TPMC: A Model Checker For Time–Sensitive Security Protocols

Massimo Benerecetti, Dept. of Physical Sciences
Nicola Cuomo, Dept. of Mathematics
Adriano Peron, Dept. of Physical Sciences
Università di Napoli "Federico II", Napoli, Italy
bene@na.infn.it ncuomo@na.infn.it peron@na.infn.it

Abstract— **In this paper we face the problem of verifying security protocols where temporal aspects explicitly appear in the description. In previous work, we proposed *Timed HLPSL*, an extension of the specification language HLPSL (originally developed in the Avispa Project), where quantitative temporal aspects of security protocols can be specified. In this work we present a model checking tool for the analysis of security protocols which employs THLPSL as a specification language and UPPAAL as the model checking engine. To illustrate how our framework applies, we also provide a specification of the Wide Mouthed Frog protocol and show some experimental results on a number of security protocols.**

I. Introduction

Much work has been devoted to formal specification and analysis of cryptographic protocols, leading to a number of different approaches and encouraging results (e.g. see [18]). Most of the proposed protocol specification languages and verification techniques are limited to cryptographic protocols where quantitative temporal information is not crucial (e.g. delay, timeout, timed disclosure or expiration of information do not affect the correctness of the protocol), and details about some low level timing aspects of the protocol are abstracted away (e.g. timestamps, duration of channel delivery etc). In this context, the specification language HLPSL has been proposed within the Avispa Project (see [1]), for the specification of industrial-strength security protocols. HLPSL allows for modular specifications, specification of control flow patterns, data-structures, and security properties. It is also sufficiently high-level to be used by protocol engineers.

In this paper we focus on the problem of specifying and verifying security protocols where temporal aspects directly affect the correctness of the protocol, and, therefore, need to be explicitly considered both in the specification and the verification. Examples of time sensitive protocols are, for instance, the non-repudiation Zhou-Gollmann protocol [19], the TESLA authentication protocol [15] and the well known Wide Mouthed Frog protocol [4].

The formal framework generally employed to model temporal features, in the context of finite state machines, is that of Timed Automata [3], and the corresponding *model checking* verification techniques are supported by a variety of tools. The model checker KRONOS [9], developed at VERIMAG, supports model checking of branching time requirements. The UPPAAL toolkit [17] allows for checking safety and bounded liveness properties. However, Timed Automata cannot be employed by protocol designer as a specification formalism in itself, being a rather low level formalism, lacking the ability of expressing parallelism and synchronization on structured messages built over cryptographic primitives. In a previous paper [5] we proposed Timed HLPSL (THLPSL for short), as temporal extension of the specification language HLPSL. The temporal feature introduced in THLPSL are: (a) temporal constraints of the control flow (the usual delays and timeouts associated with performing a transition) with respect to the occurrence of some event, (b) duration of a transition, (c) temporal constraints on the availability and usability of messages (message disclosure and expiration time) with respect to the occurrence of some event, and (d) delay in channel delivery. The semantics of THLPSL is formally defined in [5] by a mapping onto eXtended Timed Automata [7] (XTAs for short), the variant of timed automata employed by UPPAAL which, differently from the basic model of Timed Automata, allows for an explicit representation of concurrency and communication in the form of handshaking.

In this paper we describe the TPMC (Timed Protocols Model Checker) tool we developed for the analysis of timed security protocols. TPMC employs THLPSL as a specification language and UPPAAL as the model checking engine. The analysis of a protocol in TPMC consists in a translation of its THLPSL specification into the input language of UPPAAL according to the semantics presented in [5]. For the sake of ease of definition, such a semantics maps THLPSL specification onto pure XTAs, without exploiting the full expressive power of the UPPAAL language, which allows for shared integers variables, and integer and boolean arrays. The use of this additional features allows for exponentially more succinct UPPAAL specifications. As a consequence,

Proceedings 21st European Conference on Modelling and Simulation
Ivan Zelinka, Zuzana Oplatková, Alessandra Orsoni ©ECMS 2007
ISBN 978-0-9553018-2-7 / ISBN 978-0-9553018-3-4 (CD)

the mapping implemented in TPMC is not the one described in the formal semantics, but a semantically equivalent one which, taking advantage of the full expressivity of UPPAAL *XTA*s, can be more efficiently employed for implementation purposes. We also provide an experimental evaluation of the tool on a number of security protocols.

The idea of using Timed Automata for specifying real time systems and proving security properties is not new (e.g., see [6], [10], [12], [16]). Our approach differs from [6], [12] in that Timed Automata are not the specification language itself, but the back-end of an high level specification language.

The rest of the paper is organized as follows: in the next section we informally present the specification language THLPSL; in Section III we provide the specification the Wide-Mouthed protocol in THLPSL. In Section IV we describe the translation of THLPSL specifications into UPPAAL XTAs. Finally we conclude giving some experimental evaluation of our tool in Section V.

II. Timed HLPSL

In this section we informally describe the main features of the specification language THLPSL, a timed extension of the specification language HLPSL [1] and give an intuition of its semantics. A formal definition of the THLPSL syntax and the semantics, which is given in terms of *XTA*s [7], can be found in [5].

