny. VerICS: A tool for verifying Timed Automata and Estelle specifications. In *Proc. of TACAS'03*, volume 2619 of *LNCS*, 278–283. Springer-Verlag, 2003.

[7] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. Inf. Theory*, 29(2):198–208, 1983.

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

[9] J. Halpern, R. van der Meyden, and R. Pucella. Revisiting the foundations of authentication logics.

[10] J. Y. Halpern and R. Pucella. Modeling adversaries in a logic for security protocol analysis. In *Proc. FASec'02)*, volume 2629 of *LNCS*, pages 115–132. Springer-Verlag, 2003.

[11] G. Jakubowska and W. Penczek. Modelling and checking timed authentication of security protocols. In *Fundamenta Informaticae*, 79(3-4):363–378, 2007.

[12] M. Kurkowski, W. Penczek, and A. Zbrzezny. Sat-based verification of security protocols via translation to networks of automata. In *MoChart IV*, volume 4428 of *LNAI*, pages 146–165. Springer-Verlag, 2007.

[13] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *CAV'04*, volume 3114 of *LNCS*, 479–483. Springer-Verlag, 2004.

[14] 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*, Vol 72(1-3), 215–234, 2006.

[15] A. Lomuscio, F. Raimondi, and B. Woźna. Verification of the tesla protocol in mcmas-x. In *Proceedings of Concurrency, Specification & Programming (CS&P)*, Germany, 2006. Humboldt University Press.

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

[17] A. Lomuscio and B. Woźna. A complete and decidable security-specialised logic and its application to the TESLA protocol. In *Proc. of AAMAS'06*, ACM Press, 145–152, 2006.

[18] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. *Fundamenta Informaticae*, 55(2):167–185, 2003.

[19] W. Penczek, B. Woźna, and A. Zbrzezny. Bounded model checking for the universal fragment of CTL. *Fundamenta Informaticae*, 51(1-2):135–156, 2002.

[20] 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.

[21] R. van der Meyden and Kaile Su. Symbolic model checking the knowledge of the dining cryptographers. In *Proc. CSFW'04*, 280–291, USA, 2004. IEEE Computer Society.