TPMC: A Model Checker For Time–Sensitive Security Protocols

Massimo Benerecetti, Nicola Cuomo, Adriano Peron

Dept. of Physical Sciences, Università di Napoli "Federico II", Napoli, Italy

Email: {bene, ncuomo, peron}@na.infn.it

Abstract— In this paper we consider the problem of verifying time–sensitive 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, a model checking tool, TPMC, for the analysis of security protocols is presented, which employs THLPSL as a specification language and UPPAAL as the model checking engine. To illustrate the tool, we provide a specification of the Wide Mouthed Frog protocol in THLPSL, and report some experimental results on a number of timed and untimed 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 [19]). 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 [21], 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 [2], and the corresponding *model checking* verification techniques are supported by a variety of tools. Examples of model checkers for Timed Automata are KRONOS [12], which supports model checking of branching time requirements, and UPPAAL toolkit [18], which 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 this paper we propose the TPMC (Timed Protocols Model Checker) tool we developed for the analysis of timed security protocols. TPMC employs THLPSL, a timed extension of HLPSL, as a specification language, and UPPAAL as the model checking engine.

Timed HLPSL (THLPSL for short) has been introduced in previous papers ([5], [6]). The temporal features introduced in THLPSL are: (a) temporal constraints on 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 of channel delivery.

Most work in security protocol analysis implicitly assume a Dolev-Yao model of the intruder [10]. On the other hand, in complex scenarios a DY intruder may be a too strong assumption, as network infrastructures often employ heterogeneous technologies, vulnerable to different intruder threats. Moreover, some protocols are explicitly designed to guarantee security properties only under stronger assumptions on the reliability ensured by the communication channels. For example, the Zhou-Gollmann protocol [21] assumes that communication is carried out on secure or on resilient channels, where delivery is eventually guaranteed. Therefore, in this paper THLPSL has also been extended so as to provide the ability to specify different models of intruder and types of channels (resilient and operational channels). In

addition, we introduce a form of time–sensitive security goals, called timed authentication, which extends the authentication goals of HLPSL to a timed setting.

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 (eXtended Timed Automata), without exploiting the full expressive power of the UPPAAL language, which allows for shared integer variables, and integer and boolean arrays. The use of these additional features allows for exponentially more succinct UPPAAL specifications. The mapping implemented in TPMC is not the one described in the formal semantics, but an equivalent one which, taking advantage of the full expressiveness of UPPAAL XTAs, can be more efficiently employed for implementation purposes. We shall use the Wide Mouthed Protocol as a simple, but significant, running example of time–sensitive protocol. We also provide an experimental evaluation of the tool on a number of security protocols.

The paper is organized as follows: in the next section we present the specification language THLPSL; in Section III we provide the THLPSL specification for the Wide Mouthed Frog protocol. In Section IV we describe the translation of THLPSL specifications into UPPAAL XTAs [7]. In Section V we present some experimental evaluation of our tool. Finally, we discuss the related work.

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 semantics, which is given in terms of Timed Automata, 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 to express the following temporal aspects:

a) temporal constraints on the control-flow of participants to a protocol session (delays and timeouts);

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.

In addition, THLPSL provides the ability to model different kinds of intruders and channels (even within a single protocol specification). In particular, THLPSL allows to specify:

- the actions the intruder can perform on a channel;
- the actions the intruder can perform on the messages listened over the channels;
- the reliability of a channel with respect to message delivery.

As to reliability of channels, THLPSL provides different types of channels:

- *operational channels*, where the delivery of a message is guaranteed within a bounded time interval $[lb, ub]$;
- *resilient channels*, where the delivery of a message is eventually guaranteed, but no upper bound is ensured (i.e., an operational channel with unbounded delivery delay);
- *unreliable channels*, where the delivery of a message cannot be guaranteed.

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; and *composition roles* that compose in parallel (assuming the standard interleaving semantics) instances of basic roles (one for each participant to the protocol session), instantiating their formal parameters with actual parameters (i.e., constants). Following is an abstract schema of a THLPSL specification, declaring two basic roles, Alice and Bob, and a composition role Main which composes in parallel instances of the basic roles.

```
role Alice(<formal parameter list>)
 played_by <agent>  def= :
    local <variable list>
    init <initialisation predicate>
    transition <transition list>
end role

role Bob(<formal parameter list>)
 played_by <agent>
 ...
end role
...
role Main()
 def=
 composition
   Alice(<actual parameter list>) /\
   Bob(<actual parameter list>) /\ ...
end role
```

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. A channel declaration has the form `C:channel(ct,act,lb,ub)`, where `ct` may assume the following three values: `U` for unreliable channel, `O` for operational channel and `R` for resilient channel; `act` may assume the following three values: `R` for a channel on which the intruder has read-only access, `W` for a channel on which the intruder has write-only access; and `RW` for a channel on which the intruder has read-write access; `lb` is the least transmission delay (a rational number in $\mathcal{Q}^+$) and `up` is the maximum transmission delay (a rational number in $\mathcal{Q}^+ \cup \{\infty\}$).

Notice that a channel under the control of a Dolev-Yao (DY) intruder [10] (i.e., an intruder who has complete control over the channel) can be specified as follows `C(U,RW,ub,lb)`, and can be abbreviated in THLPSL as `C(dy,ub,lb)`.