A strong limitation of the original HLPSL is that it does not allow for explicit specification of temporal aspects such as delays, timeouts, timing constraints on the validity of messages, etc., therefore making it unsuited to specify protocols where temporal aspects may affect correctness. THLPSL extends HLPSL by allowing for expressing the following temporal aspects:
a) temporal constraints on the control flow of participants to a protocol session;
b) duration of a transition, expressed as lower and upper bounds on its duration;
c) temporal constraints on the availability and usability of messages (message disclosure and expiration time);
d) duration of channel delivery, expressed as lower and upper bounds on the channel delay.

THLPSL allows for structured definitions of protocols. A protocol specification consists in the description of a set of roles. It is possible to distinguish between two kinds of roles: *basic roles* which describe the behavior of a participant to a protocol; *composition roles* that compose in parallel instances of basic roles (one for each participant to the protocol sessions) instantiating their parameters with constants.

Roles can be parameterized (with the obvious exception of the main role) and can exploit local variables defined within a local declaration section. Declared variables can be initialized by means an *initialization predicate*. Local variables, formal parameters and constants are typed. THLPSL supports various kinds of types. Common built-in types are the following: `agent` type (for agent names); `channel` type (for communication channel names); `public_key` and `symmetric_key` type (for public and symmetric keys used by cryptographic primitives); `text` type (for text messages); `nat` type (for natural numbers); `function` type (for hash functions). Type `text` may have additional attributes enclosed within brackets. For instance, the type `text(fresh)` is the type for freshly generated nonces.

The type `channel` may have additional attributes enclosed within brackets, e.g., `C:channel(dy,lb,ub)` specifies a channel controlled by a Dolev-Yao (DY) intruder [8] with minimum transmission delay `lb` (a rational number in $\mathcal{Q}_{\geq\prime}$) and maximum transmission delay `up` (a rational number in $\mathcal{Q}_{\geq\prime} \cup \{\infty\}$);

THLSPL provides some form of flexibility in the specification of the constraints on delay/timeout and message disclosure/expiration, by allowing to express these constraints with respect to the occurrence of a transition executed by a participant in the protocol. To this purpose, THLPSL is equipped with a variable type `role_instance` for role instances, which can only be used for formal parameters of roles (and not for the declaration of local variables). Intuitively, a formal parameter `RI` of type `role_instance` will be instantiated with a number between 1 and n in the definition of the main composition role, where exactly n roles are composed in parallel. Therefore, if `RI` is instantiated with number i, then it refers to the i-th role instance in the parallel composition. This allows for expressing time constraints relative to occurrences of events (referred to by transition labels) taking place within specific role instances.

Participants to a protocol session communicate by sending and receiving structured messages. Structured messages are represented by *message terms* which are inductively composed from variables and constants in the following way:
- Variables and constants are terms;
- `X[dt,et,RI,lab]` is a term (timed term), where `RI` is a formal parameter of type `role_instance`, X a variable of type text, text (fresh) or key, `dt` is a rational number in $\mathcal{Q}_{\geq\prime}$ and `ut` a rational number in $\mathcal{Q}_{\geq\prime} \cup \{\infty\}$ and `lab` is a transition label.
- `V'` is a term, with `V` any variable (*priming*);
- `{T}_K` is a term, with `K` variable of type symmetric_key public_key and `T` a term (*encryption*);
- `T1.T2` is a term, with `T1`, `T2` terms (*pairing*);
- `H(T)` is a term, with `T` a term and `H` a variable of type function (*hashing*);

• `inv(K)` and `{T}_inv(K)` are terms, with `K` a variable of type public key (*private key* and *signature*).
Priming of variables is used for variable assignments, to refer to the values of the variables after the assignment. A term of the form `X[dt,et,RI,lab]` represents a term `X` that will be disclosed between time `dt` and `et` relative to the execution of the transition labeled `lab` within role instance `RI`, and it will expire after the temporal bound `et`. Moreover, we add a predicate of the form `EXP(X)`, with `X` a variable of type text, text (fresh) or key, which intuitively holds true if `X` is assigned to a message which has expired. We also assume an additional label `start` which represents a fictitious transition taken at time 0 to initialize the main role.

A protocol run start from a, chosen, composition role called *main role*. The behavior of a basic role is described by means of a state-transition formalism. Intuitively, a state of the role instance is determined by the content of its local variables and the value of its actual parameters. Set of states of the role are declaratively denoted by standard boolean expressions over the value of variables and primed variables (e.g. a Boolean expression represents the set of role states where the Boolean expression holds true). A transition allows to leave a state of the role receiving a message (from another participant to the protocol) and to reach another state delivering a message. Transitions are declared in a role by a sequence of transition schemas.

