DOTTORATO DI RICERCA
in
SCIENZE COMPUTAZIONALI E INFORMATICHE
Ciclo XXII

Consorzio tra Università di Catania, Università di Napoli Federico II,
Seconda Università di Napoli, Università di Palermo, Università di Salerno

SEDE AMMINISTRATIVA: UNIVERSITÀ DI NAPOLI FEDERICO II

---

NICOLA CUOMO

# VERIFICATION OF TIMED SECURITY PROTOCOLS

---

*TESI DI DOTTORATO DI RICERCA*

`

IL COORDINATORE
Prof. Luigi M. Ricciardi

**Università degli Studi di Napoli**
**"Federico II"**

DOTTORATO IN SCIENZE COMPUTAZIONALI ED INFORMATICHE
XXII CICLO

# Verification of Timed Security Protocols

Nicola Cuomo

Thesis Supervisor:
**Prof. Adriano Peron**
Tutor:
**Prof. Piero Bonatti**
Coordinator:
**Prof. Luigi M. Ricciardi**

---

**November 2010**

# Abstract

Security protocols are communication protocols that aim at enforcing security properties through heavy use of cryptographic primitives. These protocols are at the core of security-sensitive applications in a variety of domains (e.g. transportation, health-care, online banking and commerce). Failures are not a option as may cause heavy loss of capitals, time and even humans life. In spite of their apparent simplicity, security protocols are notoriously error-prone and so a number of verification techniques were developed to cope with the verification of such protocols. However 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 thesis we face the problem of specifying and verifying security protocols where temporal aspects explicitly appear in the description. For these kinds of protocols we have designed a specification formalism, which consists of a state-transition graph for each participant of the protocol, with edges labelled by trigger/action clauses. The specification of a protocol is translated into a Timed Automaton on which standard techniques of model checking can be exploited (properties to be checked can be expressed in a linear/branching untimed/timed temporal logic). We also study the protocol insecurity problem for time dependent security protocols with a finite number of sessions, extending to the timed case the results of M. Rusinowitch and M. Turuani [RT03] stated for the untimed case. We show that the extension

to time and the increased power of the intruder model we propose do not affect the complexity of the problem which remains NP–Complete.

# Preface

## Declaration

I declare that this thesis was composed by myself, and that the work contained herein is my own except where explicitly stated otherwise in the text. Besides this, I hereby declare that this work has not been submitted for any other degree or professional qualification except as specified.

## Publications

Some of the work in this thesis has previously been published in [BCP06, BCP07, BCP09a, BCP09b, BCP10].

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The constant growth of dependency of many human activities on computer systems and applications has made fundamental their verification phase. Malfunctions, still largely tolerated in personal computing systems, is unacceptable for *safety critical* systems whose failure or malfunction may result in: death or serious injury to people, or loss or severe damage to equipment or environmental harm. The usual verification techniques known as testing, debugging or simulation usually cover only a fraction of the admissible behaviours of the system and, while widely used and understood, do not guarantee the degree of confidence required by safety critical systems. Disasters like the explosion of Ariane 5 (due to a floating point overflow), the Pentium FDIV bug or the death of two people caused by the software controlled medical machine for radiotherapy Therac25 shows that conventional validation techniques based on informal arguments and/or testing are not adequate. Complementing those verification techniques *formal methods* have been profitably used in various phases of the design of safety critical systems. Such techniques can mathematically prof that a system conform to its safety requirements. There are roughly two approaches to formal verification, *Theorem proving* and *Model Checking*. Theorem proving consists in the encoding of the system and its requirement in the form of some logic in the attempt to build a formal proof of the safety of the system. This is usually only partially automated and is driven by the user's understanding of the

system to validate. Model checking, otherwise, is more easily automated. The user provide a representation of the system using an appropriate Labelled Transition System (LTS), usually some kind of finite state automaton, and a formula, usually in some temporal logic (LTL, CTL), representing the safety goal to satisfy. The model checking algorithm search the state space of the system in the attempt to violate the goal formula eventually providing the user with a counterexample. While the inherent complexity of both techniques somehow limit they adoption and problems like the one of *state explosions* limit their use to small systems, both theorem proving and model checking were effectively used to verify safety critical systems.

A notable class of *safety critical* systems is the one of security, or cryptographic, protocols. Our society rely on electronic communication massively. Although much faster and less restrictive than direct communications, those new kinds of communications seems to lack the trust and safety of usual direct communications. For example, a third person can easily listen to a phone communication (problem of confidentiality), or, without proper authentication, send emails on behalf of someone else (SPAM, viruses). Security protocols try to ensure some degree of safety by the use of cryptographic primitives. However, although absolutely necessary to any safe communication, encryption algorithms are not sufficient to guarantee safety. Designing secure protocols is a very challenging problem, a number of examples have shown that their informal design is error prone. In wost case scenarios, the presence of a motivated intruder or dishonest or careless principals, severe attacks can be conducted even without breaking cryptography.

Formal methods have been profitably used in various phases of the design of cryptographic protocols (specification, construction and verification). Much work has been then devoted to formal specification and analysis of cryptographic protocols, leading to a number of different approaches and encouraging results. 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 [ABB+05]), 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 thesis 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 and how time affect the complexity of the verification. Examples of time sensitive protocols are, for instance, the non-repudiation Zhou-Gollmann protocol [ZG97], the TESLA authentication protocol [PCTS02] and the well known Wide Mouthed Frog protocol [BAN89].

## 1.1 Thesis Structures

This thesis is divided in the following main chapters:

- Chapter 2 - where we set the basic formal concepts this thesis build on. In this chapter are defined the concepts of timed automaton, model checking, and temporal logics;

- Chapter 3 - where we define what is a security protocol and how can be modelled and verified using the industry standard HLPSL language;

- Chapter 4 - where we define what is a timed security protocols. How this class of protocols relate with the untimed ones and the challenges it pose for a correct specification. We will present three timed protocols and an extension to the HLPSL language that allow for a easy specification of temporal constraints. We will end with the language formal semantics;

- Chapter 5 - where we will describe a framework for the verification of timed security protocols based on the extended language. We will

verity the presented protocols and show the framework performances compared to some alternatives;

- Chapter 6 - where we will focus on the computational complexity of the addressed problem and how it relate to previous bibliography results;

- Chapter 7 - where we summarise the conclusions of this thesis and future directions that may be worth exploring.

# Chapter 2

# Formal Background

The purpose of this chapter is to set the basic formal concepts this Thesis focuses on. The chapter presents the formalism of the Timed Automata in section 2.1 and the one of the Difference Logic in 2.7. In section 2.4 are presented decidability results related to the model checking of both those formalisms.

## 2.1 Timed Automata

Timed systems, like device drivers, ATM, communication protocols, are systems whose behaviour and dynamics are dependent on time. Subject to rather stringent timing constraints those systems must react in time: they are time-critical.

**Example 2.1.1** *For a system controlling a gate is essential to close within a certain time bound after detecting the approaching train to halt car and pedestrian traffic before the train reaches the crossing.*

Many different formalisms are used to model timed system, both in continuous-time and discrete-time. Between those formalism the one of Timed Automata, introduced by [AD94], have received much interest in the past years. There are many extension of that formalism accounting many peculiar featured and enjoying different decidability properties.

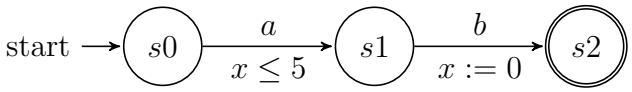



Figure 2.1: An Example of Timed Automaton

We begin by giving the intuition behind the model before a more formal insight of its syntax and semantic.

Consider the automaton depicted in figure 2.1.

Much like an usual Büchi Automaton we have a some states (s0, s1, s2), a starting (s0) and an accepting state (s2), we have transitions triggered by symbols of an alphabet $\Sigma = \{a, b\}$; however, differently from the usual untimed models, transitions are also decorated ($x := 0$ and $x \leq 5$ ) using constraints and resets over a set of non negative real valued variables called clocks. Intuitively, as the automata *stay* in a location, the value of all the clocks grows synchronously from the initial value of 0. Transitions firing is instantaneous and conditions (guards) can check for the value of those clock variables or reset them. In our example the clock $x$ in the automaton is checked before taking the transition between $s0$ and $s1$ to see if its value is less than 5 (i.e.: 5 time units passed from the last reset of the clock) and is later resetted to 0 when taking the transition between $s1$ and $s2$. Generally we can have an arbitrary, but finite, number of clocks in our automaton and each transition can check and reset the value of more clocks at once. The syntax of checks performed, the kind of resets allowed and, moreover, features like the ability to constraint the time passed in a location (via guards like constraints called invariants), synchronisation between different automata, etc.., determine the particular timed automata extension used and its decidability properties.

Timed Automata accept sequences of symbols paired with a non-negative real value called *timed words*, intuitively the time that symbol was accepted. For example the timed automaton in figure 2.1 can accept the sequence $(a, 3), (b, 2)$.

We now present formally the Timed Automata syntax as defined in [AD94].

### 2.1.1 Syntax

Given a finite set of clocks $\mathcal{C}$ valued over a dense domain (eg.: $\mathcal{R}_{\geq 0}$) and a finite set of symbols $\Sigma$.

The set of clocks constraints over $\mathcal{C}$, $\Phi(\mathcal{C})$ is built using boolean combinations of atomic constraints of the form $x\#c$ with $x \in \mathcal{C}$, $\# \in \{=, \leq, <, \geq, >\}$ and $c \in \mathcal{Q}_{\geq 0}$.

Inductively $\Phi(\mathcal{C})$:

- $(x \leq c) \in \Phi(\mathcal{C})$, with $x \in \mathcal{C}$ and $c \in \mathcal{Q}_{\geq 0}$;

- $(c \leq x) \in \Phi(\mathcal{C})$, with $x \in \mathcal{C}$ and $c \in \mathcal{Q}_{\geq 0}$;

- $\neg\phi \in \Phi(\mathcal{C})$ if $\phi \in \Phi(\mathcal{C})$;

- $(\phi_1 \bigwedge \phi_2) \in \Phi(\mathcal{C})$ if $\phi_1 \in \Phi(\mathcal{C})$ and $\phi_2 \in \Phi(\mathcal{C})$.

Given a set of clocks $\mathcal{C}$ with their values being in $\mathcal{R}_{\geq 0}$, a clocks valuation $\nu$ is a function $\mathcal{C} \to \mathcal{R}_{\geq 0}$ that associates to each clock $x \in \mathcal{C}$ its value in $\mathcal{R}_{\geq 0}$, $\nu(x)$.

A clocks valuation $\nu$ satisfies an atomic constraint $(x\#c)$ if and only if, using the usual semantic of the constraints, $(\nu(x)\#c)$ is true.

We denote with $\nu \models g$ the fact that the clocks valuation $\nu$ satisfies a timing constraint $g$.

Given a clocks valuation $\nu$ over $\mathcal{C}$, for each $t \in \mathcal{Q}_{\geq 0}, \nu + t$ is the valuation that assign to each clock $x \in \mathcal{C}$ the value $\nu(x)+t$. Moreover given two subsets $X, Y \subseteq \mathcal{C}$ and $Y \subseteq X$ with $\nu' = [Y \mapsto t]\nu$ we denote the clocks valuation for $X$ such that $\forall y \in Y \nu'(y) = t$ and $\forall x \in X - Y \nu'(x) = \nu(x)$.