THLPSL 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 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 X is a variable of type text, text (fresh) or key, `dt` is a rational number in $\mathcal{Q}^+$ and `et` a rational number in $\mathcal{Q}^+ \cup \{\infty\}$, `RI` is a formal parameter of type `role_instance`, and `lab` is a transition label.
- `V'` is a term, with `V` any variable (*priming*);
- `{T}_K` is a term, with `K` a term of type symmet-

ric_key or 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 term of type function (*hashing*);
- `inv(K)` and `{T}_inv(K)` are terms, with `T` a term and `K` a term of type public key (*private key* and *signature*).

A message term where no variable occurs is called *ground message*. Priming of variables is used for variable assignments, to refer to the values of the variables after the assignment. Priming can be used in different contexts:

- a primed variable `X'` can occur within a message term sent over a channel and models the generation of a fresh ground message (nonce) assigned to `X`;
- a primed variable `X'` can occur within a message term received over a channel, and models the assignment to `X` of the ground message received in the communication with some party;
- a primed variable can occur in a predicate of the form `X' = <const>` to assign the value `<const>` to `X` in the target state of a transition.

A term of the form `X[dt,et,RI,lab]` represents a message term `X` that is disclosed between time `dt` and `et` relative to the execution of the transition labeled `lab` within role instance `RI`, and which expires after the bound `et`. Moreover, we add predicates `DIS(X)` and `EXP(X)`, with `X` a variable of type text, text (fresh) or key, which holds true if `X` is assigned to a message which has already been disclosed, resp. has expired. A predefined label `start` is provided to refer to the event of initializing the 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). For instance, the Boolean expression `Stat=1 /\ DIS(X)` represents the set of states where variable `Stat` evaluates to 1 and the message assigned to `X` has been disclosed.

A transition allows to leave a state of the role, possibly receiving a message (from another participant), and to reach a state, possibly sending a message. Transitions are declared in a role by a sequence of transition schemas having one of the following forms:

```
lab. Pred Rec_Op >>(t1,t2,lb,ub,RI,lab1)
              Primed_Pred /\ Send_Op
```
(1)
```
lab. Pred ->(t1,t2,RI,lab1) Primed_Pred
```
(2)

Transition schema of the form (1) specifies *timed transitions* where:

- `lab` is a *label* identifying the current transition in the role schema;
- `Pred` is the *triggering predicate* defining the set of states which the transition takes place from. 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 $DIS(X)$ or $EXP(X)$, where X and Y are message variables, namely variables occurring in message terms within the role;
- `Rec_Op` is an optional receive action of the form `C(T)`, where `C` is a channel variable and `T` a message term;
- `Primed_Pred` specifies the target states of the transition. It 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 target states, the variables occurring in `Primed_Pred` are assigned the corresponding constant value, and the remaining variables keep their current values;
- `Send_Op` is an *optional* send action on a channel of the form `C(T)`, where `C` is a channel variable and `T` a term. Primed variables in `T` are assigned newly generated values (e.g., fresh nonces);
- `RI` is a formal parameters of the current role of type `role_instance`, `t1` and `lb` are rational numbers in $\mathcal{Q}^+$, `t2` and `ub` rational numbers in $\mathcal{Q}^+ \cup \{\infty\}$, 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 the role instance `RI`, that will complete between time `lb` and `ub`.

Notice that untimed transitions can easily be recovered in THLPSL. For instance, a transition without any temporal constraints (neither delay/time out nor duration constraints) can be specified as follows: `>>(0,∞,0,∞,RI,start)`. Without loss of generality, we assume that no timed transition schema contains both a send action and a receive action.

Transition schema of the form (2) specifies *urgent transitions*, namely transitions which are intended to model activities local to a role, which have no duration and whose execution must not affect the overall timing. The parameters have the same interpretation as in the previous case. The intuitive reading 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 send or receive actions in an urgent transition are not

allowed.

Security properties can be specified within the goal section. THLPSL provides three kind of properties: secrecy properties, (weak and strong) authentication, and (weak and strong) timed authentication properties. Following is an example of a goal section containing a security goal of each type:

```
goal
    secrecy_of M
    Alice authenticates Bob on M
    Alice Tauthenticates Bob on M
end goal
```

The first goal is a *secrecy property* and requires that message term M is kept secret during any protocol run (i.e., it is not known by the intruder). The second goal is a *authentication property*, which requires that a receipt of an instance of message M by Alice must match with the same instance previously sent by Bob (in other words, message M serves as a proof of authentication between the two parties). The third goal is a *timed authentication property*, which requires authentication of Alice and Bob with the additional timed constraint that the authentication must occur during the temporal validity of message M, namely after its disclosure and before its expiration.

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 of the protocol 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 10, a value greater than the expiration time of the timesptamps. The role for agent Alice is specified as follows:

```
role Alice(A,B,S:agent, SND:channel(dy,0,inf),
           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)
   /\ Twitness(A,B,k,Kab)
end role
```

Notice that role Alice is parameterized 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. The disclosure/expiration interval of Ta is set between time 0 and 5 relative to the execution of the transition a0 of the current role instance AI. This transition also issues a Twitness action, which is used to model timed authentication together with a matching Trequest issued by Bob. The role for Bob is specified as follows:

```
role Bob(A,B,S:agent, RCV:channel(dy,0,inf),
        Kbs:symmetric_key, BI:role_instance)
 played_by B def=
 local Stat, 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) -->(0,∞,RI,start)
     Stat'=2 /\ Trequest(B,A,k,Kab)
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 transition is an urgent transitions which test the validity of the timestamp and, if it is not yet expired, accepts the key issuing a Trequest action on it.

The Server role is specified as follows:

```
role Server(A,B,S:agent, RCV:channel(dy,0,inf),
     SND:channel(dy,0,inf), Kas:symmetric_key,
     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.