A *timed transition* schema has the form:

```
lab. Pred Rec_Op >>(t1,t2,lb,ub,RI,lab1)
                     Primed_Pred /\ Send_Op
```

where,
• `lab` is a *label* identifying the current transition in the protocol schema;
• `Pred` is the *triggering predicate* defining the set of states from which the transition take place. It is a conjunction of a state predicate `SPred`, and possibly a message predicate `MPred`. `SPred` has the form $\bigwedge_{i=1}^{k} X_i = c_i$, where X_i is a state variable, namely a variable not occurring in any message term in the role, and c_i is a constant. `MPred` is a conjunction of (negations) of atoms either of the form $X = Y$ or $EXP(X)$, where X and Y are message variables occurring in message terms within the role.
• `Rec_Op` is an *optional* receive operation on a channel of the form `C(T)`, where `C` is a channel variable and `T` a term;
• `Primed_Pred` specifies the resulting state of the transition and has the form $\bigwedge_{i=1}^{z} X'_i = c_i$, where X'_i is a primed variable not occurring in any message term of the role and c_i is a constant. In the resulting state, the variables occurring in `Primed_Pred` are assigned the current value of the corresponding constant, and the remaining variables keep their current value;
• `Send_Op` is an *optional* send operation on a channel of the form `C(T)`, where `C` is a channel variable and `T` a term. Possible primed variables in `T` are assigned newly generated values (e.g., fresh nonces generation);
• `RI` is a formal parameters of the current role of type `role_instance`, `t1` and `lb` are rational numbers in $\mathcal{Q}_{\geq'}$, `t2` and `ub` rational numbers in $\mathcal{Q}_{\geq'} \cup \{\infty\}$, and `lab1` is a transition label. These parameters specify a transition that will be enabled between time `t1` and `t2` relative to the execution of the transition labeled `lab1` within the role of role instance `RI`, that will complete between time `lb` and `ub`.
THLPSL also allows for untimed transitions. For instance, a transition without any temporal constraints (neither delay/time out nor duration constraints) can be specified by `>>(0,∞,0,∞,RI,start)`.

An *urgent transition* schema has the form:

```
lab. Pred ->(t1,t2,RI,lab1) Primed_Pred
```

where all the parameters have the same interpretation as in the previous case. The intuitive semantics of an urgent transition is a transition which is enabled between time `t1` and `t2`, relative to the execution of the transition labeled `lab1` within the role instance `RI`, and is forced to trigger as soon as enabled, i.e. without any further delay. Notice that triggering of an urgent transition does not depend upon synchronization with other roles, as it cannot send or receive messages. Urgent transitions are intended to model activities local to a role, which have no duration and must not affect the overall timing.

The composition section of a composition role is used to instantiate other basic and composition roles using the following syntax:

```
R1(actual_parms) /\ R2(actual_parms) /\ ...
```

The intended meaning is that composed roles "run" in parallel with interleaving semantics.

III. Specification of the Wide Mouthed Frog protocol in THLPSL

In this section we consider the well known Wide Mouthed Frog authentication protocol [4]. The protocol involves three participants: Alice, Bob and the Server. Alice sends a message to the Server containing the identity of Bob (the intended receiver), a fresh session key K_{ab}, and a timestamp T_A, encrypted with a symmetric key K_{AS}, shared by Alice and the Server. The Server then checks if the timestamp is recent and, if this is the case, forwards the session key and a new timestamp T_B to Bob, encrypted with a symmetric key K_{BS}, shared by Bob

and the Server. Bob can now check if the timestamp T_S is recent and, if this is the case, accepts the session key as valid. Following is a description of the protocol steps:

1 $A \rightarrow S : A, \{B, K_{ab}, T_A\}_{K_{AS}}$
2 $S \rightarrow B : \{A, K_{ab}, T_S\}_{K_{BS}}$

The idea is that the participants use the timestamps to assess validity of the session key. A session key should be considered valid if the associated timestamp is recent enough. The protocol is known to be vulnerable to reply attacks, where an intruder simply repeatedly intercepts the message sent by the Server and, exploiting the structural similarity of the encrypted components in the two messages, repeatedly replies it back to the Server, who interprets it as a request to establish a new session key between the participants. If the intruder replies are fast enough, it can succeed in forcing the Server to keep the timestamps updated indefinitely, causing a, possibly compromised, session key to be associated to a fresh timestamp.

In order to model the validity of timestamps and session keys in THLPSL, we associate to each of them an expiration time. In particular, the initiator assigns an expiration time to the session key, wide enough to cover the estimated maximum delays of both the communication channels from Alice to the Server and from the Server to Bob. Similarly, Alice (resp., the Server) assigns the expiration time to each generated timestamp. An attack would be detected if Bob receives an expired session key associated with a non expired timestamp.