A finite state timed automaton over the symbols alphabet $\Sigma$ and the set of clocks $\mathcal{C}$ is a tuple

$$\langle \Sigma, L, L_0, \mathcal{C}, \delta \rangle$$

where,

- $\Sigma$ is the finite set of symbols;

- $L$ is a finite set of locations;

- $L_0 \subseteq L$ is the set of initial locations;

- $\mathcal{C}$ the set of clocks;

- $I : L \to \Phi(\mathcal{C})$ the invariant map, associating to each location an invariant guard, timely constraining the ability of the automaton to stay in a location;

- $\delta \subseteq L \times L \times \Sigma \times 2^{\mathcal{C}} \times \Phi(\mathcal{C})$ is the transition function. An element of $\delta$, $\langle s, s', a, \lambda, \phi \rangle$, also written as $s \overset{a,\lambda,\phi}{\to} s'$, represent the transition between the location $s$ to the location $s'$ upon the receipt of the input symbol $a$. The set $\lambda \subseteq \mathcal{C}$ is the set of the clocks resetted to 0 upon the firing of the transition while $\phi \in \Phi(\mathcal{C})$ is the guard of the transition (i.e.: the condition that must be true before the transition take place).

The timed automaton depicted in figure 2.1 can be defined as follow:

- $\Sigma = \{a, b\}$;

- $L = \{s_0, s_1\}$;

- $L_0 = \{s_0\}$;

- $C = \{x\}$;

- $I = \emptyset$;

- $\delta = \{(s_0, s_1, a, x, \emptyset), (s_1, s_2, b, \emptyset, x \leq 5)\}$

A *time sequence* $\tau = \tau_1 \tau_2 \ldots$ is an infinite sequence of clock values with $\tau_i \in \mathcal{Q}_{\geq 0}$ satisfying the following constraints:

- Monotonicity: the values of $\tau$ grown monotonically; $\forall i, t_i \leq t_{i+1}$;

- Progress: $\forall t \in \mathcal{Q}_{\geq 0}\ \exists i \geq 1 : \tau_i > t$.

As we said a *timed word* on the $\Sigma$ alphabet is a pair $(a, \tau)$ where $a = a_1 a_2 \ldots$ is a infinite sequence of symbols of $\Sigma$ and $\tau$ is a time sequence. The

set of the timed word accepted by an automaton $A$ is the language accepted, $\mathcal{L}(A)$, by that automaton.

A run $r$ over a timed automaton is an infinite sequence like:

$$r : \langle s_0, \nu_0 \rangle \overset{(a_1,\tau_1)}{\rightarrow} \langle s_1, \nu_1 \rangle \overset{(a_2,\tau_2)}{\rightarrow} \langle s_2, \nu_2 \rangle \ldots$$

with $s_i \in L$ and $\nu_i$ clock valuation satisfying the following conditions:

- Initialisation: $s_0 \in L_0$ e $\forall x \in \mathcal{C}, \nu_0(x) = 0$;
- Sequentiality: $\forall i \geq 1$ there is a transition $\langle s_{i-1}, s_i, a, \lambda, \phi \rangle$ in $\delta$ and $\nu_i$ is equal to $[\lambda_i \mapsto 0](\nu_{i-1} + \tau_i - \tau_{i-1})$.

Acceptance for a timed automaton is defined like the one over generalised *Büchi* automaton: defined a subset $F \subseteq L$ of accepting states a run $r$ is accepting on a timed word it passes through at least one state of every set of accepting states infinitely often; denoting with $inf(r)$ the set of infinitely recurring states in $r$ i.e: $\cap F \neq \emptyset$.

### 2.1.2 Semantic

The semantic of a timed automaton $A = \langle \Sigma, L, L_0, \mathcal{C}, I, \delta \rangle$ can be defined using an infinite *Labelled Transition System* (LTS), $TS_A$.

Intuitively each state in $TS_A$ is a pair $\langle l, \nu \rangle$, called *instantaneous description*, with $l$ location of the timed automaton and $\nu$ a clock valuation.

Formally the LTS associated to an automaton $A = \langle \Sigma, L, L_0, \mathcal{C}, I, \delta \rangle$ is a tuple $TS_A = \langle S, s_0, \rightarrow, \Sigma \rangle$ where:

- $S = L \times \mathcal{R}_{\geq 0}$;
- $s_0 = (l_0, v_0)$ with $l_0 \in L_0$ and $v_0(x) = 0 \forall x \in \mathcal{C}$ is the clock valuation that assign 0 to each clock;
- $\rightarrow$: $S \times S \times \Sigma$ the transition relation. We have:
  - action transitions: $(l, v) \rightarrow^a (l_0, v_0)$ if and only if there exists $e = (l, g, a, r, l_0)$ such that $v \models g$, $v_0 = [r \mapsto 0]v$ and $v_0 \models I(l_0)$

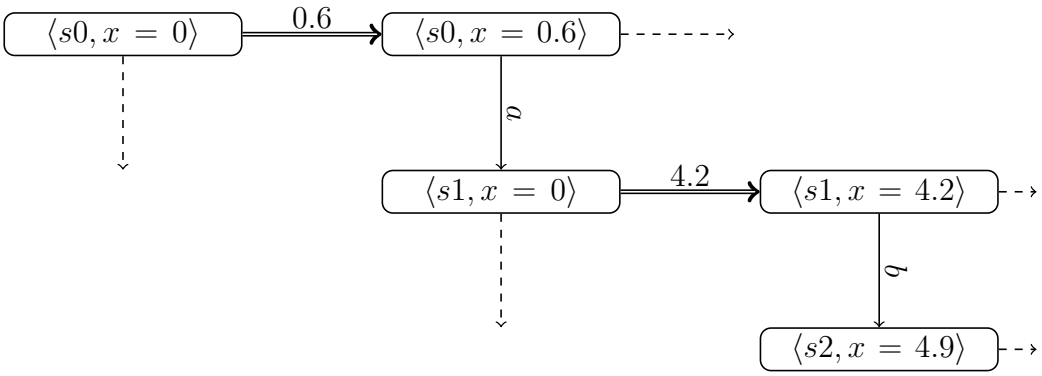



Figure 2.2: The LTS associated to the automaton of figure 2.1

i.e.: there is a transition from $l$ to $l_0$ on the symbol $a$ whose guard is satisfied by $v$ while the valuation $v_0$ obtained by $v$ resetting the clocks in $r$ satisfy the invariant of $l_0$;

- time transitions: if $d \in \mathcal{R}_{\geq 0}$ then $(l, v) \rightarrow^d (l, v + d)$ if and only if $v + d \models I(l)$.

An important remark is that due to the underlying clocks dense domain two consecutive time transition can be merged, e.g.: the two consecutive transition $(q, v) \rightarrow^t (q, v' = v + t) \rightarrow^{t'} (q, v' + t')$ are equivalent to the transition $(q, v) \rightarrow^{t+t'} (q, v + t + t')$. Inversely a time transition $(q, v) \rightarrow^t (q, v + t)$ can be decomposed in an arbitrary number of consecutive time transitions $(q, v) \rightarrow^{t'} (q, v) \rightarrow^{t_1} \ldots \rightarrow^{t_n} (q, v + (t_1 + t_n)) \rightarrow^{t'}$ for a suitable choice of $t_1$, $t_n$.

As an example in figure 2.2 is the associated LTS of the timed automaton in 2.1, double lines represent time transitions, single lines are action transition and dashed lines represent transitions to not showed states.

### 2.1.3 Timed Automata Related Results

In the seminal paper [AD94], the authors prove a number of properties regarding timed automata. In detail:

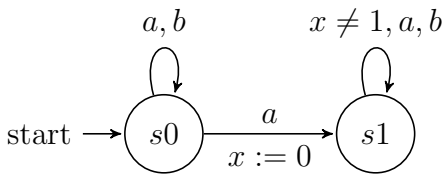



Figure 2.3: A non complementable Timed Automaton

### Closure Properties

Timed automata are closed under the operation of union and intersection but, differently from the Büchi Automata, not complementation. The closure for union and intersection come as a direct consequence of the non deterministic nature of the timed automaton. The non deterministic nature of timed automata is also the cause of its non closure under complementation. The proof in [AD94] is based on the observation that, due to the non determinism of the timed automaton, a timed word can have an execution ending in both a final and a non final location making impossible to build the complement for a timed automata. For example the timed automaton in figure 2.3 is not complementable [AD94].

### Universality, Timed Language Inclusion, Equivalence and Emptiness Problems

The universality problem, checking if given an alphabet $\Sigma$ an automaton accept all the words over it, for timed automata is undecidable. The prof come by reduction from the undecidable problem of halting of 2-counter machines. The problems of timed language inclusion, i.e.: $\mathcal{L}(A) \subset \mathcal{L}(A')$, and equivalence, i.e.: $\mathcal{L}(A) = \mathcal{L}(A')$ is also undecidable since proving them require the ability to complement a timed automata. The emptiness problem, i.e.: the automaton recognises the empty language, it never reach an accepting location, is PSPACE-Complete.

| Problem | Results |
|---|---|
| Union | Closed |
| Intersection | Closed |
| Projection | Closed |
| Complementation | Not closed |
| Language inclusion | Undecidable |
| Language equivalence | Undecidable |
| Universality | Undecidable |
| Language emptiness / reachability analysis | PSPACE-Complete |

Table 2.1: Timed Automata Decidability Problems

### 2.1.4 Timed Automata Classes and Extensions

A number of works stemmed from the [AD94] paper trying to find useful subsets or extensions to the timed automata formalism obtaining better decidability results or more expressive formalisms. We summarise the principal results, a more thorough work is available in [BP09].

#### Deterministic Timed Automata

The original definition of the Timed Automata was inherently non deterministic, but differently from untimed automata, a source of non determinism source could be the transition guards. In [AD94] the authors define the class of deterministic timed automata restricting the definition by adding the constrains:

- there is only a single initial location;
- two transitions from the same location on the same input symbol must have their guards disjoint.

Adding the above constrains effectively remove the non determinism caused by the transition guards. The Deterministic Timed Automata are a less expressive subclass of (non deterministic) Timed Automata. That also mean that the problem of determinizing a Timed Automaton is undecidable (i.e: there is no procedure that accepts a non deterministic timed automaton and

| Class or extension | Emptiness checking | Language inclusion |
|---|---|---|
| Timed automata | PSPACE-Complete | Undecidable |
| Deterministic timed automata | PSPACE-Complete | Decidable |
| Event-clock automata | PSPACE-Complete | Decidable |
| Robust timed automata | PSPACE-Complete | Undecidable |
| $\epsilon$-transitions without clocks resets | PSPACE-Complete | Undecidable |
| $\epsilon$-transitions with clocks resets | PSPACE-Complete | Undecidable |
| Diagonal constraints $(x - y \sim c)$ | PSPACE-Complete | Undecidable |
| Additive constraints $(x + y \sim c)$ | Decidable for 1 or 2 clocks, open problem for 3 clocks and undecidable starting from 4 clocks[BD00] | Undecidable |
| Constraints of the form x = 2y | Undecidable[AD94] | Undecidable |
| Constraints with irrational constants | Undecidable[Mil00] | Undecidable |
| Non-standard $(x := 0)$ clocks resets | Decidable for $x := c$, undecidable for $x := x-1$ and decidable for $x := x + 1$ if diagonal constraints are not allowed[BP09] | Undecidable |

Table 2.2: Timed Automata Extensions

returns a deterministic timed automaton that recognise the same timed language). The problem of language inclusion is however decidable.

### Event-Recording Timed Automata

The event-recording automata [AFH99] are a class of Timed Automata that contains, for every input symbol $a$, a clock, $C_a$, that records the time of the last occurrence of $a$. The fundamental property in event-recording automata

is that the value of clocks only depends on the input word. This characteristic make this formalism determinizable and closed under all boolean operations while expressive enough to model timed transition systems. Being closed under all boolean operations make also the language-inclusion problem decidable.

### Robust Timed Automata

The ability to measure precise time constraints is probably the main source of undecidability and non closure under complementation [AD94]. As no real world system can be expected to be as precise as Timed Automata expectations, Robust Timed Automata [GHJ97] relax its time constraints and recognises timed words with some fuzziness in the event. Unexpectedly, the authors believed that the removal of real-time equality constraints would lead to a decidable theory that is closed under all boolean operations like what happen in temporal logic, however the Robust Timed Automata still cannot be determinized.