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., the declared constants) to formal ones. The resulting role instances are composed in parallel.

```
role Main()
 def=
 const a,b,s: agent, kas,kbs: public_key,
       snda,rcva,sndb,rcvb: channel(dy,0,5)
 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
```

The security goal for the protocol can be specified within the goal section. A *timed authentication property* is specified in THLPSL using the Twitness and
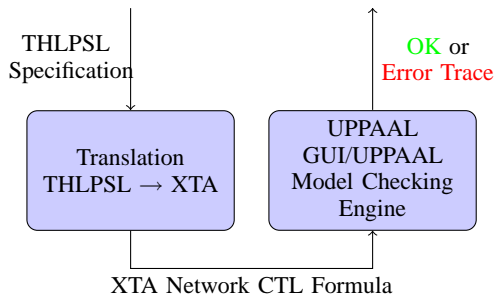
THLPSL Specification

OK or Error Trace

Translation THLPSL → XTA

UPPAAL GUI/UPPAAL Model Checking Engine

XTA Network CTL Formula

Figure 1. The architecture of the tool.

the `Trequest` keywords. For instance, the following timed authentication goal:

```
goal
   Alice Tauthenticates Bob on Kab
end goal
```

together with the request and witness condition occurring in the specification (`Trequest(B,A,k,Kab)` and `Twitness(A,B,k,Kab)`), requires that a receipt of a not yet expired instance of `Kab` by Bob must match with the same instance sent by Alice.

IV. FROM THLPSL SPECIFICATIONS TO UPPAAL XTAs

The general architecture of TPMC is depicted in Fig. 1. The system is composed of two modules, a verification engine, namely the model checker UPPAAL [18], and a compiler which takes as input a THLPSL specifications and translates it into the input language of UPPAAL. The input language of UPPAAL is a textual representations of an XTA. The tool takes a THLPSL specification and automatically generates a network of XTAs, simulating the protocol and an appropriate CTL formula, which encodes the desired security goal possibly included in the THLPSL specification. Therefore, in this section we shall show how a $THLPSL$ specification can be encoded into $XTAs$, which are then model checked by UPPAAL.

UPPAAL $XTAs$ are an extension of Timed Automata. A Timed Automaton TA is a finite state automaton enriched with a set CK of real-valued clocks, whose value can constrain the triggering of transitions (for a formal definition and an account of the semantics see [2]). Transitions of a TA have the form $l \xrightarrow{\langle \phi, a, \lambda \rangle} l'$, 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. The *update* set $\lambda \subseteq CK$ states the set of clocks to be reset to zero on executing the transition. Additional constraints over clocks, called *invariants*, may be associated to locations. An invariant states the temporal condition under which the automaton is allowed to remain idle in that location. An eXtended Time Automaton [7] is the

parallel composition $A_1 \parallel \ldots \parallel A_n$ of a collection of Timed Automata $A_1, \ldots, A_n$. 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, $XTAs$ can use (boolean and integer) variables and arrays. Therefore, the guard ϕ of a XTA transition may also constrain 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 the basic fragment of $THLPSL$ has been 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. In order to define these automata, one has to consider, besides the control-flow information provided by the protocol specification, also the set of ground messages which can be sent or received or are simply known by the participants and the intruder.

A. Generation of ground messages.

From a theoretical viewpoint, under the hypothesis of a finite number of sessions and of a finite number of fresh messages, finiteness of the set of ground messages necessary to assess security (or insecurity) of a protocol is guaranteed by the results reported in [17]. In order to satisfy the theoretical constraints above, we restrict to THLPSL specifications where a finite number of role instances are allowed and no cycles are allowed in the control-flow of roles. This ensures that only a finite number of nonces can be generated. In this subsection we sketch the procedure generating the finite set of ground messages upon which the runs of the protocols are defined.

Let GM be the possibly infinite set of ground messages defined as the closure of the set of atomic ground messages GM_0, under the formation rules of message terms given in Section II. The set GM_0 is defined as follows. It contains: (i) all the constants declared in the roles; (ii) the actual parameters used to instantiate each role instance; (iii) a distinct copy of a fresh atom for each primed variable occurring in a send action of a role instance (nonces generated by role instances); (iv) a distinct copy of a fresh atom for each primed variable occurring in a receive action of a role instance (nonces generated by the intruder). Under the enforced restrictive hypothesis, GM_0 is a finite set. We can now give the definition of the unification relation between message terms and ground terms, namely the typed structural matching of messages involved in the synchronous communication (handshaking) between a sending and

a receiving transition. Let $\mathcal{X}$ be a set of variables and MT be the set of message terms built from variables and constants. Let $\rho : \mathcal{X} \to 2^{GM_0}$ be a type-preserving function, assigning sets of atomic ground messages to variables according to their type. We can define the *unification relation* $\Rightarrow_\rho \subseteq MT \times GM \times 2^{\mathcal{X} \times GM_0} \times 2^{\mathcal{X} \times GM_0}$, which associates a message term M and a ground message m with two partial valuation functions θ and θ^a, binding unprimed and primed variables occurring in M to ground messages, respectively. Intuitively, θ captures a possible instantiation of the unprimed variables of M according to ρ, while θ^a captures possible assignments to the primed variables occurring in M, so that the resulting ground message is an instance of the term M. The unification relation is defined inductively as follows:

- $(m, m) \Rightarrow_\rho \varnothing, \varnothing$ if $m \in GM$;
- $(X, m) \Rightarrow_\rho \{(X, m)\}, \varnothing$ if $m \in \rho(X)$;
- $(X', m) \Rightarrow_\rho \varnothing, \{(X, m)\}$, with $X \in Var$;
- $(\{M\}_Z, \{m\}_k) \Rightarrow_\rho \theta_1 \cup \theta_2, \theta_1^a \cup \theta_2^a$ if $(M, m) \Rightarrow_\rho \theta_1, \theta_1^a$, $(Z, k) \Rightarrow_\rho \theta_2, \theta_2^a$, and both $\theta_1 \cup \theta_2$ and $\theta_1^a \cup \theta_2^a$ are partial functions;
- $(M_1.M_2, m_1.m_2) \Rightarrow_\rho \theta_1 \cup \theta_2, \theta_1^a \cup \theta_2^a$ if $(M_1, m_1) \Rightarrow_\rho \theta_1, \theta_1^a$, $(M_2, m_2) \Rightarrow_\rho \theta_2, \theta_2^a$, and both $\theta_1 \cup \theta_2$ and $\theta_1^a \cup \theta_2^a$ are partial functions;
- $(H(M), m) \Rightarrow_\rho \theta_1 \cup \theta_2, \theta_1^a \cup \theta_2^a$ if for some $h, \hat{m} \in GM$, $(M, \hat{m}) \Rightarrow_\rho \theta_1, \theta_1^a$, $(H, h) \Rightarrow_\rho \theta_2, \theta_2^a$ and $h(\hat{m}) = m$, and both $\theta_1 \cup \theta_2$ and $\theta_1^a \cup \theta_2^a$ are partial functions;
- $(inv(M), m) \Rightarrow_\rho \theta, \theta^a$ if for some $\hat{m} \in GM$, $inv(\hat{m}) = m$ and $(M, \hat{m}) \Rightarrow_\rho \theta, \theta^a$;
- $(\{M\}_{inv(Z)}, \{m\}_k) \Rightarrow_\rho \theta_1 \cup \theta_2, \theta_1^a \cup \theta_2^a$ if $(M, m) \Rightarrow_\rho \theta_1, \theta_1^a$, $(inv(Z), k) \Rightarrow_\rho \theta_2, \theta_2^a$, and both $\theta_1 \cup \theta_2$ and $\theta_1^a \cup \theta_2^a$ are partial functions.

Actually, only a finite subset of messages in GM is relevant for verification purposes, since the protocol runs can only progress when the ground message received over a channel matches the structure of a message term in a receive action. Therefore, let $\overline{GM} \subset GM$ be the set of ground messages in GM which unify with a message term M received over a channel in some receive THLPSL transition. In other words, $m \in \overline{GM}$ if $(M, m) \Rightarrow_\rho \theta, \theta^a$ for some term M occurring in a receive action of some transition. Since the number of such terms is finite as finite are the possible instantiations of every variable since GM_0 is finite, the set $\overline{GM}$ is clearly a finite set. The set $\overline{GM}$ can easily be computed by a simple least fixed-point computation, which applies the unification relation starting from GM_0 to the message terms occurring in send and receive actions of the roles. Each ground message $m \in \overline{GM}$ is, then, encoded by a unique integer. In the following, we shall confuse a message m with its integer encoding.

B. Definition of clocks, channels and data structures.

We shall now describe the set of clocks used to express time constraints of the protocol, the set of channels used for communication, and additional data structures (arrays and variables) to encode the knowledge of the participants and the intruder.

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 ground message possibly sent (resp., received) by a role instance and CHN a channel name (i.e., the channel where the message has been sent), a XTA synchronization channel named `C_CHN_s_m` (resp., `C_CHN_r_m`) is created.

Delays/timeouts of timed transitions and disclosure/expiration of timed messages are specified relative to a transition label. In order to model these features, a clock named `CK_lab_ri` is associated to every pair $\langle$lab,ri$\rangle$, such that transition label `lab` and role instance `ri` occur among the parameters of some timed transition or timed message term. 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. 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. To model durations of transitions taken by role instances, a local clock named d_{ri} is associated to each role instance `ri`. Finally, for every channel CHN for which a delay constraint is specified, a clock `CK_CHN` models channel delays. 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.

For each role instance `ri`, we define a type preserving function $\rho_{ri} : \mathcal{X}_{ri} \longrightarrow 2^{GM_0}$, mapping message variables of the role of `ri` ($\mathcal{X}_{ri}$) onto sets of possible atomic ground messages. To each role ri, a vector $N_{ri}[]$ is associated which encodes the knowledge of the role instance at a particular stage of the protocol. The array location $N_{ri}[X]$, where X is a variable occurring in the role of ri, may contain a possible instantiation of X belonging to $\rho_{ri}(X)$.

Differently from role instances, the knowledge of the intruder depends upon the set of ground messages initially known by the intruder, the set of ground messages sent by principals during the protocol and the capability of the intruder to read a message and its structural components, and, possibly, to modify a message or some of its components. The intruder knowledge is, therefore,
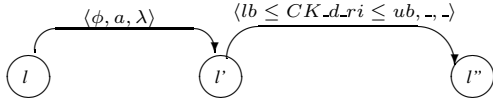
Figure 2. XTA transitions encoding a timed transition.

encoded by a boolean array K[] indexed by the set of ground messages $\overline{GM}$. A location of the array K[] is set to true when the intruder knows the corresponding message.