Below is a possible specification of the protocol, where we assume a maximum delay 5 to the channels connecting the participants. The expiration of the session key is set to the sum of the channels delays. The role for agent Alice is specified as follows:

```
role Alice(A,B,S:agent, SND:channel(dy,0,5),
          Kas:symmetric_key, AI:role_instance)
   played_by A def= :
   local Stat:nat, Ta:text(fresh), Kab:symmetric_key
   init Stat=0
   transition
   a0. Stat=0 >>(0,∞,0,0,AI,start) Stat'=1 /\
       SND(A.{Ta'[0,5,AI,a0].B.Kab[0,10,AI,a0]}_Kas)
end role
```

Notice that the role Alice is parametrized with respect to three agent names (`A, B,S`), one dy channel `SND`, one symmetric key `Kas`, and one role instance parameter `AI`. The `played_by` keyword states that the agent playing the role corresponds to the first agent parameter `A`. In the local variable declaration section the variable `Stat`, of type natural number, a fresh nonce variable `Ta` and a symmetric key `Kab` are declared. The `init` clause opens the variable initialization section, while the `transition` clause opens the section containing transition schemas. The transition schema, labeled `a0`, is a send timed transition which takes from a state where variable `Stat` is equal to 0 to a state where `Stat` is equal to 1, and all the remaining variables, except `Ta`, remain unchanged. The additional effect of the transition is that the term `A.{Ta'[0,5,AI,a0].B.Kab[0,10,AI,a0]}_Kas` is sent over the channel `SND`, where `Ta'[0,5,AI,a0]` represents a fresh timestamp generated and assigned to `Ta` by the transition, with disclosure/expiration interval between time 0 and 5 relative to the execution of the transition `a0` of the current role instance `AI`.

The role for agent Bob is specified as follows:

```
role Bob(A,B,S:agent, RCV:channel(dy,0,5),
        Kbs:symmetric_key, BI:role_instance)
 played_by B def=
 local Stat, Valid:nat, Ts:text, Kab:symmetric_key
 init Stat=0
 transition
 b0. Stat=0 /\ RCV({Ts'.A.Kab'}_Kbs)
          >>(0,∞,0,0,BI,start) Stat'=1
 b1. Stat=1 /\ not EXP(Ts) /\ not EXP(Kab)
  ->(0,∞,BI,start)
     Stat'=2  /\ Valid'=1
 b2. Stat=1 /\ not EXP(Ts) /\ EXP(Kab)
  ->(0,∞,BI,start)
     Stat'=2 /\ Valid'=0
end role
```

As to Bob's role, the first transition is a receive transition which requires that another party synchronously sends a message along the channel `RVC`, and that the sent message conforms to the structure of the term `{Ts'.A.Kab'}_Kbs`. The primed variables `Ts'` and `Kab'` in the received term are assigned, after the transition is executed, the value of the corresponding subterm in the unifying received message. The last two transitions are urgent transitions (always enabled) which test the validity of the timestamp and of the key and accept (resp., reject) the key by assigning the value 1 (resp., 0) to the boolean variable `Valid`.

The Server role is specified as follows:

```
role Server(A,B,S:agent, RCV,SND:channel(dy,0,5),
        Kas,Kbs:symmetric_key, SI:role_instance)
 played_by S def=
  local Stat:nat, Ts:text(fresh),
   Ta:text, Kab:symmetric_key
  init Stat=0
  transition
  s00. Stat=0 /\ RCV(A.{Ta'.B.Kab'}_Kas)
       >>(0,∞,0,0,SI,start) Stat'=1
  s01. Stat=1 /\ not EXP(Ta) >>(0,∞,0,0,SI,start)
       Stat'=3 /\ SND({Ts'[0,5,SI,s02].A.Kab}_Kbs)
end role
```

Notice that both the Server and Bob check for non expiration of timestamps (`not EXP(Ta)` and `not EXP(Ts)`) before proceeding (resp., before accepting the session key). Moreover, the Server sets expiration of the timestamps it generates relative to the transition generating it. To model possible acceptance by Bob of an invalid key, we use a variable `Valid` in

Bob's role, which is set to 0 (transition b2) if the accepted key has already expired, and to 1 (transition b1), otherwise.

The main role Main instantiates one instance of role Alice, one of the role Bob and three of the role Server. Roles are instantiated by associating actual parameters (i.e., constants) to formal ones. The resulting role instances are composed in parallel.

```
role Main()
 def=
 composition
   Alice(A,B,S,Snda,Kas,0) /\ Bob(A,B,S,Rcvb,Kbs,1)
   /\ Server(A,B,S,Snda,,Rcvb,Kas,Kbs,2)
   /\ Server(B,A,S,Sndb,Rcva,Kbs,Kas,3)
   /\ Server(A,B,S,Snda,Rcvb,Kas,Kbs,4)
end role
```

A simple property requiring acceptance only for valid keys is the following CTL formula $AG\neg(Alice0.Stat = 1 \wedge Bob1.Stat = 2 \wedge \neg Bob1.Valid)$, which can be checked by the model checker UPPAAL. The property is false, as it is possible for role instance bob to accept as valid a key after it has expired.

IV. From THLPSL specifications to UPPAAL XTAs

In this section we shall show how to encode a $THLPSL$ specification into a XTA suitable for model checking in UPPAAL.