### Bounds and Extensions of the Timing Constraints, Resets, Transitions

Many authors have studied how the kind of timing constraints, reset and transitions allowed effect the decidability properties of the Timed Automata. For example silent, $\epsilon$-transition, (i.e.: transition triggered by no symbol), that in the case of untimed automata do not add to the expressiveness and can be removed easily, strictly adds to the expressiveness of the Timed Automata and cannot be easily removed ([BPDG98]). In [DGP97] the authors show a complex procedure to remove $\epsilon$-transitions when they do not reset clocks. In [AD94, BDFP04, BD00] the authors show how constraints of the form $x = 2y$, additive constraints (i.e.: $x + y \sim c$), diagonal constraints (i.e.: $x - y \sim c$), constraints with irrational constants, and non-standard clocks resets (i.e., $x := c$), affect the the closure and decidability of an automaton. Table 2.2 summarise some of the results.

### 2.1.5 Other Timed Formalisms

Timed Automata, while widely studied, are only one of the many formalism used to model timed system. Others used formalism for modelling timed systems are Timed Petri Nets and Temporal Logics. Timed Petri Nets [Ram74] are an extension of the traditional Petri Nets where a transition can be red only if its enabling duration is in a certain time window. In [BHR06] the authors, proving that timed Petri nets are not more expressive than Timed Automata, introduces a class called Read-arc timed Petri nets which is language-equivalent to timed automata. Temporal Logics even if usually used for formal verification purposes are sometimes used for modelling timed systems, for example the Temporal Logic of Action (TLA) [Lam90]. We will give more details about Temporal Logics in a following section.

## 2.2 Extended Timed Automata

While the expressiveness of Timed Automata is enough to model even complex timed system trying to model by hand even a simple system can be a daunting task. There are many extension of the Timed Automata framework that, while keeping the same decidability/closure results, add features to simplify the creation of models. In particular the Extended Time Automata (XTA) [BY04] add to the TA formalism features like parallelism and synchronisation, integer variables, urgent transitions.

### 2.2.1 Parallelism and Synchronisation

In the XTA framework a network of Timed Automata $A$ is the parallel composition $A_1 \parallel \ldots \parallel A_n$ of a series of Timed Automata $A_1, \ldots, A_n$, sometimes called *process*, combined as a single system using parallel composition in in the style of the Calculus of Communicating Systems (CCS)[Mil89].

Automata communicate by means of channels and the synchronous communication style is handshaking. Let $\Sigma$ be the set of communication channels, then the symbol $a?$ denotes the receiving action over channel $a \in \Sigma$, while

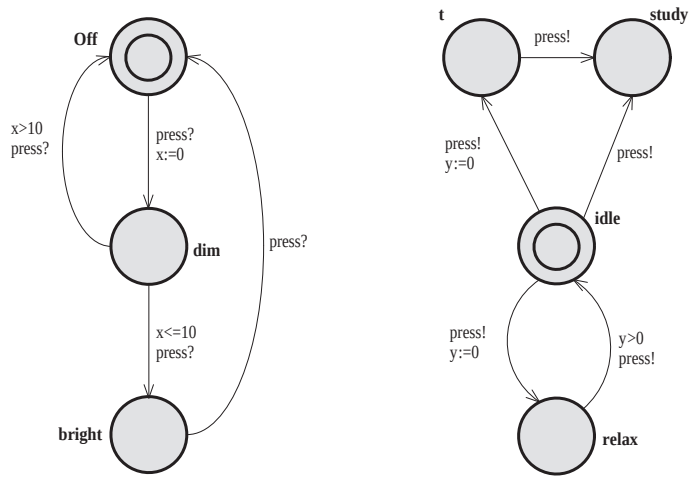



Figure 2.4: An XTA modelling the interaction between a light and its user

the symbol $a!$ denotes the sending action over channel $a$. Moreover, internal actions are denoted by an additional symbol $\tau$.

An example of a network of Timed Automata, modelling the interaction between a light and its user, is in figure 2.4.

The classical Timed Automaton resulting from the parallel composition of the above network is the closed system represented in figure 2.5.

The semantics is again given by means of labelled transition systems. A state of the LTS $TS_{A_1,\ldots,A_n}$ is a pair $< l, \nu >$, where $l$ and $\nu$ are vectors of current locations and clock valuations, respectively, one for each TA in $\{A_1, \ldots, A_n\}$. In XTA we distinguish between two kinds of transition: delay transitions and discrete transitions. The rule for delay transition is similar to the case of a single TA, except that the invariant of a location is the conjunction of the location invariant of all the parallel components. There are two rules for discrete transitions defining local actions, where one of the components makes a move on its own, and synchronisation actions, where two components synchronise on a channel and move simultaneously. Formally:

*Delay Transitions:* $\langle l, \nu \rangle \xrightarrow{d} \langle l, \nu + t \rangle$ if $\nu$ satisfies $I(l)$ and $(\nu + d)$ satisfies $I(l)$, where $I(l) = \bigwedge_{i=1}^{n} I(l_i)$;

*Internal Transitions:* $\langle l, \nu \rangle \xrightarrow{\tau} \langle l[l'_i/l_i], \nu' \rangle$, if $l_i \xrightarrow{g,\tau,\lambda} l'_i$, $\nu$ satisfies $g$, $\nu' =$

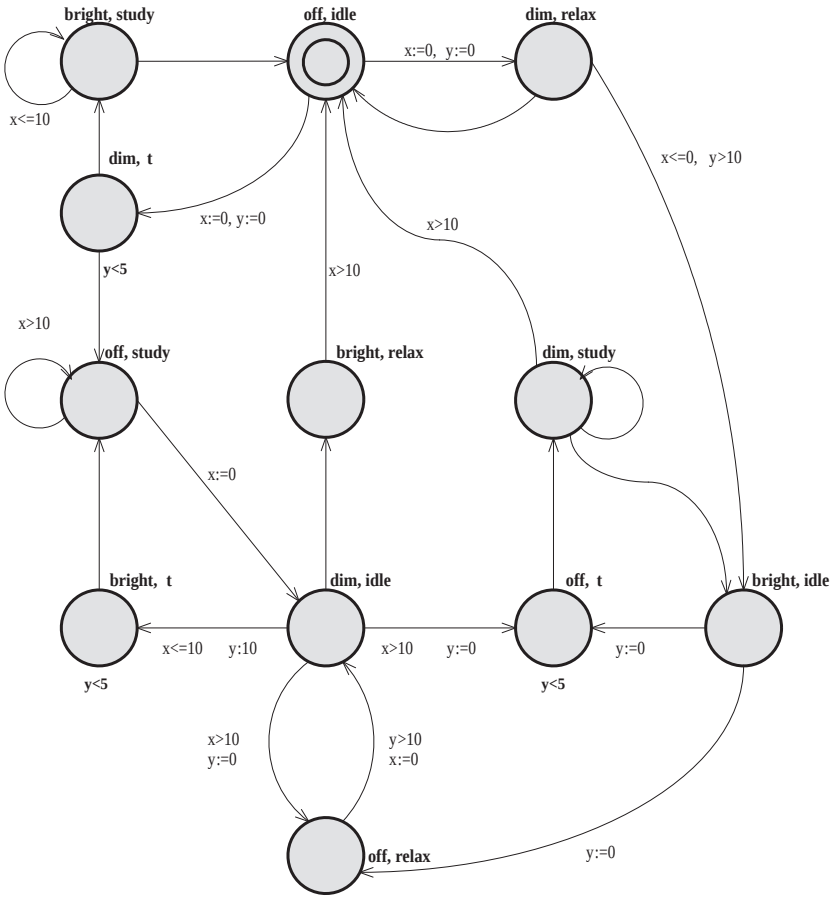



Figure 2.5: The Timed Automaton resulting from the parallel composition of the network in figure 2.4

$\nu[\lambda := 0]$, $\nu'$ satisfies $I(l[l'_i/l_i])$;

*Synchronisation Transitions:* $\langle l, \nu\rangle \xrightarrow{\tau} \langle l[l'_i/l_i][l'_j/l_j], \nu'\rangle$, if there exists $i \neq j$ such that:

1. $l_i \xrightarrow{g_i, a?, \lambda_i} l'_i, l_j \xrightarrow{g_j, a!, \lambda_j} l'_j$ and $\nu$ satisfies $g_i \wedge g_j$, and
2. $\nu' = \nu[\lambda_i \cup \lambda_j := 0]$ and $u'$ satisfies $I(l[l'_i/l_i][l'_j/l_j])$.

Is important to note that the system progress only when the conjunction of all the invariants is verified.

### 2.2.2 Integer Variables

Beside the usual clocks variables XTA allow the use of bounded, initialised, integer variables. Integer variables con have a local or network global scope and transition predicates are augmented to allow to check those variables and change their values.

Semantically, Integer Variables are handles like some kind of non increasing clocks. The clocks valuations $\nu$ are extended accordingly as the LTS rules. The only ambiguity come from the ability to change the value of the same global integer variable on both the transition of a synchronisation transition. This case is explicitly handled by choosing a priority in the update order, i.e.: first execute the update on the output side of the synchronisation transition and then on the input side.

LTS rules are extended in the following way:

- $\langle l, \nu\rangle \xrightarrow{\tau} \langle l[l'_i/l_i][l'_j/l_j], \nu'\rangle$ if there is $i \neq j$ such that

1. $l_i \xrightarrow{g_i, a?, r_i} l'_i, l_j \xrightarrow{g_j, a!, r_j} l'_j$ e $\nu \in g_i \wedge g_j$, e
2. $\nu' = [r_i \mapsto 0]$ $([r_j \mapsto 0]\nu)$ e $\nu' \in I(l[l'_i/l_i][l'_j/l_j])$

### 2.2.3 Urgent Transitions

To force a *strong time* decisions, i.e.: to force an automaton to leave a state as soon as possible without enforcing an invariant, XTA allow for the definition

of *urgent transitions*. An automaton will not delay execution while in a state with an enabled outbound urgent transition.

## 2.3 Model Checking

Model checking is a powerful and automatic technique for verifying finite state concurrent systems. Introduced by Clarke et al. [CGP99] has been applied widely and successfully in practise to verify digital sequential circuit design and communication protocols and it has been integrated in the quality assurance process of several major hardware companies.

Wide research field in computer science it use a number of different methods for solving the general model-checking problem:

$$M \models p$$

Where $M$ is a model of system, usually in the form of a finite state transition system, and $p$ is a logic formula, usually of a temporal logic, expressing some desired requirement. The technique involved in answering that question depends on the particular modelling language used to model the system and the associated temporal logic and in a later section we will detail some of those technique. A problem that is inherently present when using model checking technique is the one of state explosion. The use of finite state systems force the explicit representation of all the system state and that usually mean that the number of system states grows exponentially with the number of system components. This problem severely limited the size the application of model checking to designs with less than one million states (e.g. an hardware circuit designs with at most 20 logic gates).

A partial solution to the problem was proposed by K. L. McMillan [McM93]. The proposed idea, called *symbolic model checking*, was based on the symbolical exploration of the state space through the use of *Binary Decision Diagrams* (BDDs) whose allow computation of transition among sets of states rather than individual states. Symbolic model checking allowed to verify system with up to $10^{20}$ states. While addressing the state explosion problem,

symbolic model checking itself lacked a certain robustness. The problem lied in the fact that BDDs may grow exponentially, limiting to the amount of available memory the size of the system to verify. Moreover BDDs are very sensitive to how the particular system is modelled (and trying to find the best encoding of a state space as BDDs is NP-Complete). In 1999 Biere et al. [BCCZ] proposed a technique called Bounded Model Checking (BMC), which uses a propositional SAT solver rather than BDDs manipulation techniques. Exploiting the dramatic speed-up of propositional solvers it is able to analyse designs with million of states. Moreover bounded model checking has been proved to be particularly suited in finding counter-examples, i.e.: to return paths through the transition system that violate one of the specific system requirements.