C. Definition of the automaton of a role instance

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

Let us first consider send timed transitions, whose general form 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 (according to the unification relation $\Rightarrow_{\rho ri}$) of the message variables occurring in the transition, as shown in Fig. 2. The first XTA transition models the effect of the THLPSL transition, while the second one models its duration.

With reference to Fig. 2, for every $m \in \overline{GM}$ such that $(M, m) \Rightarrow_{\rho ri} \theta, \theta^a$, the first transition added is such that

1. l is a location corresponding to a set of atoms in L which contains all the atoms in SPred;
2. 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;
3. l'' is a distinct copy of l', introduced to model transition duration;
4. ϕ is a conjunction of:
   a) atoms of the form $N_{ri}[X_i] = m_i$, where $(X_i, m_i) \in \theta$, for every message variable X_i occurring unprimed in the message term M;
   b) the constraint F[lab1_ri] $\wedge$ $t_1 \leq$ CK_lab1_ri $\leq t_2$;
   c) atoms of the form $N_{ri}[X_i] = N_{ri}[X_j]$ (resp., $\neg N_{ri}[X_i] = N_{ri}[X_j]$), for every atom of the form $X_i = X_j$ (resp., $\neg X_i = X_j$) occurring in MPred;

d) constraints of the form F[lab2_ri] $\wedge$ CK_lab2_ri $\geq dt$ ($\neg$ (F[lab2_ri] $\wedge$ CK_lab2_ri $\geq dt$), resp.), for every atom of the form DIS(X_i) ($\neg$ DIS(X_i), resp.) occurring in MPred, where the atomic message X_i is timed with the signature $X_i[dt, et, ri, lab2]$;
   e) constraints of the form F[lab3_ri] $\wedge$ CK_lab3_ri $\geq$ et (resp., $\neg$ (F[lab3_ri] $\wedge$ CK_lab3_ri $\geq et$)), for every atom of the form EXP(X_i) ($\neg$ EXP(X_i), resp.) occurring in MPred, where the atomic message X_i is timed with the signature $X_i[dt, et, ri, lab3]$;
5. a is C_CHM_s_m!;
6. λ is a set of assignments containing F[lab_ri] := 1, CK_lab_ri := 0, CK_d_ri := 0, and $N_{ri}[Y_i]$:= m_i for $(Y_i, m_i) \in \theta^a$, with Y_i occurring primed in M.

Notice that constraints in $4.a)$ check that the unprimed variables of the message term sent comply, via θ, with the current knowledge of the role instance; constraints in $4.b)$ check that the delay/timeout constraint is satisfied; constraints in $4.c)$ check that the current knowledge of ri satisfies all the conditions in MPred; and constraints in $4.d)$ (resp., $4.e)$) encode the disclosure (resp., expiration) predicates. Finally, the updates in λ keep track of the occurrence of the event associated to the current transition (F[lab_ri]:=1); reset the clock counting the time passed from the execution of the transition and the clock used to model the duration of the transition; and assign, according to θ^a, the fresh values to the primed variables in the message term sent, thus updating the knowledge of ri.

The general form of a receive timed transition is:

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

Similarly to send transitions, each receive $THLPSL$ transition defines a set of XTA transitions, one for each possible ground message m unifying with M, according to the relation $\Rightarrow_{\rho ri}$. For each $m \in \overline{GM}$, such that $(M, m) \Rightarrow_{\rho ri} \theta, \theta^a$, a pair of XTA transitions, as in Fig. 2, is introduced, where l, l', l'', ϕ and λ are defined exactly as for a send transition, while the transition label a is C_CHM_r_m?.

To guarantee that the duration of a timed (send or receive) transition is modeled correctly, the intermediate location in Fig. 2 is equipped with the invariant $CK\_d\_ri \leq ub$, which forces the second XTA transition to trigger before the time upper bound of the THLPSL transition elapses.

The general form of an urgent transitions is:

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

Notice that an urgent transition cannot send nor receive messages, and that no duration is specified. Therefore, neither synchronization nor update are necessary, and no intermediate location is needed. 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*. A urgent location is a location where time is not allowed to pass, namely where entering and exiting from the location is instantaneous. The guard of the transition is a conjunction of atoms containing the same kinds of constraints defined for cases $4.b)$, $4.c)$, $4.d)$ and $4.e)$, for timed send and receive transitions.

D. Definition of the automaton of the intruder

The automaton for the intruder clearly depends on the intruder model. While THLPSL and TPMC provide different kind of intruder models, for the sake of space, we describe only the strongest form of intruder allowed by the tool, i.e., the Dolev-Yao intruder. A Dolev-Yao intruder can intercept, decompose (provided it has the necessary encryption/decryption keys), forward, block and delay messages. In other words, it has complete control over the communication channel and is usually identified with it. For the sake of presentation, we present an automaton for the intruder which, while being semantically equivalent to the optimized version implemented in TPMC, does not correspond precisely with it. At the end of this section a qualitative discussion of the optimizations implemented in TPMC is given.

The automaton has a single location and four kinds of loop transitions: (i) *sending transitions*, which send messages known by the intruder to some role instance; (iii) *receive transitions*, which receive messages sent by role instances; (iii) *composition transitions*, which freely build new ground messages from known ones; and (iv) *decomposition transitions*, deriving message components from known structured messages.