UPPAAL XTA are an extension of Timed Automata. A Timed Automaton TA is a finite state automaton enriched with a set of real-valued clocks whose value can constrain the triggering of transition (for a formal definition and a account of the semantics see [3]). Transitions of a TA have the form $\langle l, \phi, a, \lambda, l' \rangle \in \delta$, written also $l \stackrel{\langle \phi, a, \lambda \rangle}{\longrightarrow} l'$, which represents a transition from the location l to the location l' on input symbol a; the *guard* ϕ is a constraint on clocks, and specifies when the transition is enabled; and the *update* set $\lambda \subseteq CK$ states the set of clocks to be reset on executing the transition. An eXtended Time Automaton[7] is the parallel composition $A_1 \parallel \ldots \parallel A_n$ of a collection of Timed Automata $A_1, \ldots, A_n$, in the style of CCS [14]. Automata communicate by means of channels and the communication style is handshaking. Input symbols of TA are replaced by channel names in XTA. If a is the name of a communication channel, then the symbol $a?$ denotes the receiving action over channel a, while the symbol $a!$ denotes the sending action over channel a. In addition, XTA can use (boolean and integer) variables and arrays. Therefore, the guard ϕ of a XTA transition may also constraints values of variables and array elements besides clocks. The update λ is generalized allowing also assignments involving variables and arrays.

As previously said, the formal semantics of $THLPSL$ has be given in [5] by translation into a network of timed automata. In such a translation a timed automaton is provided for each instance role and a timed automaton is provided for the intruder. States of both the participants and the intruder are structured and, in particular, encode besides control information also the knowledge of the playing part at the represented stage of the interaction. Knowledge is suitably encoded by sets of ground instances of message term (ground messages).

The intruder's knowledge is a monotonically increasing set of structured messages. A DY intruder can send to role instances any structured message that it can derive from its knowledge. For every received message the intruder can extract any possible submessage, compatibly with its knowledge of the necessary cryptographic keys. Conversely, known submessages can be recombined freely, using the algebra of message operators. Since structured message provide an unbounded use of pairing cryptographic encoding operators, the number of message the intruder can possibly build is unbounded. However, the messages composed by the intruder which are relevant for the protocol are those unifiable with the message patterns expected by the role instances. Even considering a bounded set of messages, the fact that the intruder can compose and/or modify communicated messages results in an explosion in the number of states (which depend on the subset construction of the set of received messages) and in the number of transitions. For a succinct encoding, the translation implemented in TPMC exploits the ability of UPPAL of handling XTA specifications enriched with variables and arrays. Arrays and variables are used both to encode the knowledge of the role instances and of the intruder, as well as the intruder's ability to compose and decompose messages. In particular, the intruder's knowledge is encoded by a boolean array K, where each location represents either a structured message sent along a channel, or a (sub)message obtained by composition/decomposition of known messages. A location of the array K is set to true when the intruder knows the corresponding (sub)message. Similarly, each role instance ri is encoded by an array of integers $\mathtt{N}_{ri}$, which contains the current ground instance associated to each variable occurring in a send or receive operation of the corresponding role.

Communication between role instances is not direct, but implemented by a pair of synchronizations, one between the sender and the intruder and one between the intruder and the receiver. Since communication in the formalism of XTA takes the form of pure communication, a different channel is provided for each conveyed message. Therefore, for each pair $< M, CHN >$, with m a structured ground message sent (resp., received) by a role instance and

CHN a channel name (intuitively, the channel where the message has been sent), a XTA syncronizzation channel named `C_CHN_s_m` (resp., `C_CHN_r_m`) is created.

Since delay/timeouts of timed transition and disclosure/expiration of timed messages are specified relative to a transition label, in order to model these timed feature a clock named `CK_lab_ri` is associated to every pair ⟨`lab`,`ri`⟩, such that transition label `lab` and role instance `ri` occur among the parameters of some timed transition or timed message term. Moreover, a boolean array `F` is used to record, for each transition label referenced within a timed message term or timed transition, whether it has been already executed. An additional clock named `CK_start` is used to model timed constraints referencing the special label `start`, corresponding to the initialization time of the main role. To model the duration of transitions taken by role instances, a local clock named $\mathtt{d}_{ri}$ is associated to role instance `ri`. To model channels delays, to every channel CHN, for which a delay constraint is specified, a clock `CK_CHN` is added. Finally, in order to model disclosure/expiration of timed messages, two boolean arrays `D` and `E` are used, which record whether a timed message has been disclosed or has expired, respectively.

The translation of a $THLPSL$ specification generates:
- an automaton for each role instance;
- an automaton for the intruder;
- an automaton (the *Time Machine*) responsible for handling disclosure and expiration of timed messages.
As to the generation of the automata for the role instances, the first step consists in collecting, by means of a fixpoint construction, the set GM of ground messages and the set $TM \subseteq GM$ of timed messages possibly generated by the protocol participants and the intruder, according to a typed model. This phase defines, for each role instance `ri`, the following correspondences, recorded in suitable data structures:
i. a function $\rho_{ri} : MVar_{ri} \longrightarrow 2^{GM}$ mapping each message variable of the role of `ri` onto a set of possible instances of that variable;
ii. for every message term M occurring in a receive operation in the role of `ri`, a function
$\chi^M_{ri} : Var'_{ri}(M) \longrightarrow 2^{GM}$, mapping the primed variable occurring in M ($Var'_{ri}(M)$) onto sets of possible instances of ground messages.
The function ρ_{ri} encodes a set of possible evaluations for the message variables of the role instance, in the sense that for a message variable X, $\rho_{ri}(X)$ gives the set of possible values of the vector element $N_{ri}[X]$, up to a suitable encoding of ground messages into integers. Function χ^M_{ri} encodes the result of a structure preserving unification mechanism between message terms expected by the receiver and ground terms sent by a sender (see [5] for a formal account).