## 2.4 Temporal Logics

Timed Automata, enjoying decidable reachability properties, are a well-established model to verify real-time systems. However they are not usable to represent property of systems in fact one of the most common technique to verify a system is through the so called *Test Automaton*. The technique revolve around the construction of a (timed) automaton representing the property, i.e.: all the desired behaviour, that we want, and then checking if all the behaviour of the test automaton are in the our system. This is an inclusion question, and that problem is unfortunately undecidable for timed automata.

From this problem the idea to extend classical untimed temporal with timing constraints creating a formalism, the temporal logics, to express timed properties. Traditionally temporal logics were better suited speak about the relative order of events, not about the distance (in time) between these events.

There are two branches of temporal logics:

- linear-time temporal logics allowing reasoning over a single time line;
- branching-time temporal logics allowing reasoning over several time lines.

| Logic | Model-checking problem |
|---|---|
| TCTL | PSPACE-Complete |
| MTL over finite runs | Decidable under the pointwise semantics<br>Undecidable under the continuous semantics |
| MTL over infinite runs | Undecidable under pointwise semantics |
| TPTL over infinite runs | Undecidable under the pointwise and continuous semantics |

Table 2.3: Temporal Logics Decidability Results

The traditional linear-time temporal logis is the Linear Temporal Logic (LTL) [Pnu77], while the traditional branching-time temporal logics is the Computational Tree Logic (CTL) [CES86].

Both CTL and LTL have been extended to support quantitative time, CTL giving birth to the Timed Computational Tree Logic (TCTL) [HNSY94] and LTL to the Metric Temporal Logic (MTL) [Koy90] and Timed Propositional Temporal Logic (TPTL) [RT94].

The branching-time logic TCTL has a rather low complexity, and offer very good decision properties. The linear-time timed temporal logics, instead, while being more expressive, an interesting properties for writing specifications, are more complex and offer worse decidability results. In details the table 2.3 summarise the main result regarding temporal logics.

Since it will be used in the following chapters we will give a more detailed definition of the TCTL logic.

### 2.4.1 Timed Computational Tree Logic

Given a set of atomic proposition $AP$ the syntax of TCTL is given by the following grammar:

$$\varphi ::= a|\neg\varphi|\varphi_1 \vee \varphi_2|E\varphi_1 U_{\mathcal{I}}\varphi_2|A\varphi_1 U_{\mathcal{I}}\varphi_2$$

where $a \in AP$ and $\mathcal{I}$ an integral interval of $\mathcal{R}_{\geq 0}$.

Given a Timed Automaton $TA$ whose locations are encoded as atomic proposition (i.e: there is a function between the set of locations $L$ and $AP$

such that $L \to 2^{AP}$), and a run $r$ on it $\langle s, \nu \rangle$. The semantic of the TCTL formula $\varphi$ is given by the following:

- $\langle s, \nu \rangle \models a \Leftrightarrow a \in (l)$;
- $\langle s, \nu \rangle \models \neg\varphi \Leftrightarrow \langle s, \nu \rangle \not\models \varphi$ ;
- $\langle s, \nu \rangle \models \varphi_1 \vee \varphi_2 \Leftrightarrow \langle s, \nu \rangle \models \varphi_1$ or $\langle s, \nu \rangle \models \varphi_2$;
- $\langle s, \nu \rangle \models E\varphi_1 U_{\mathcal{I}} \varphi_2 \Leftrightarrow$ there is an infinite run $\rho$ in $TA$ starting from $\langle s, \nu \rangle$ such that $\rho \models \varphi_1 U_{\mathcal{I}} \varphi_2$;
- $\langle s, \nu \rangle \models A\varphi_1 U_{\mathcal{I}} \varphi_2 \Leftrightarrow$ any infinite run $\rho$ in $TA$ starting from $\langle s, \nu \rangle$ such that $\rho \models \varphi_1 U_{\mathcal{I}} \varphi_2$;
- $\rho \models \varphi_1 U_{\mathcal{I}} \varphi_2 \Leftrightarrow$ there exists a *position* along $\rho$ such that $\rho[\pi] \models \varphi_1$, for every position $0 < \pi' < \pi$, $\rho[\pi'] \models \varphi_2$, and $duration(\pi_{\leq \pi}) \in \mathcal{I}$.

where $\rho[\pi]$ is the state of $\rho$ at the position $\pi$ and $duration(\pi_{\leq \pi})$ is the sum of all the delay along $\rho$ up to the position $\pi$.

The exact definition of the term *position* change the semantic of TCTL. In the *continuous* semantics, a position in a run $\rho$ is any state appearing along it. In the *pointwise* semantics, a position in a run $\rho$ is a state only right after a discrete action has been done. As seen in table 2.3, differently from the other temporal logics, the use of one or the other semantics does not change the decidability of the model checking problem on TCTL.

## 2.5 Model Checking: Software Tools

Model Checking technique, being almost automatic, go a wide acceptance in checking real systems and a number of tools were developed for it. For most uses a Model Checker can be seen as black box, it takes a model of the system and a properties to check and gives in output the whatever the properties is satisfied or a trace showing the problematic behaviour of the system. Properties can be roughly classified into three categories:

- safety properties, i.e.: something deemed *bad* is never reached by the system, *"the system never does ..."*;
- reachability properties, eventually the property will be satisfied by the system, i.e.: *"the system can do ..."*;
- liveness properties, something deemed *good* will eventually be reached by the system, *"eventually the system does ..."*.

Table 2.4 shows some of the most used model checker and the kind of formalisms the use for the specification of the model of the system and the goals.

| Tool Name | Modelling Language | Properties Language |
|---|---|---|
| Kronos[1] | Timed Automata | TCTL |
| UPPAAL[2] | Extended Timed Automata | Fragment of TCTL |
| NuSMV[3] | Proprietary Language | CTL, LTL |
| BLAST[4] | Proprietary Language | Proprietary Language |
| SPIN[5] | Promela (Process Meta Language) | LTL |
| TLA+ Proof System (TLAPS)[6] | Temporal logic of actions (TLA) | TLA |

[a] http://www-verimag.imag.fr/ tripakis/openkronos.html
[b] http://www.uppaal.com/
[c] http://nusmv.fbk.eu/
[d] http://mtc.epfl.ch/software-tools/blast/index-epfl.php
[e] http://spinroot.com/spin/whatispin.html
[f] http://msr-inria.inria.fr/ doligez/tlaps/

Table 2.4: Model Checking Tools

### 2.5.1 UPPAAL Model Checker

We will now give more detail about the UPPAAL model checker being it the model checker used as the verification engine in this work.

UPPAAL is an integrated environment for the modelling, the simulation and the verification (through model checking technique) of realtime timed

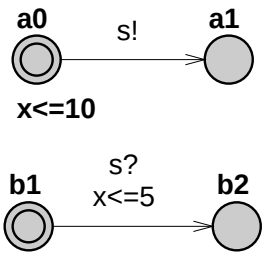



Figure 2.6: UPPAAL XTA Automata

systems. Started as a JAVA project developed at the Uppsala University in Sweden, it has since emerged from the academy and reached commercial maturity and has been used with success to verify a number of systems.

The core of the modelling language used by UPPAAL is the Extended Timed Automata framework. While being considerably lower level if compared to some model checking specification language the ability to use a GUI to draw the specification make the specification process comfortable. Moreover is possible to provide a textual description of the XTA network as a, .xta, text file.

The .xta file format is documented and its syntax is similar to the formal notation used to describe XTA automaton. This make possible to interface the model checker with other tools able to automatically generate compatible .xta files.

For example the two automata in figure 2.6 can be specified using the following .xta files.

```
// Comments
clock x; chan s;    // Clocks and Synchronisation symbols
                    // (called channels)

// First Automaton -> A
process A { state a0{x<=10}, a1;// States
                                // x<=10 invariant of state a0
```

```
init a0;              //A initial state

// Transitions
// From a0 to a1 synchronising on s
trans a0 -> a1{sync s!; }; }

// Second Automaton -> B
process B { state b1, b2;

init b1;        // B initial State

trans b1 -> b2{guard x<=5; sync s?; }; }

// The system is composed by the A and B automata
system A,B;
```

### Simulation and Verification in UPPAAL

While not being automatic and by far exhaustive simulation can be used to better understand complex systems behaviour. The UPPAAL GUI integrate a simulator that allow to dynamically *execute* the automata network, giving the possibility to the user to choose what transition are fired and to see the value of the clocks step by step as shown in Fig.2.7.

One of the most useful feature of the simulator is its integration with the model checker, in fact the error traces returned by the model checker can be visualised in it. This allow the modeller to see what steps the system took to end on a state violating the provided goals.

The verification language used by UPPAAL is a fragment of the TCTL logic, particularly it doesn't allow the nesting of the TCTL operators.

We can define as state formulae the conjunction of formulae on the locations, e.g.: A.a0, the automata A is in the location a0, clock formulae, e.g.: $x \leq 15 \wedge y \leq 5$ and integer formulae, e.g.: X := true. There is also the particular state formula `deadlock` that evaluate to true when the system is

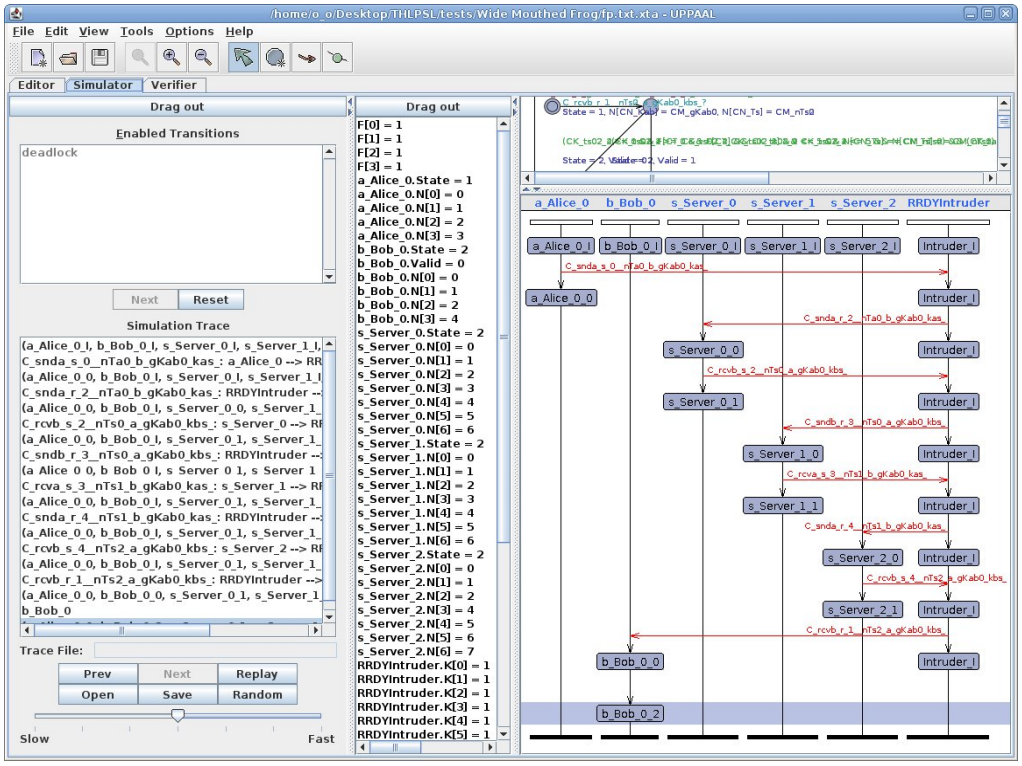



Figure 2.7: UPPAAL Simulation UI

unable to progress, no exit transitions enabled but forced to leave the current location.