We need a sending transition for every channel CHN and ground message $m \in \overline{GM}$ unifying the message term in some THLPSL receive transition on CHN. The decoration $\langle \phi, a, \lambda \rangle$ for a sending transition is $\langle \texttt{K[m]} = 1 \land \texttt{CK\_CHN} \geq lb, \texttt{C\_CHN\_r\_m!}, \_ \rangle$, where the triggering constraint checks that the message m is known by the intruder ($\texttt{K[m]} = 1$) and that the minimal channel delay associated to CHN (of type $\texttt{CHN(dy,ub,lb)}$) has passed ($\texttt{CK\_CHN} \geq lb$). Notice that, under the assumption of a DY intruder, the upper bound on the delay of a channel cannot be guaranteed. The label $\texttt{C\_CHN\_r\_m!}$ corresponds to the receiving action associated to CHN for communication from the intruder to the role instances. No update is necessary for this kind of transitions. Similarly, for every channel CHN and ground message $m \in \overline{GM}$ unifying the message term in some

Composition rule	Decomposition rule
$m_1, m_2 \vdash m_1.m_2$	$m_1.m_2 \vdash m_1 \quad m_1.m_2 \vdash m_2$
$m, k \vdash \{m\}_k$	$\{m\}_k, k \vdash m \quad (k \in SK)$
$m, inv(k) \vdash \{m\}_{inv(k)}$	$\{m\}_{inv(k)}, k \vdash m \quad (k \in PK)$
$m, k \vdash \{m\}_k$	$\{m\}_k, inv(k) \vdash m \quad (k \in PK)$
$h, m \vdash h(m)$	

TABLE I.
THE DERIVATION RULE OF A DY INTRUDER.

THLPSL receive transition on CHN, a loop transition for a receive action is introduced, whose decoration is $\langle \_, \texttt{C\_CHN\_s\_m?}, \texttt{K[m]} := 1; \texttt{CK\_CHN} := 0 \rangle$, which is always enabled (no constraint), updates the knowledge of the intruder and resets the clock associated to the channel, which measures the time for channel delay.

Transitions for composition/decomposition of messages encode the standard rules of a DY intruder, which are reported in Table I. The first composition rule states that if the intruder knows two ground messages m_1 and m_2 and $m_1.m_2 \in \overline{GM}$, then it also knows $m_1.m_2$ (and vice versa for the corresponding decomposition rule). The corresponding composition (resp., decomposition) loop transition is decorated by $\langle K[m_1] \land K[m_2], \_, K[m_1.m_2] := 1 \rangle$ (resp., $\langle K[m_1.m_2], \_, K[m_1] := 1; K[m_2] := 1 \rangle$). The second composition (resp., decomposition) rule is for messages encrypted with symmetric keys. The loop transitions for the two rules are decorated by $\langle K[\{m_1\}_k] \land K[k], \_, K[m_1] := 1 \rangle$ and $\langle K[m_1] \land K[k], \_, K[\{m_1\}_k] := 1 \rangle$, respectively. The loop transitions for the other rules are defined similarly.

As previously mentioned, the automaton for the intruder implemented in TPMC is an optimization of the one described above. The idea is to try to reduce the size of the vector $\texttt{K[]}$ for the intruder knowledge, by explicitly storing only the complete ground messages sent or received by role instances, and the atomic ground messages composing them. This avoids to store all the intermediate submessages obtained by freely decomposing a complete message. Clearly, the guards of sending transitions needs to be modified as, in order for the intruder to be able to send a message, a guard needs to check whether the intruder is able to build the message to be sent starting from the atomic components it knows. The advantages are that this makes the encoding of the intruder knowledge much more compact and reduces the number of composition and decomposition rules. On the other hand, the drawback is that it may result in a worst-case exponential blowup in the size of the guards. This is because the guard of a send transition must encode all the possible derivation paths which could lead, according to the rules of Table I, to build that message. This drawback mainly affects the performances of the translation procedure of TPMC, whereas the performances of the verification phase have always improved. This is, in general, an advantage since the translation is usually done once for many verification

sessions, and verification is typically the most expensive phase.

E. Encoding of security goals

The model checker UPPAAL allows to verify properties of XTAs expressed in a fragment of Timed CTL [18]. Therefore, TPMC allows to verify THLPSL specification against properties directly written in that fragment. In this subsection we describe how the security goals of a THLPSL specification are automatically encoded for verification in UPPAAL.

Since the number of ground messages in the protocol can be kept finite, a *secrecy goal* for a message term M is encoded by a CTL formula which checks that, in every reachable state, every ground message, possibly instantiating the message term M, is not contained into the knowledge of the intruder.[1] A *Timed authentication property* is specified in THLPSL using the `Twitness` and the `Trequest` keywords (see Section III for the intuitions). For instance, the timed weak authentication goal for the Wide Mouthed Frog protocol:

```
goal
    Alice Tauthenticates Bob on Kab
end goal
```

together with the clauses `Twitness(A,B,k,Kab)` and `Trequest(B,A,k,Kab)` (see the specification in Section III), requires that a receipt of an instance of message `Kab` by Bob must match with the same instance previously sent by Alice, and this must occur during the validity time interval of term `Kab`. While this kind of property cannot be directly encoded in the fragment of CTL supported by UPPAAL, an equivalent CTL formula can be constructed by introducing, for every possible instance of message `Kab`, three additional Boolean variables: one for the `Trequest`; one for the `Twitness`; and which records the temporal validity of the message instance. A Trequest variable is set to true exactly when the transition issuing the `Trequest` for the corresponding instance of `Kab` is executed, and similarly for a Twitness variable. The variable for the temporal validity of the instance is set to true together with the Trequest variable only if the clock constraint expressing the disclosure and non-expiration of that instance is satisfied. Therefore, timed authentication of Alice and Bob w.r.t. message term `Kab` reduces to a conjunction of implications, one for each instance of `Kab`, each conjunct requiring that the truth of a Trequest variable implies the truth of both the corresponding Twitness variable and the temporal validity variable. Untimed *authentication goal* is similar to Timed authentication, except that reference to the validity of the message term is absent.