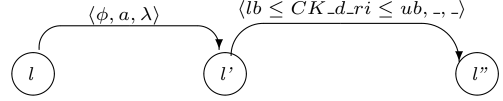

Fig. 1. XTA transitions encoding a timed transition.

In the following we sketch the construction of the instance role automaton for `ri`. For the sake of presentation, some of the technical details are omitted. Each location of a role instance automaton represents a location in which some state predicate holds. Let L be the set of atoms of the form $X = c$, such that $X = c$ occurs in a transition `SPred` or of the form $X' = c$, such that $X' = c$ occurs in a transition `Primed_Pred`. The set of locations of the role instance automaton for `ri` are in correspondence with subsets of L.

The general form of a sending timed transition is:

```
lab. SPred /\ MPred >>(t1,t2,lb,ub,RI,lab1)
                        Primed_Pred /\ CHM(M)
```

Each $THLPSL$ transition defines a set of pairs of XTA transitions, a pair for each possible instantiation (given by the functions ρ_{ri} and χ^M_{ri}) of the message variables $\{X_1, ..., X_k\}$ occurring in the transition, as shown in Fig. 1. The first XTA transition models the effect of the THLPSL transition, while the second one models its duration. With reference to Fig. 1, given $\theta = \{m_1, ..., m_k\}$ a possible instantiation of the message variable $\{X_1, ..., X_k\}$ according to ρ_{ri}:
• l is a location corresponding to a set of atoms in L which contains all the atoms in `SPred`.
• l' is a location corresponding to a set of atoms in L which contains all the atoms of the form $X = c$, such that $X' = c$ occurs in `Primed_Pred`, and all the atoms $X = c$ occurring in `SPred` such that X' does not occurs in `Primed_Pred`.
• l'' is a distinct copy of l', introduced to model transition duration.
• ϕ is a conjunction of: (i) atoms of the form $N_{ri}[X_i] = m_i$ for every message variable X_i occurring unprimed in the message term M; (ii) atoms of the form $N_{ri}[X_i] = N_{ri}[X_j]$ (resp., `not` $N_{ri}[X_i] = N_{ri}[X_j]$), for every atom of the form $X_i = X_j$ (resp., `not` $X_i = X_j$) occurring in `MPred`, and atoms of the form $E[X_i] = 1$ (resp., $E[X_i] = 0$), for every atom of the form $EXP(X_i)$ (resp., `not` $EXP(X_i)$) occurring in `MPred`; and (iii) the clock condition $t_1 \leq$ `CK_lab1_ri` $\leq t_2$.
• a is `C_CHM_s_m!`, where m is the ground message obtained by substituting $\{X_1, ..., X_k\}$ by θ.
• λ is a set of assignments contag `F[lab_ri]` $:= 1$, `CK_d_ri` $:= 0$, `CK_lab_ri` $:= 0$, $N_{ri}[X_i] := m_i$ for each X_i occurring primed in M.
Notice that the transition guard ϕ enables the transition when the current state: (i) assigns to the un-

primed variables the ground messages assigned by θ; and (ii) satisfies all the conjuncts in `MPred`. The update λ sets the flag `F[lab_ri]` to record the execution of the transition, resets the clocks associated to the transition, and assigns fresh values to the primed variables in the message term sent. The general form of a receive timed transition is:

```
lab. SPred /\ MPred  /\ CHM(M)
     >>(t1,t2,lb,ub,RI,lab1) Primed_Pred
```

Each receive $THLPSL$ transitions defines a set of XTA transitions, one for each possible instantiation (given by the functions ρ_{ri} and χ_{ri}^M) of the message variables $\{X_1, ..., X_k\}$ occurring in the transition. Given $\theta = \{m_1, ..., m_k\}$ a possible instantiation of the message variable $\{X_1, ..., X_k\}$ according to ρ_{ri}, and $\psi = \{u_1, ..., u_z\}$ a possible matching, according to χ_{ri}^M, for the message variable $\{Y_1, ..., Y_z\}$ occurring primed in M, a pair of XTA transitions as in Fig. 1 is added, where:

- l, l', l'' and ϕ are defined as for send transitions;
- a is `C_CHM_r_m?` where m is the ground term obtained by substituting the message variables in $\{X_1, ..., X_k\}$ which occur unprimed in M by θ and all the message variables in $\{Y_1, ..., Y_z\}$ by ψ;
- λ is a set of assignments containing `F[lab_ri]` $:= 1$, `CK_d_ri` $:= 0$, `CK_lab_ri` $:= 0$, $N_{ri}[Y_i] := u_i$, for each $Y_i \in \{Y_1, ..., Y_z\}$.