With $\phi$ and $\psi$ state formulae the properties that can be checked are:

- $A\Box\phi$, invariantly $\phi$, i.e.:every reachable state verify *phi*;
- $E\Diamond\phi$, possibly $\phi$, i.e.:there is a reachable state that verify $\phi$;
- $A\Diamond\phi$, always eventually $\phi$, i.e.: for every possible path, there exists a state such that $\phi$ is satisfied;
- $E\Box\phi$, potentially always $\phi$, i.e.: there exists a infinite path to a state where $\phi$ is satisfied by all states at some point;
- $\phi \rightarrow \psi$ every path that contain a state that satisfy $\phi$ contain a state that satisfy $\psi$. Shorthand for $A\Box(\phi \Rightarrow A\Diamond\psi)$.

Those kind of properties are enough to express the reachability in a system.

## 2.6 Boolean satisfiability problem: SAT

Boolean satisfiability (SAT) is the problem of deciding if there is an assignment for the variables in a propositional formula that makes the formula true. It was the first known NP-complete problem, as proved by Stephen Cook in 1971. It is of considerable practical interest and has received a lot of attention and many different algorithms and techniques have been devised to try and solve it efficiently as many decision problems, such as graph colouring problems, planning problems, and scheduling problems can be encoded into SAT. The history of SAT solving can be roughly divided in two eras. Pre 1960, partially because the limited computational powers of time computers and the complexity of the problem, there was no implementation of SAT solving algorithms.

In 1960, Davis and Putnam published an algorithm [DP60] (denoted by DP) which started the interest in SAT solvers. While being very inefficient it motivated the subsequent development of the Davis-Logemann-Loveland (DLL) algorithm [DLL62]. The algorithm has been used extensively to solve many kinds of problems using computers (especially from artificial intelligence and operating research).

In the beginning of the 90's, computers became powerful enough to solve medium sized SAT instances using simple implementations. As a consequence, some researchers started studying how to improve SAT solvers in practice. Benchmarks and worldwide competition has been established and the quest for the fastest ever solver was born.

Formally, given a set of Boolean variables, called proposition, $\mathcal{B}$, a Boolean literal is a formula of the form $b$ or $\neg b$ with $b \in \mathcal{B}$. A clause is a finite disjunction of literals. A formula $F$ in the conjunctive normal form (CNF) is a finite a conjunction of clauses.

This can be described by the following set rules:

- $Proposition ::= b_1|b_2|\ldots|b_n$;
- $Literal ::= \neg Proposition|Proposition$;
- $Atom ::= Literal|\bot|\top$;
- $Clause ::= Atom|(Clause \vee Clause)$;
- $CNFFormula ::= Clause|Clause \wedge Clause$;
- $Formula ::= Atom|\neg Formula|(Formula \wedge Formula)|(Formula \vee Formula)$;

Solving the SAT problem for a propositional formula $\phi$ is the answer to whenever exist an assignment for the proposition in $\mathcal{B}$ that make $\phi$ true.

Some common method for solving the problem are:

- enumerating all possible truth values and checking each of them to see whether it satisfies f;
- performing a backtracking search algorithm through the possible truth as- assignments of f to show that it is satisfiable. This is what the DLL algorithm do and is by far the most common;
- checking directly if the formula is a contradiction by completely simplifying or by using the resolution method and testing if the resulting formula is empty;
- showing that the complement of f 1 is not valid using a theorem prover;
- using binary decision diagrams.

## 2.7 Difference Logic

The Difference Logic (DL) extend the propositional logic with *difference constraints*, i.e. inequalities of the form $(x - y \sim c)$ where $\sim \in \{<, =\}$, x and y are numerical variables, and c is a constant.

Formally let $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$ be a set of Boolean variables and $\mathcal{X} = \{\mathcal{X}_1, \mathcal{X}_2, \ldots\}$ a set of numerical variables. The set of atomic formulae of

$DL(\mathcal{B}, \mathcal{X})$ consists of the Boolean variables in $\mathcal{B}$ and the numerical constraints in the form $\mathcal{X}_i - \mathcal{X}_j \circ c$ with $c \in \mathbb{Q}$ and $\circ \in \{<, >, \leq, \geq\}$.

The set $\mathcal{F}$ of *DL formulae* is the smallest set of formulae containing $\mathcal{B}$, all the numerical constraints over $\mathcal{X}$ and closed under the usual Boolean connectives ($\neg$, $\wedge$ and $\vee$).

The semantics is given w.r.t. a pair of valuation functions $(v_{\mathcal{X}}, v_{\mathcal{B}})$ defined as follows.

An $(\mathcal{X}, \mathcal{B})$ valuation consists of two functions $v_{\mathcal{X}} : \mathcal{X} \to \{T, F\}$ and $v_{\mathcal{B}} : \mathcal{X} \to \mathbb{R}$ which associate a Boolean value to each Boolean variable and a real value to each numerical variable, respectively.

A $(\mathcal{X}, \mathcal{B})$-valuation can be extended to DL formulae in the obvious way. In particular, $\langle v_{\mathcal{X}}, v_{\mathcal{B}} \rangle$ satisfies the constraint $\mathcal{X}_i - \mathcal{X}_j \leq c$ if and only if $v_{\mathcal{X}}(\mathcal{X}_i) - v_{\mathcal{X}}(\mathcal{X}_j) \leq c$.

It is well known that the satisfiability problem for DL is an NP–Complete problem. Notice, however, that satisfiability of the conjunctive fragment of DL can be solved in polynomial (cubic) time using a variant of the Floyd-Warshall algorithm.

**Algorithm 1** Floyd-Warshall

**Require:** edgeCost(i,j) cost of the edge between i and j or infinity
**Require:** path[i][j] = edgeCost(i,j)
**Ensure:** path[i][j] shortest path between i and j

```
for k = 1 to n do
  for i = 1 to n do
    for j = 1 to n do
      path[i][j] = min ( path[i][j], path[i][k]+path[k][j] );
      check that ∀x path[x][x] is not negative;
    end for
  end for
end for
```

Infact the conjunctive fragment of DL has the following graph interpretation:

- variables are nodes;
- atoms $x - y \leq c$ are weighted edges;

- a set of literals is satisfiable iff there is no negative cycle (i.e.: the there is a minimum path).

# Chapter 3

# Formal Verification of Security Protocols

In this chapter we will introduce the problem of the formal verification of security protocols. We will start in section 3.1 introducing some background informations and common notations used in the protocol modelling field. After this basic information we will show, in section 3.2, how timing information are important for the specification of a protocol. In section 3.3 we will conclude presenting existing, state of the art, approaches to the problem of the formal verification of security protocols.

## 3.1 Communication Protocols

Communication protocols specify an exchange of messages between entity called principals, they are *distributed algorithms which focus on messages exchange* [Com00]. The principals are the agents participating in a protocol execution and can also be called users, hosts, or processes. The communication take place over *channels*, ranging from secure ad hoc connection (e.g.: the communication bus of a secure computing platform, smartcard buses), open cabled networks, over the air radio (wireless, WiFi) channels, .... In general every message sent over a networks cannot be considered *secure*. Even using ad hoc network, and even more in the case of open networks

such as the Internet, protocols should be designed "robust" enough to work even under worst-case assumptions, that is dishonest or careless principals or a motivated and powerful third party intruder able to eavesdrop or tamper with the messages flow. A security protocol, called also cryptographic protocol, try to secure the communications over an insecure networks and to provide security guarantees such as authentication of principals or secrecy of exchanged messages through the application of *cryptographic primitives*.

### 3.1.1 Cryptographic Primitives

To communicate and to create messages, agents use a number of tools, or cryptographic primitives. These basic primitives are the following.

#### Concatenation

The first type of cryptographic primitive is the concatenation of messages. Message exchanged between principals can be seen as a concatenation of smaller sub messages. Usually concatenation is indicated with the notation $\langle M1.M2\rangle$, with $M1$ and $M2$ sub messages. A defining properties of concatenation is whatever it is associative or not, i.e.: $\langle M1.\langle M2.M3\rangle\rangle == \langle\langle M1.M2\rangle.M3\rangle$. Usually in this thesis we will assume associative concatenation and will write $\langle M1.M2.M3\rangle$ for $\langle M1.\langle M2.M3\rangle\rangle$ or, equivalently, $\langle\langle M1.M2\rangle.M3\rangle$.

#### Nonces

Nonce (abbreviation of Number used Once) are random unique identifiers that provide the ability to differentiate between different sessions of the same protocol. As long as not disclosed to another principals, a nonce is a secret owned by that particular principal and cannot be guessed by any other principals (honest or not).

### Cryptography

Cryptography is the practise and study of hiding information and is one of the most fundamental part of a security protocol. Informally encryption is the process of converting information (the plaintext, e.g.: a message) into an intelligible form (the cipher text) and decryption is its reverse. Formally a cipher is a pair of algorithm, a encryption algorithm and a decryption algorithm (for some kind of cipher the encryption and decryption algorithms are the same but its not the norm). The cipher operation depends on the plaintext/cipher text to, respectively, encrypt/decrypt and on some *keys*. How the keys are used by the ciphers classify them into symmetric or asymmetric ciphers. Symmetric ciphers use the same key, $K$, for both the encryption and the decryption process while in asymmetric cipher there are two distinct keys one for encryption, $K$ called *public key*, and its *inverse*, $K^{-1}$ for decryption called *private key*. Its important to note that, while we use the notation $K^{-1}$, in a sound asymmetric cipher, there is no way to compute it from $K$.

Applying the encryption algorithm to a plaintext, $M$, with a key, $K$, result in cipher text, $\{M\}_K$ that is:

1 *strongly* dependent on the value of $M$ and $K$;

2 cannot be inverted without the knowledge of the inverse key $K^{-1}$ (for simplicity sake in the case of a symmetric cipher we will assume that $K = K^{-1}$).

Point 2, called *perfect cryptography*, is a really strong assumption and while no practical cipher enjoy this property, all ciphers try to at least make, computationally, difficult to recover a plaintext without knowing the corresponding keys. On a sound cipher (i.e.: the best way to recover a decryption key is by brute–forcing) this is achieved by the use of a large enough key space.

As a matter of fact cryptographic algorithms ensure a high degree of confidence in exchanging messages over insecure communication channels. The best-known symmetric cryptographic algorithms are the DES (Digital Encryption Standard) [FIP76] and the AES (Advanced Encryption Standard)

[Sta02]. Meanwhile the best-known asymmetric algorithm is RSA (Rivest, Shamir, and Adleman) [RSA78].

### Hash Function

A cryptographic hash function is a deterministic function that takes an arbitrary block of data (e.g.: a message) and returns a fixed-size string, the (cryptographic) hash value. Any accidental or intentional change to the data will change the hash value. The ideal cryptographic hash function has four main or significant properties:

- it is easy to compute the hash value for any given message;
- it is infeasible to find a message that has a given hash;
- it is infeasible to modify a message without changing its hash;
- it is infeasible to find two different messages with the same hash.

Cryptographic hash functions have many information security applications, notably in digital signatures. Know cryptographic hash function are MD5 (Message Digest 5) [132] and SHA1 (Secure Hash Algorithm 1) [EJ01].

### Digital Signatures

A digital signature is a mathematical scheme for demonstrating the authenticity of a message. It gives a recipient reason to believe that the message was created by a known sender, and that it was not altered in transit.

Usually digital signature employ asymmetric encryption algorithm and cryptographic hash function. A common way to create a digital signature of a message $M$ consist in the encryption of its hash, $H(M)$, using a private key, $K$. Any principals willing to check the signature, and so the authenticity of a message, could decrypt the hash using the corresponding public key and then check that the hash correspond to the sent message, since it is infeasible to modify a message without changing its hash and the key used to encrypt the message is private.