---

[1]In the syntax of CTL this is expressed by: $A[] \bigwedge_{m \in \rho(M)} \neg K[m]$, where $\rho(M)$ denotes the set of possible instances of M.

F. Resilient and operational channels

Resilient and operational channels share the property that the delivery of a sent message is always guaranteed. This property belongs to the class of fairness properties and, therefore, cannot be directly expressed in CTL. As a consequence, a suitable encoding involving XTAs is necessary. Let us consider the encoding for resilient channels. In detail, we need to add three boolean arrays:

- `I[]` used to record, for each message/channel pair, whether the message has been sent along that channel and it is potentially available for delivery;
- `R[]` used to record, for each message/channel pair, whether there is a role instance currently able to receive that message along that channel;
- `RSL[]` used to record, for each message/channel pair, whether the resilience property is currently violated.

When a message m is sent along a resilient channel c and the lower bound on delivery delay of c has passed, the location of array `I[]` corresponding to the pair $\langle c, m \rangle$ is set to true. When a role instance is in a state where message m can be received along channel c, the location of array `R[]` corresponding to the pair $\langle c, m \rangle$ is set to true, and it is set back to false as soon as the role instance changes state. Whenever both the location of `I[]` and `R[]` for a pair $\langle c, m \rangle$ are set to true, the corresponding location in the array `RSL[]` is set to true (the resilience property is currently violated). This location of the array `RSL[]` is set back to false when a receive transition is executed by some role instance for message m along channel c (thus witnessing delivery of the message). These updates are suitably encoded by extending the definition of the transitions of the intruder.

Verification of any reachability (security) property ϕ under the assumption of channels resilience reduces to check whether every reachable state satisfy the following property: $(\text{DEADLOCK} \wedge \forall i \neg \text{RSL}[i]) \Rightarrow \phi$. The proposition DEADLOCK is a built-in proposition in UPPAAL and holds only in those states reached by maximal computations, i.e., computations which cannot be extended with further control actions. The intuitive reading of the invariant above is that in all the states, where no further messages can be sent or received and where no message/channel pairs violates the resilience property, the security property ϕ must hold.

As to operational channels, where delivery must be guaranteed within a time upper bound, the encoding is performed similarly to resilient channels. The main difference is that a location of the array `RSL[]` can be reset to zero only if the time when the message is actually delivered is within the upper bound on delivery of the channel.

V. EXPERIMENTS

The verification tool TPMC is implemented in C++ and integrates the compiler from THLPSL specifications

Protocol	Inst	CT	VT
WMF	1-1-3	.01	.34
WMF$_{Fix}$	1-1-3	.01	.01
WMF	5-5-15	.15	628.8
WMF$_{Fix}$	4-4-12	.05	25.83
ZG	1	.14	.04
ZG	3	.27	758.47
ZG Res.	1	.04	.05
ZG Res.	3	.28	1928.3
ZG Oper.	1	.04	.03
ZG Oper.	3	.3	249.9
TESLA	1	.028	1.8

Figure 3.   Experimental results for Timed Protocols (times in seconds).

Protocol	Inst	CT/VT	OFMC	SatMC
NSPK	3	.05 / .33	.09	.85
NSPK	17	18.96 / 55.53	-	25.6
NSPK$_{Fix}$	3	.07 / .04	.09	.9
NSPK$_{Fix}$	5	.17 / 21.67	13.64	1.34
ISO1	2	.01 / .01	.06	.50
ISO1	32	57.40 / 2.72	19.41	-
PBK	2	.03 / .03	.10	.87
PBK	19	9.72 / 344.57	-	55.62
PBK$_{Fix}$	2	.036 / .065	.19	1.022
PBK$_{Fix}$	19	9.36 / 250.25	-	15.66

Figure 4.   Experimental results for Untimed Protocols (times in seconds).

to UPPAAL XTAs with the model checking engine UPPAAL. To assess the efficiency and scalability of the resulting environment, we ran it on a number of timed and untimed protocols. An excerpt of the results of our experiments is given in Figure 3 and Figure 4. The experiments have been performed on a 3.0GHz Pentium IV with 1Gb of memory running Linux (Slackware 11.0). The column Inst. reports the number of protocol sessions for the corresponding test. Both compilation time (CT) and verification time (VT) are reported. Figure 3 reports the tests performed on the following timed protocols: the original and fixed version (as proposed by Lowe [13]) of the Wide Mouthed Frog protocol; three versions of the Zhou-Gollmann protocol, each testing the protocol under different assumptions on the communication channel: unreliable (ZG), resilient (ZG Res), operational channels (ZG Oper), where all but the last one report an attack; the correct version of the TESLA protocol [15]. The column Inst. for the Wide Mouthed Frog protocol reports the number of instances of the participants involved (one for Alice, one for Bob and the last for the Server).