To guarantee that the duration of a timed (send or receive) transition is modeled correctly, the intermediate location in Fig. 1 is equipped with the invariant[1] $lb \leq CK_d_ri \leq ub$.

Since urgent transitions cannot send or receive messages, neither synchronization nor update is necessary. Therefore, they are encoded as XTA transitions between a starting location l to an ending location l' defined as in the previous cases, with the addition that the starting location is set urgent, and the guard condition is a conjunction of atoms of the same form as those defined for cases *(ii)* and *(iii)* for timed send or receive transitions.

To model a DY intruder, the intruder automaton plays the role of the communication channel between the role instances, and it is allowed to compose, decompose, forward, block and delay messages. The automaton has a single location and loop transitions for sending known messages to role instances, receiving messages sent by role instances and composing/decomposing messages.

For every ground message $m \in GM$ and channel CHN, there is a loop transition for a send action, whose decoration $\langle \phi, a, \lambda \rangle$ is
$\langle \mathrm{K[m]} = 1 \wedge \texttt{CK_CHN} \geq lb, \ \texttt{C_CHN_r_m!}, _ \rangle$,

[1] Invariants are associated to locations; remaining in a location is allowed as long as the invariant holds true.

where `CK_CHN` is the clock associated to channel CHN to model channel delay.

For every a ground message $m \in GM$ and channel CHN, there is a loop transition for a receive action, whose decoration $\langle \phi, a, \lambda \rangle$ is
$\langle _, \ \texttt{C_CHN_s_m?}, \ \texttt{K[m]:=1;CK_CHN:=0} \rangle$

Transitions for composition/decomposition of messages encode the standard rules of a DY intruder. For instance, if the intruder knows two ground messages m_1 and m_2 and $m_1.m_2 \in GM$, then it also knows $m_1.m_2$ (and vice versa). Similarly, if it knows a ground messages $\{m_1\}_k$ and a ground key k, and $m_1 \in GM$, then it also knows m_1 (and vice versa). The loop transitions for the above two composition, decomposition actions have the following decorations: the former is $\langle K[m_1] \wedge K[m_2], _, K[m_1.m_2] := 1 \rangle$ $(\langle K[m_1.m_2], _, K[m_1] := 1; K[m_2] := 1 \rangle)$, and the latter is $\langle K[\{m_1\}_k] \wedge K[k], _, K[m_1] := 1 \rangle$ $(\langle K[m_1] \wedge K[k], _, K[\{m_1\}_k] := 1 \rangle)$.

The *Time Machine* automaton (TM) is responsible for handling disclosure and expiration of timed messages by updating the boolean arrays D and E. The array F is used to record the execution of a transition referenced by some timed message (or timed transition). Therefore, disclosure or expiration of a timed message relative to a given transition is performed only if the referenced transition of role instance `ri` labeled `lab` has been executed (`F[lab_ri]` $=$ `1`).

For every ground message m which is an instance of a timed variable message `X[dt,et,RI,lab]` in role instance `ri`, a loop transition for disclosure is added, whose decoration $\langle \phi, a, \lambda \rangle$ is

$$\langle \texttt{F[lab_ri]} = 1 \wedge \texttt{not D[m]} \wedge \texttt{CK_lab_ri = dt}, _, \texttt{D[m]} := 1 \rangle$$

and a loop transition for expiration is added whose decoration $\langle \phi, a, \lambda \rangle$ is

$$\langle \texttt{F[lab_ri]} = 1 \wedge \texttt{not E[m]} \wedge \texttt{CK_lab_ri} = et, _, \texttt{E[m]:=1} \rangle$$

To guarantee disclosure/expiration transition at due time without any further delay, the location of TM is equipped with an appropriate invariant. The invariant is a conjunction, over all the ground instances m of the timed message terms of the form `X[dt,et,RI,lab]` within role instance `ri`, of constraints of the form:

$$(\texttt{F[lab_ri]} \wedge \texttt{not D[m]}) \rightarrow \texttt{CK_lab_ri} \leq \texttt{dt}) \wedge$$
$$(\texttt{F[lab_ri]} \wedge \texttt{D[m]} \wedge \texttt{not E[m]}) \rightarrow \texttt{CK_lab_ri} \leq \texttt{et})$$

V. Conclusions

We have implemented a prototype model checker TPMC in C++. The tool integrates a compiler from THLPSL specifications to UPPAAL XTAs with the model checking engine provided by UPPAAL. To assess the efficiency and scalability of the tool, we ran

Protocol	Inst	CT	VT	Max Inst	CT	VT
WMF	1-1-3	.01	.44	2-2-5	.06	337.37
WMF_{Fix}	1-1-3	.01	.03	2-2-5	.06	111.31
NSPK	3	.02	.86	5	.68	14.75
NSPK_{Fix}	3	.02	.10	5	.12	26.42
ISO1	1	.07	.31	8	326.79	339.96
PBK	2	.01	.02	8	.38	15.29
PBK_{Fix}	2	.01	.02	8	.30	138.18

TABLE I: Experimental results.