### 3.1.2 A-B Notation

A common notation used in the specification of security protocols is the so called *A–B Notation*, *Alice-Bob Notation* [CJ]. The notation has the following formal syntax:

```
SpecAB ::= Message_Flow_List
Message_Flow_List ::= Message_Flow |
                      Message_Flow Message_Flow_List
Message_Flow ::= Party_Name "->" Principal_Name ":" Message_Spec
Message_Spec ::= Nonce | Message | "{"Message_Spec"}"Key_Name |
                 Message_Spec"."Message_Spec |
                 "f(" Message_Spec ")"
```

With `Message`, `Nonce`, `Key_Name` e `Principal_Name` elements of the sets of, respectively, messages, nonces, keys, principal names.

This notation allow the specification of the expected execution of a protocol, i.e.: the exchanged messages. One of the main problem with this notation and is that its limited only to the message flow, important details, like the sharing of keys, are omitted or implicit.

#### An Example: the Needham-Schroeder Public Key Protocol

As an example of protocol specification using the A–B Notation consider the Needham-Schroeder Public Key Authentication Protocol (NSPK):

1. $A -> B : \{NA.A\}_{KB}$
2. $B -> A : \{NA.NB\}_{KA}$
3. $A -> B : \{NB\}_{KB}$

where $A$ and $B$ are the principals involved in the protocol; $KA$ and $KB$ are the public keys of $A$ and $B$, respectively; $NA$ and $NB$ are nonces generated by $A$ and $B$, respectively. Step (1) of the protocol models $A$ sending $B$ a message with the identity of $A$ and the nonce $NA$ encrypted

with $KB$, here the specification assume implicitly that $B$ known $KB^{-1}$ and so could receive the sent message and learn the value of $NA$. In Step (2) $B$ send to $A$ his nonce $NB$ and the received the nonce $NA$ proving its participation to the protocol. The message is encrypted with $KA$ so, like in the first message, only $A$ is able to decrypt the message and learn the value $NB$. In Step (3) $A$ concludes by proving to $B$ her own participation in the protocol. The protocol end with both party *assured* of the identity of each other (i.e.: the protocol perform a mutual authentication).

### 3.1.3 Honest Agents and Intruder Models

As we said a cryptographic protocol is basically a communication protocol defined over a certain number of participants, called agents, trying to assure a *secure* communication. However there can be of two types of agents. Honest agents, or principals, "official" participants to the protocol, whose behaviour is precise defined by the specification of protocol they execute, and rogue agents, or intruders which don't follows the protocol specification and try to gain an unfair advantage over the hones agents (be it gaining knowledge of secret messages, unauthorised authentication, ...).

While the behaviour of the principal is defined by the protocols itself the behaviour of the intruder is defined by its intruder model. The most famous intruder model is the one devised by Dolev and Yao in [DY83]. The idea behind this intruder model (Dolev-Yao Intruder Model) is to give the intruder the most resources possible without violating the assumption of perfect encryption, that is without permitting the intruder to decrypt a message without knowing the necessary key. The Dolev-Yao intruder can intercept, read and delay any message sent by an honest agent. In addition, it can decompose any message acquired and use the knowledge obtained to construct new messages. Is can also send messages, under any false identity. The Dolev-Yao Intruder Model infact represent the worst case scenario for a security protocol execution to the point it is normally associated to the idea that the intruder is itself the communication channel.

### 3.1.4 Channels

Messages exchange between protocols principals take place on communication channels. While generally abstracted from the protocol specification (e.g.: the A–B Notation does not provide any facility to specify channels characteristics) can be of fundamental importance for a sound specification. The characteristic of a channel determine what the agent (honest or not) can perform on the channel. There are various kind of channels:

- unsafe channels, the most common channel type, where communication between principals is not assured i.e.: an intruder can perform all the operation typical of the Dolev-Yao Intruder Model;
- safe channels, where communication between principal is safe from any intruder;
- resilient channels, unsafe channels where, however, intruder cannot block messages.

Specifying the kind of channels gives the ability to accurately model thing like safe, *out of the band*, key exchange, wireless channels and so on.

### 3.1.5 Goals

Cryptographic protocol try to assure a *secure* communication. There are many properties that may be required by a security protocol. The following are the most common.

#### Secrecy

Intuitively, a security protocol assure secrecy if there is no way for the intruder to know some, secret, message. This can be simply accomplished via encryption with an undisclosed (i.e.: not obtainable by the intruder) key. A stronger version of secrecy, called non-interference, is the inability of the intruder to indirectly know a secret. Consider for example a simple communication protocol where all the possibles message are $M1$ and $M2$ (e.g.:

buy item 1/2, vote for 1/2). The agent $A$ sent his choice ($M1$ or $M2$) to $B$ encrypted with $KB$, $B$ public key. Since only $B$ has $KB^{-1}$, the protocol preserve the (weak) secrecy of the message, hoverer, since the intruder can infer what message was sent by simply comparing the intercepted message with all its possibles value (remember that the key $KB$ is public) the protocol does not preserve the strong secrecy.

#### Authentication

Intuitively, authentication is the ability for a principal to be sure of the identity of his correspondent. For example, a bank receiving an order for money transfer checking for the identity of his client, a mail server checking the identity of the mail box owner. This is usually achieved by sending a nonce (that in many real network protocol is the user name/password pair) encrypted by the public key of the client. A problem common to many authentication methods is their vulnerability to man in the middle attacks where an intruder use a principals as an oracle to respond to an authentication request.

#### Non Repudiation/Anonymity

Non-repudiation ensure that a principal, in the event of dispute cannot, repudiate, or refute the fact of being the originator of a message. This is usually achieved through the extensive use of digital signatures. Opposite to the non repudiation is the anonymity where the security protocol assure that is impossible to charge a principal as the originator of a message.

## 3.2 Timed Security Protocols

Security protocols where the explicit specification of temporal aspects is needed to correctly to preserve their security properties are called Timed Security Protocols. Aspects like:

- **Delays and Timeouts**: the ability to enter a recovery phase or terminate the execution of a protocol or to delay an event,

- **Timed Messages**: the ability to assign a temporal constraints on the availability and usability of a messages (message disclosure and expiration time). The use of timestamps (i.e.:the time at which an event occurred),
- **Channels Timings**: the time that communication over a channel takes to be performed and its subtle interaction with the intruder model,

can affect the security of a protocol.

### An Example: the Wide Mouthed Frog authentication protocol

For example consider the well known Wide Mouthed Frog authentication protocol [BAN89]. 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 (i.e.:a form of authentication). 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.

Abstracting the time from the protocol, i.e.: removing the timestamps and the communication channel timings, effectively changes the nature of the protocol. This protocol will be one of the running examples in the following chapters.

#### Timed Goals

Quantitative time information can be significative also for the security goals that the protocol is trying to achieve. For example the secrecy of a message could be needed in a precise time interval (e.g.: the secrecy of a temporary key, after the end of the protocol its secrecy is no more needed) or an authentication must be completed in a determined time frame.

## 3.3 State of the Art

Limiting our interest to the verification of the secrecy in a security protocol there is a number of works that have researched the problem.

Unfortunately, with no restrictions on the protocol this problem is undecidable [EG82]. Limiting the protocols, by limiting the number of concurrent sessions and/or the structure of the messages, a number of interesting decidability result and tools appeared in bibliography.

Notably Gavin Lowe in [Low98] applied the FDR model checker to analyse security protocols specified in CSP (Communicating Sequential Processes [Hoa85], i.e.: an abstract language for modelling concurrent systems). Intuitively, the various protocol steps are modelled as processes that exchange messages through channels. Also the intruder and the network are modelled as CSP processes and channels are used to model both the intruder abilities and important events in the protocol. In [CJM00] D. Clarke et al. developed Brutus, a model checker that performed a depth-first search of the state graph and implementing a message derivation mechanism modelling the intruder's

capabilities. This two approaches, and others based on explicit state model checker, suffer from the state explosion problem. Limiting a priori the size of messages, and the number of nonces exchanged as well as using partial order techniques the authors have partially address that problem. Differently from explicit state model checker the tools Athena [Son99] and CASRUL [JRV00] both use symbolic state exploration, one in the backward from the security goal to the initial state (Athena), the other forward (CASRUL). The problem with this approaches was that as a semi-decision methods they always end on unsafe protocols but can loop indefinitely without proving the security of the protocol. Also the use of symbolic methods make the production of counterexamples more complex and time consuming.

### The AVISPA Framework

The AVISPA (Automated Validation of Internet Security-sensitive Protocols and Applications [ABB+05]) Framework is a push-button tool for the automated validation of security protocols. This framework provide a high level specification language and number of different verification engines.

#### *HLPSL*

The idea behind the High Level Protocol Specification Language (*HLPSL*) was to provide protocol engineers with a convenient, human readable, and easy to use language easily translatable into a lower-level formalism well-suited for implementation into model-checking tools.

The *HLPSL* language derive part of its syntax from the TLA logic, modelling a protocol by describing its actual state and how this state change.

The language allow the use of typed variables, the structuring of the specification using a kind of procedural abstraction, provide the ability to express common cryptographic primitives (concatenation, encryption, hashing) and allow to specify the goals that a protocol specification must satisfy.

In a *HLPSL* specification the global state of a protocol is defined by an assignment of values to all the system variables. First order logic formulae on state variables are called *state predicate*. The evolution from a state to

another is described by logic formulae called *transition predicates* that bind the value of the variables in the current state to the value in the next state. The variables of the next state are called *primed*; given a variable $X$, with $X$ we denote its value in the current state and with $X'$ (X primed) its value after the transition.

An $HLPSL$ specification is composed by a series of *roles*, a modular template for the behaviour of the principals of the protocols.

A $HLPSL$ specification is described by the following grammar in (*var_ident* are the variables of the language):

```
SpecHPSL ::= role_definition_list
             [goal_Declaration]
             main_role_call
role_definition_list ::= role_definition | role_definition_list
role_definition ::= basic_role | composition_role
main_role_call ::= var_ident "()"
goal_declaration ::= ...
```

A specification in $HLPSL$ can be partitioned into *transition roles* (called also basic role) and *composition roles*, followed by a *call* to a particular composition role called *main role* and, eventually, by the goal specification.

```
Transition role definition 1
Transition role definition 2
⋮
Transition role definition n

Composition role definition 1
Composition role definition 2
⋮
Composition role definition n

Main composition role call

Goals declaration
```

### The Intruder

A *HLPSL* specification allows, like the A-B notation, to models the *expected* execution of a protocol but, differently from the A-B notation, implies the presence of an *intruder*.

In *HLPSL* to every communication channel (intuitively the media that the principals use to exchange the messages) can be associated a different *intruder model* the determine the power that the intruder have on that particular channel. At moment the, only, allowed intruder model is the Dolev-Yao intruder model.

The intruder in *HLPSL* as well as being implicitly present on the communication channels can be explicitly referenced during the instantiation of a role, using the constant *i*, in this way is possible to model sessions where there is an explicit communication between the principals and the intruder. In this way the intruder can possibly enrich his knowledge of the protocol and use this knowledge to build attacks on the protocol.

### Types and Variables: Base Types

In the roles is possible to define the variables that will be used in the protocol specification, also the roles can be parametrised.

*HLPSL* is a typed language, there are many different types and for some types additional attributes can be used to specify additional features. The base types of the language are:

- `agent` variables identify the principals of the protocols. *HLPSL* define the particular `agent` variables, *i*, to explicitly model the intruder;

- `public_key` and `symmetric_key` variables models the keys used for, respectively, asymmetric (public key) cryptography and symmetric cryptography. Given a public key `k`, the corresponding private key is denoted by `inv(k)`. Given a key `k` and a text variable `m` `{m}_k` is the encryption of the variable `m` with the key `k`. Given `{m}_k` the decryption is possible if the principal knows `k`, if symmetric, or `inv(k)`, if asymmetric;

- `text` or `msg` variables represent the messages that the principals exchange during the protocol run. Variables of type `text (fresh)` (where textttfresh is an attribute of the type) subjected to priming are used to represent the nonces of the protocol;

- `nat` variables represent the natural numbers;

- `bool` variables represent the Boolean values;

- `function` variables represent non invertible function on the space of messages. This kind of variable is used to model *one way hash functions*. Given a variable `a` of type `function` with `a(X)` we denote a value of type `msg`, i.e.: the application of the function `a` to `X`. A principal (or the intruder) knowing only `a(X)` cannot obtain `X`;

- `channel` variables models the communication channels.