In order to compare with state-of-the-art verification tools for security protocols, we ran our tool on some untimed protocols. Figure 4 compares the experimental results of TPMC with two of the verification engines included in the AVISPA suite, namely OFMC and SATMC. The untimed protocol analyzed (taken from the AVISPA library of protocols [20]) are the following: the Needam–Schroeder Public Key protocol (original and fixed version), the PBK protocol (original and fixed version), and the ISO1 protocol. All the tests are parametric in the number of sessions, where a session involves two participants. The property checked for all the protocols is authentication. On all the tests, our tool correctly reports the expected attack on the flawed versions of the protocols and no attacks for the fixed versions. We only report the results for the minimal and maximal instance of the protocols we tried to analyze (absence of a value for the time spent indicates non-termination within 20 minutes).

The results show that, even though our tool has not been optimized for untimed protocols and the compiler and the model checker are not tightly integrated as in the

competitor tools, the performances are still comparable and, in some cases, scale better as the number of sessions increases. Notice that verification and compilation times (column CT/VT) are usually not correlated. For example in the case of the ISO1 protocol the exchanged messages are highly structured. Therefore, increasing the number of sessions the compiler has to face an exponential growth in the number of messages. The verification time for the same protocol scales much better as the model checker only needs to explore a portion of the state space to find an attack. Timing also plays a fundamental role in the scaling of the compilation/verification procedure. For example WMF is a timed protocol whose message structure is quite simple. The number of messages which can be generated grows slowly as the number of sessions increases. Therefore, the additional work for the compiler is quite limited. On the other hand, the verification time increases rapidly, as the model checker has to work on a Timed Automaton where timing constraints are actually present, while for the ISO1 protocol (and all the untimed protocols) timing is absent.

VI. RELATED WORK

The idea of using formal methods to check time–sensitive security properties is not new (e.g., see [8], [9], [11], [14]). A number of those papers relies on low level formalisms as specification languages.

In [14] the low-level formalism of Timed Automata is extended to allow for the ability to express parallelism and synchronization on structured messages built over cryptographic primitives, providing a more convenient way to model security protocols with Timed Automata. Similarly, in [11] a timed process algebra is proposed as the specification language for security protocols. Both approaches do not provide a high-level specification language comparable to THLPSL.

In [8] the authors use XTAs as a modeling language and give an explicit representation of the intruder as a Timed Automaton and a fine grained representation of cryptography and nonce generations. The resulting model is somehow similar to ours. The main novelties of THLPSL with respect to [8] are that our work builds on top of a specification language already settled and

accepted in the security protocol community, provides the definition of several temporal features and different models of intruder and channels, and integrates the temporal features within a protocol verification tool.

In [9] the authors use constraint programming combined with symbolic exploration to analyze infinite state protocols with explicit use of timestamps (focussing, in particular, on the Wide Mouthed Frog protocol).

VII. CONCLUSIONS

In this paper we presented a verification tool for time–sensitive security protocols. The specification language THLPSL permits explicit modeling of many temporal features of protocols, while remaining sufficiently high level to be used by protocol designers and engineers. The verification engine, based on a translation into Timed Automata, has been tested against a number of timed and untimed protocols, showing encouraging performances.

At the current stage, THLPSL still suffers from some expressiveness limitations, mainly inherited from the underlying semantic model (Timed Automata). For instance, it is not currently possible to specify temporal constraints in a parametric way. This ability would allow to fully specify protocols where temporal constraints are negotiated among the participants in the bootstrapping phase of the protocol, which is quite common in many protocols (for instance, the TESLA and the Zhou-Gollmann protocols). We are currently investigating techniques to overcome these limitations, by employing other formalisms as the semantic model. For instance, we are considering Parametric Timed Automata [3], where some forms of parametric timed constraints are allowed, still preserving the decidability properties essential for verification.

We are also investigating possible strengthening of the Dolev-Yao intruder model, by, e.g., relaxing the perfect cryptography assumption, which forces the intruder to decode encrypted messages only by known keys. Some preliminary work has been done to model an intruder with the ability to derive messages by exploiting the prefix property [16] (formally captured by the derivation rule $\{a, b\}_k \vdash \{a\}_k$), a property exhibited by some encryption algorithms based on cipher-block-chaining (CBC).

Another extension under implementation concerns the possibility to associate time costs to the intruder operations. Indeed, a DY-intruder model assumes an intruder with unbounded computational power. By allowing constraints on its computational power and time costs associated to actions, TPMC would allow to model and verify more realistic scenarios.

REFERENCES

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

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

[3] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1993.

[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 FCS-ARSPA'06, Seattle, August 15–16, 2006.

[6] M. Benerecetti and N. Cuomo and A. Peron, TPMC: A Model Checker For Time–Sensitive Security Protocols. In proc. of HPCS'07, June 4th - 6th, 2007, Prague, Czech Republic, pp. 742–749, 2007.

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

[8] R.J. Corin, S. Etalle S., P.H. Hartel, and A.H. Mader. Timed analysis of security protocols. *Journal of Computer Security*, 15(6):619–645, 2007.

[9] G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In *Proc. of TACAS 2004, Barcelona, Spain*, pages 342–356, 2004.

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

[11] R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for real-time cryptographic protocol analysis. In *ESOP: 12th European Symposium on Programming*, 2003.

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

[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] M. Napoli, M. Parente, and A. Peron. Specification and verification of protocols with time constraints. *Electr. Notes Theor. Comput. Sci*, 99:205–227, 2004.

[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. Rusinowitch. Automated analysis of security protocols. *Electr. Notes Theor. Comput. Sci*, 86(3), 2003.

[17] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *TCS: Theoretical Computer Science*, 299, 2003.

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

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

[20] L. Viganò. Automated security protocol analysis with the AVISPA tool. *Electr. Notes Theor. Comput. Sci*, 155:61–86, 2006.

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