TPMC on a number of timed and untimed protocols. An excerpt of the results of our experiments is given in Table I. The table shows the results of TPMC for the original and fixed version (as proposed by Lowe [13]) of the Wide Mouthed Frog protocol, and the following untimed protocols: the Needam–Schroeder Public Key protocol (both in the original and fixed version), the PBK protocol (both in the original and fixed version), and the ISO1 protocol (the specifications of these protocols are taken from the AVISPA library of protocols [1]). All the tests are parametric in the number of sessions, where a session involves from two to three participants, depending on the protocol analyzed. Clearly, the bigger the number of sessions, the higher the number of agents and of ground messages sent/received, leading to a growth in the state space to be analyzed. The column Inst. of the table reports the number of sessions of the corresponding test except for the Wide Mouthed Frog protocol, where it reports the number of instances of role Alice, of role Bob and of role Server, respectively. The property verified for the Wide Mouther protocol is the one reported in Section III, while the property checked for all the untimed protocols is strong authentication. We only report the results for the minimal and maximal instance of the protocols we tried to analyze with TPMC. The table reports both the time in seconds spent by the compiler (CT) from THLPSL to UPPAAL and the time in seconds spent by UPPAAL (VT) in the verification phase. The experiments have been run on a 3.0GHz Pentium IV with 1Gb of memory running Linux (Slackware 11.0). On all tests, TPMC correctly reports the expected attack on the flown version of the protocol and no attacks for the fixed versions. Notice that, TPMC allows for both the specification of timed and untimed protocols. Even though TPMC is not optimized for handling untimed protocols, the tests show on those protocols performances which are comparable with those of available tools for untimed protocols (see, e.g., [1]).

We are currently working on the specification and verification of the TESLA protocol[15] as well as other time dependent protocols. We also plan to investigate different forms of intruder, e.g., intruders with restricted abilities compared to a DY intruder, or intruders whose actions take non negligible time. We are also working on an extension of language for security goals so as to allow for time dependent goals.

References

[1] AVISPA: Automated Validation of Internet Security Protocols and Applications. `http://avispa-project.org`.

[2] R. Alur, T. Henzinger, M. Vardi, Parametric real-time reasoning, STOC 1993, pp.592-601.

[3] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science, 126, pp. 183-235, 1994.

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

[5] M. Benerecetti, N. Cuomo, and A. Peron, Timed HLPSL for specification and verification of time sensitive protocols. Proceedings of the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'06), Seattle, August 15–16, 2006.

[6] R. Barbuti, N. De Francesco, A. Santone, L. Tesei, A Notion of Non-Interference for Timed Automata, Fundamente Informaticae, 54(2-3), 177-150, 2003.

[7] J. Bengtsson, W. Yi: Timed Automata: Semantics, Algorithms and Tools. Lectures on Concurrency and Petri Nets 2003: 87-124

[8] D. Dolev, A.C. Yao, On the Security of Public-Key Protocols, IEEE Transactions on Information Theory, 29(2):198–208, 1983.

[9] C. Daws, A. Olivero, S. Tripakis, S. Yovine, The tool KRONOS, In Hybrid Systems III: Verification and Control, LNCS 1066, pp. 208-219, 1996.

[10] R. Gorrieri E. Locatelli, F. Martinelli, A simple Language for Real Time Cryptographic Protocol Analysis, ESOP 2003, LNCS 2618, pp. 114-128, 2003-03-27

[11] Leslie Lamport, The Temporal Logic of Actions, in ACM Transactions on Programming Languages and Systems, Vol. 16(3), ACM Press, pp. 872–923,1994.

[12] R. Lanotte, A. Maggiolo-Schettini, S. Tini, Timed Information Flow for Timed Automata, submitted.

[13] Gavin Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[14] R. Milner. A Calculus for Communicating Systems. LNCS 92, 1980.

[15] A. Perrig, R. Canetti, J. D. Tygar, D. Song, Efficient Authentication and Signing of Multicast Streams over Lossy Channels. IEEE Symposium on Security and Privacy 2000: 56-73.

[16] M. Napoli, M. Parente, and A. Peron. Specification and verification of protocols with time constraints. Electronic Notes in Theoretical Computer Science, 99:205–227, 2004.

[17] K. Larsen, P. Petterson, W. Yi, UPPAAL in a nutshell, Springer International Journal of Software Tools for Technology Transfer, 1, 1997.

[18] C. Meadows, Formal methods for cryptographic protocol analysis: emerging issues and trends. IEEE Journal On Selected Area in Communications, 21, 2003

[19] J. Zhou, D. Gollmann, An Efficient Non-repudiation Protocol, 10-th Computer Security Foundation Workshop (CSFW'97), Rockport, Massachusets, USA, 126–132, 1997.