The syntax used to declare a variable is the following:

```
var_decl_list ::= exists var_decl{"," var_decl }
var_decl ::= var_ident {"," var_ident} ":" base_type_name["("attr_name")"]
base_type_name ::= "agent" | "public_key" | "symmetric_key" |
                   "text" | "nat" | "bool" | "function" |
                   "channel" | "msg"
attr_name ::= "fresh" | "dy"
```

To declare a variable, `x` of type $t$ we use the expression `exists x : t`; its possible to declare multiple variables on same line by sepatating them by a comma eg.: `exists x1, x2 : t`. A similar syntax is used to declare the list of the parameters of a role.

### Structured Types

The basic types can be structured in three different ways :

- tuple;

- lists and sets;

- functions and mappings on messages.

The general grammar is the following:

```
var_decl_list ::= "exists" var_decl{"," var_decl } |
                  "local" var_decl{"," var_decl }
var_decl ::= var_ident {"," var_ident} ":" type_name["("attr_name")"]
type_name ::= subtype_of | subtype_of "->" subtype_of
subtype_of ::= base_type_name |
               subtype_of "list"|
               subtype_of "set"|
               "(" subtype_of {"," subtype_of} ")"
base_type_name ::=  "agent" | "public_key" | "symmetric_key" |
                    "text" | "nat" | "bool" | "function" |
                    "channel"
```

For example, `(1, 2, 3, 4)` is a tupla whose type is (nat,nat,nat,nat), `(a, 1, 2, b)` have type (bool,nat,nat,public_key) if `a` is a `bool` and `b` is a `public_key`. A list of naturals is defined with `exists IntList: nat list`, similarly with the keyword `set` we can define a set of naturals. The list can be initialized like `IntList = []`, element are added to the list using the operator `cons`, e.g.: `IntList'=cons(10,IntList)`.

*HLPSL* allow the definition of two different types of functions: *functions on messages* and *mappings*. The main difference between the two types of functions is that functions on the messages are *constant*, i.e.: enjoy the property that once defined do not change their value during the execution of the protocol, which is the case of the mappings. For example, a function on messages that associates names with numbers: `F: text -> nat` initialise with `F = [(A, 1), (B, 2)]` (1 is associated to A and 2 to B) will not change once initialised and for all the duration of the protocol `F (A) = 1`.

The mappings are defined and initialised in a manner identical to the functions on messages with exception that at any time you can add elements using the mapping operation, for example `F '(X) = 3` add (or change) the mapping between X and 3.

### *HLPSL* Roles

A role in *HLPSL* is a description of the behaviour of a principal. A role can be parametrised and have local variables. There are two types of role, the transition roles that describe the actions of a principal and the composition role consisting of the instantiation of one or more of role. The composition roles allow the structuring of the specification.

A role definition is composed by the declaration of its name, any formal parameters and, in the case of a transition role, a *player* declaration that binds an agent passed as parameter to the role; the agent tells you who is executing the role. The formal parameters are declared using the same syntax used to declare local variables in a role.

Following there are the, optionals, *role headers*:

- the declaration of the local variables (using the `exists/local` operator);

- the declaration of the starting state (using the `init` operator) specifying the initial values of the local variables of the role. For example given a variable of nat type `X`, `init X = 0` sets its value to 0. Set or list variables can be initialised using the operator `/\_` that allow iteration on all the element of the set es: `init /\_ {in(IT,X)} IT = 0` set all the element of `X` to 0;

- the declaration of the accepting states for the role using the `accept` operator. The operator define fir a transition role a predicate that identify the accepting state of the role;

- the declaration of the initial knowledge of the role using the operator `knowledge`. The initial knowledge allow to specify messages know by a principal before the start of the protocol, e.g.: public, private keys, share keys. It can also be used to specify the intruder initial knowledge, i.e.: already. With `A agent` variables and `k public_key` variables, `knowledge(A) = { k, inv(k) }` state that the agent `A` knows the key `k` and its inverse `inv(k)`;

- the declaration of owned variables using the *own* operator; while in $HLPSL$ is possible to share variables between roles, an *own declaration* in a role state that a role will be the only one to modify that variable;

- the transition section (is we are in a transition role) or the composition section (is we are in a composition role).

The following syntax is used for the roles specification:

```
role_definition ::= basic_role | composition_role basic_role ::=
role_declaration player_def role_header
               [transition_declaration]
               "end role"
composition_role ::= role_declaration role_header
               [composition_declaration]
               "end role"
role_declaration ::= "role" var_ident "(" parameter_list ")"
played_def ::= "played_by" var_ident role_header ::= "def ="
                [exists_declaration]
                [owns_declaration]
                [init_declaration]
                [accept_declaration]
                [knowledge_declaration]
exists_declaration ::= "exists" var_decl{"," var_decl }
owns_declaration ::= "owns" var_decl{"," var_decl }
init_declaration ::= "init" init_declaration_list
init_declaration_list ::= var_ident "=" expression [init_declaration_list] |
                          "/\_{" parameter_list "}" var_ident "=" expression
accept_declaration ::= "accept" predicate
knowledge_declaration := "knowledge("var_ident")" "={ expression "}"
parameter_list ::= var_decl
```

### Transition section

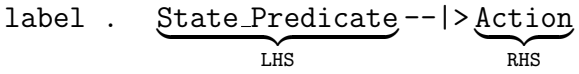
The state of a role is the value of all its non primed variables (*state variables*). A *state predicate* is a logical formula over state variables and con-

stants (e.g.: `S = 2`). A *transition predicate* is a logical formula over state variables, constants, and variables subjected to priming (e.g.: `S = 2 /\ X'` = 1). The transition predicates model the evolution of the protocol while the state predicates constrain it.

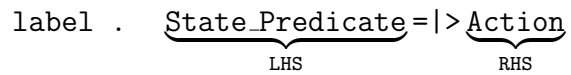
A transition section has the following syntax:

```
transition_declaration ::= "transition" transition_list
transition_list ::= transition [transition_list]
transition ::= label "." state_predicate "--|>" transition_predicate_list |
               label "." state_predicate "=|>" transition_predicate_list
```

A transition role is composed by a transition section that contains one or more transitions with the following form:




or




There are two different kind of state transition, the spontaneous transitions (denoted by `--|>`) and instantaneous transitions (`=|>`). Transitions connect a predicate on the left side (LHS) with an action on right side (RHS); spontaneous transitions, i.e.: `A -|> B`, indicates that if the predicate A is satisfied by the current state then it is possible to move to the state described by B, the principals, however, is not forced to change state and can delay the transition. Conversely, the immediate transition has the property that the transition is executed immediately when the predicate on the LHS is satisfiable.

The LHS of a transition has the following syntax:

```
state_predicate ::= formula ["/\" state_predicate] |
                    var_ident"(" expression_list ")" ["/\" state_predicate] |
                    var_ident"(start)"
formula ::= expression "=" expression |
```

```
            expression "<" expression |
            "in(" expression "," expression ")" |
            "not(" expression "=" expression ")" |
            "not in(" expression "," expression ")" |
            expression "/=" expression |
            "true" |
            "false"
expression_list ::= expression [expression_list]
expression ::= var_ident"'" |
               "inv("expression")" |
               expression "." expression |
               "{" expression "}" expression |
               "(" expression ")" |
               "var_ident(" expression_list ")" |
               "const_ident(" expression_list ")" |
               "cons(" expression "," expression ")" |
               var_ident |
               const_ident |
               "[]" |
               "[" expression_list "]"
```

The LHS of a transition is a predicate on the role state where is possible to:

- test for equality and inequality two variables, `A = B`, `not( X = Y)`;
- use the inequality operator on nat variables, `X <4`;
- test lists ans sets for the presence of elements, `in(X, 1)`;
- receive a message on a channel (details are given in a following section).

The above operations can be composed using the conjunction operator, `/\`.

The LHS denote for with states the transition will be executed and the action described in the RHS performed.

The RHS of a transition has the following syntax:

```
transition_predicate_list ::= transition_predicate "/\"
                              transition_predicate_list
transition_predicate ::= var_ident "'=" expression |
                         var_ident"'(" argument ")" "=" expression |
                         "var_ident(" expression_list ")" |
                         "const_ident(" expression_list ")"
```

Its possible to:

- modify a variable e.g: `A' = 2`;

- send a message on a channel (details are given in a following section).

The above operations can be composed using the conjunction operator, `/\`. Also transitions can be labelled.

For example the following transition:

```
t1. S = 1 /\ RCV(M) =|> S' = 2 /\ SND(X)
```

Declare a transition labelled `t1` that specify that, when the variable `S` is equal to 1 and on the channel `RCV` is available a message unifiable with `M`, assign to `S` the value 2 and send on the channel `SND` the message X.

### Composition section

Roles of type composition instantiate other transition or composition roles.

Composition between roles can be sequential, using the `;` operator, or parallel with interleaving semantic, using the `/\` operator. The role hierarchy is closed by a *top level role*, not parametrised, that is called at the end of the specification.

A composition section have the following syntax:

```
composition_declaration ::= "composition" composition_list
composition_list ::= composition |
                     "/\_{" parameter_instance "}" composition
composition ::= role_instance [/\ composition]
role_instance ::= "(" role_instance ")" |
```

```
                role_instantiation ";" role_instance |
                role_instantiation
role_instantiation ::= var_ident"(" arguments ")"
arguments ::= var_ident [, arguments ]
```

An example of composition could be the following:

```
role C( ... ) def=
    composition
        A1( ... ) /\ B1( ... )
        A2( ... ) ; B2( ... )
end role
```

Where role `A1` and `B1` are executed sequentially while `A2` and `B2` in parallel.

### Channels Operations

Communications between the principals of a protocol in $HLPSL$ occur along *channels*. These communications are asynchronous and the process of sending on a channel is instantaneous and independent from any receipt. The channels acts as a unlimited buffer.

Channels operations syntax is the following:

```
expression_list ::= expression [expression_list]
expression ::= ...
               "var_ident(" expression_list ")"
               ...
transition_predicate_list ::= transition_predicate
                              ["/\" transition_predicate_list]
transition_predicate ::= ...
                         "var_ident(" expression_list ")" |
                         ...
```

An operation on a channel is interpreted differently if located on the LHS or RHS of a transition. On the LHS of a transition its interpreted as a receive

operation on the channel (e.g.: `RCV(Msg)` with `RCV` a channel variable). On the RHS of a transition its interpreted as a send operation on the channel.

A receive operations on a channel state the message structure that will be accepted. A receive operation will be successful, allowing its transition to fire, if the message received conform to the structure (i.e.: is *unifiable*)specified in the receive operation.

Side effect of a receive operation is the binding of role variables with parts of the received messages.

For example:

```
role A( ... ) ...
    exists X : text,
           Y : public_key,
           State : nat;
    init State = 0;

    transition
    State=0 / RCV(X'.Y) =|> ...
end role
```

The receive operation `RCV(X'.Y)` denote a *template* of the messages that can be received, in particular `Y` denote that all the messages must end with the public_key `Y`; as a side effect to the, primed, variable X, will be binded the first part of the received message.

In $HLPSL$ is possible to specify properties of the channel used by adding a parameter to its type definition. Currently the only channel type implemented is of type Dolev-Yao (`channel(dy)`).

### Goals

$HLPSL$ also provides the ability to specify the goals that must be satisfied by the protocol. Currently, you can specify authentication and secrecy requirements. The authentication requirements between two role constrain the roles to agree on the value of a specific variable. The secrecy requirements

constrain the value of a variable to be known only to role to which it belongs and, in particular, to be unknown to the intruder. *HLPSL* also provides the ability to specify constraints on the protocol in LTL.

### *HLPSL* Example: NSPK

A possible encoding in *HLPSL* of the Needham-Schroeder Public Key Protocol presented in section 3.1.2 is the following:

```
role A( a, b : agent, ka, kb : public_key, SND, RVC : chanel(dy) )
                                                    played_by a def =
    exist state : nat
          na : text(fresh),
          nb : text

    knowledge(a) = { inv(ka) }

    init state = 0

    transition

    1. state  = 0 /\ RCV(start) =|> state' = 1 /\ SND({na'.a}_kb)
    2. state  = 1 /\ RCV({nb'}_inv(ka)) =|> state' = 2
end role

role B( a, b : agent, ka, kb : public_key, SND, RVC : chanel(dy) )
                                                    played_by b def =
    exist state : nat
          na : text,
          nb : text(fresh)

    knowledge(b) = { inv(kb) }

    init state = 0
```

```
26
27      transition
28
29      1. state  = 0 /\ RCV(na'.a)_inv(kb) =|> state' = 1 /\ SND({na'.nb'}_ka)
30  end role
31
32  role Session( ) def=
33      exist a, b : agent,
34            ka, kb : public_key
35            RCV, SND : chanel
36      composition
37      A( a, b, ka, kb, SND, RCV ) /\ B( a, b, ka, kb, RCV, SND )
38  end role
39
40  Session( )
```

The specification begin with the definition of the role `A`, its important to note among its formal parameters the declaration of the two communication channels `SND` and `RCV` of type `chanel(dy)`. Following there is the declaration of the role local variables, lines 3-5, among them a nonce (typed `text(fresh)`) `na`. At line 7 there is the declaration of the initial knowledge of the role; this principals know the inverse, `inv(ka)`, of the key `ka` passed as a parameter. The variable `state` is initialised at line 9. At lines 13-14 there is the transition section declaration describing the behaviour of this role; the sent and received messages are the same described in section 3.1.2 using the A-B Notation, just note the use of `RCV(start)` to define the start of the protocol. Similarly, from line 17 to 30, there is the definition of the role `B` describing the other principal of the protocol. The declared roles are parametric and must be instantiated to define the desired protocol execution. At line 32-38 a composition role, `Session`, create an istance of the role `A` and `B` using the opportune parameters; it's important to note that the `channel` variables passed as role parameters, `SND` e `RCV`, are passed in inverted order to the role `B` if compared to the role `A`; this way what is sent by `A` along

channel `SND` is received on the channel `RCV` and, on the contrary what is sent by `B` along the channel `RCV` is received on the channel `SND`.

At line 40 the role hierarchy is closed by the call to the composition role `Session`.

### AVISPA Verification Engines

In the AVISPA Framework the *HLPSL* language is paired with a number of verification engines providing different ways to verify the same protocol.

- The On-the-Fly Model-Checker (OFMC) [BMV05] performs protocol falsication and bounded verication by exploring the protocol state space in a demand-driven way. OFMC implements a number of correct and complete symbolic techniques. It supports the specification of algebraic properties of cryptographic operators (e.g.: exclusive or between messages), and typed and untyped protocol models;

- The Constraint-Logic-based Attack Searcher (CL-AtSe) [CV02] applies constraint solving with some powerful simplification heuristics and redundancy elimination techniques;

- The SAT-based Model-Checker (SATMC) [AC04] builds a propositional formula encoding a bounded unrolling of the transition relation specified by the protocol. The propositional formula is then fed to a state-of-the-art SAT solver and any model found is translated back into an attack;

- The TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols) [BHK08] approximates the intruder knowledge by using regular tree languages and rewriting. For secrecy properties, TA4SP can show whether a protocol is flawed (by under-approximation) or whether it is safe for any number of sessions (by over-approximation).

### Timed Security Protocols Verification

The idea of using formal methods to prove time dependent security properties is not new (e.g., see [BFST02, GLM03, NPP04, CSHM07, DG04]). A number of those papers relies on low level formalism as specification language. For example in [BFST02] the authors develop a theory of "non-interference" to prove security property in concurrent systems modeled as timed automata. The papers does not consider explicitly security protocols or issues like modeling encryption or other common primitives of security protocols.

Extending the low-level formalism of the Timed Automata, [NPP04] provide the ability to express parallelism and synchronisation on structured messages built over cryptographic primitives, allowing for a more convenient way to model security protocols that Timed Automata. Similarly, [GLM03] employ a timed process algebra as the specification language for security protocols. Both approaches require the designer to explicitly model the timing aspects of protocols, as neither Timed Automata nor timed process algebra provide, high level, timed protocol specific construct as the ones presented above. In [CSHM07] the authors use the timed automata (in the form of XTAs) as modelling language but, contrary to the ones above, they give an explicitly representation of the intruder as a timed automaton and a fine grained representation of cryptography and nonce generations; while using a higher level formalism, the fine grained approach to the protocol modelling (requiring explicit specification of nonce generation and cryptography operations) show how problematic is protocol modelling using only formal/low level languages. In [DG04] the authors use constraint programming combined with symbolic exploration to analyse infinite protocols with explicit use of timestamps (in particular the Wide Mouthed Frog Protocol). In [BL02] the authors using finite-state model checking and abstraction technique verify the TESLA protocol in an untimed setting.

# Chapter 4

# Timed Security Protocols

In this chapter 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. In section 4.1 we will present three timed protocols showing the temporal aspects that are needed to correctly model them. In section 4.1.4 we will present an extension of the, state of art, modelling language *HLPSL* for these kinds of protocols and present in section 4.1.4 we will account for its formal semantics.

## 4.1 Timed Protocols Examples

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). Examples of time sensitive protocols are, for instance, the non-repudiation Zhou-Gollmann protocol [ZG97], the TESLA authentication protocol [PCTS02] and the well known Wide Mouthed Frog protocol [BAN89].

### 4.1.1 Wide Mouthed Frog Protocol

Already presented in section 3.2 for the sake of clarity we will repeat here its detail, the Wide Mouther Frog Protocol is a well known authentication protocol.

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 (i.e.:a form of authentication). A session key should be considered valid if the associated timestamp is recent enough.

#### Timed Features in the Wide Mother Frog Protocol

To correctly model the protocol we need the ability to:

- model the use of the timestamp exchanged by the principals;
- model timing in both principals action and channels.

Abstracting the time from the protocol, i.e.: removing the timestamps and the communication channel timings, effectively changes the nature of the protocol. Abstracting time, what remain of the protocol is effectively safe. However its possible to show that in a, realistic, timed setting the protocol is unsafe. Infact the protocol is known to be vulnerable to reply attacks, where an intruder simply repeatedly intercepts the message sent by the Server

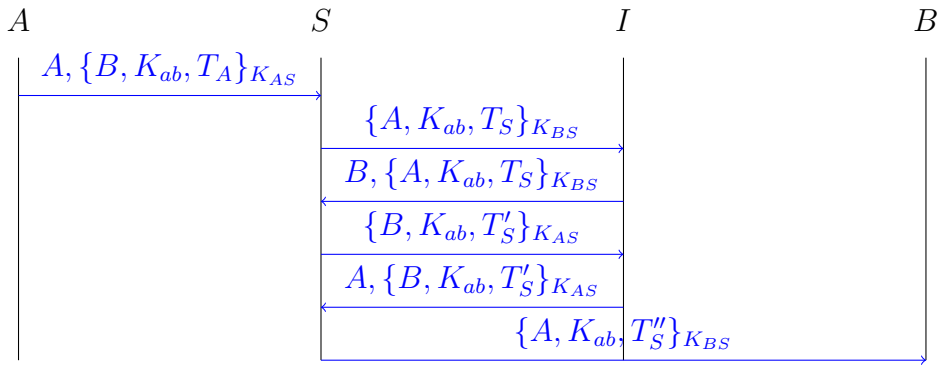



Figure 4.1: A possible attack trace on the Wide Mouthed Frog Protocol

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. Figure 4.1 show a possible attack trace on the WMF protocol.

### 4.1.2 TESLA Authentication Protocol

The TESLA protocol [PCTS02] is an authentication protocol developed for multicast authentication over an unreliable channel. There are many variant of this protocol [LN03] and its low communication and computation overhead and tolerance to packet loss allow its application ranges from authenticated audio/video steaming to sensor networks to vehicular networks.

What distinguishes it from other types of cryptographic protocols is its peculiar use of timing to provide authentication. Usual authentication protocols relies on the heavy use of public key cryptography which is difficult in scenarios where devices with low computational and networking power are used, or where the efficiency of networking throughput is relevant.

In the TESLA protocol, however, authentication is provided by using only symmetric keys, hash-functions and loose time synchronisation. Indeed, the receiver is assumed to know an upper bound on the difference between the

sender and the receiver local time, namely the maximum time synchronisation error.

A simple algorithm providing loose time synchronisation is the one described in ([PCTS02] pag. 4-6):

- `Setup`: The sender S has a asymmetric encryption key pair, with the private key $K_S^{-1}$ and the public key $K_S$. The receiver have saved its current local time $t_r$. We assume public keys to be pre distributed between the principals.

- `Protocol steps` :

  1. $R \rightarrow S : N$
  2. $S \rightarrow R : \{t_s, N\}_{K_S^{-1}}$

Where $N$ is a random and unpredictable nonce. And $t_s$ is the sender, $S$, local time. The receiver verifies the digital signature and checks that the nonce in the packet equals the nonce it randomly generated. If the message is authentic the receiver computes $t - t_r + t_s$ that is an upper bound on the current sender's time.

Moreover, packet loss resilience can also be achieved by functionally relating the keys used by the protocol in a one-way hash chain. Intuitively, from a single starting secret key a chain of keys is generated by successive applications of a hash function. From any element of the chain and the hash function, any of the following keys can then be generated. This mechanism is a common cryptographic primitive [Lam81], and is used, for example, in the well know S/KEY one-time password system [Hal94].

In the following, we present two versions of the TESLA protocol. The first one is the basic scheme, which ensures packet authentication under the hypothesis of channel reliability (i.e., no packets get lost), while the second version is more robust and guarantees packet authentication also under the hypothesis of packet loss.

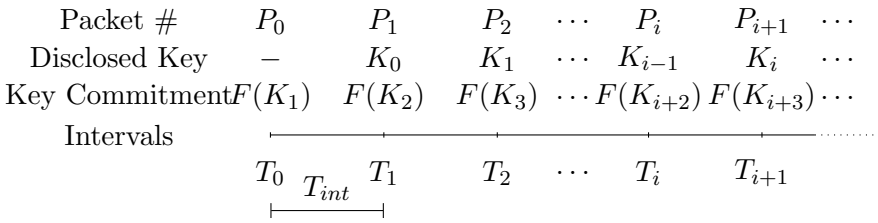



Figure 4.2: The time intervals of the TESLA protocol

### Basic TESLA Scheme

The basic scheme of the protocol involves two kinds of agent: a sender, which broadcasts a stream of packets, and many receivers, which need to authenticate the delivered packets.

One fundamental assumption in the design of the protocol is that, while the sender has enough computational power to execute complex operations, the receivers may not have, and that the sender and the receivers are loosely time synchronised.

The idea behind the protocol is quite simple. The sender splits the data stream into uniform intervals, by choosing a time interval size $T_{int}$ and the number of intervals, so that a single packet is sent within each time interval. We assume that the first time interval starts at time $T_0$, the second interval at time $T_1 = T_0 + T_{int}$, and so on. The packet sent in each time interval is signed using a key which is kept secret by the sender during the current time interval, and which is disclosed in a packet delivered in the following interval, as shown in Figure 4.2. The receivers can then use the information contained in later packets to authenticate earlier packets. To ensure that the packets cannot be forged, the disclosure of the keys is scheduled in such a way that, by the time a key is received, all the packets signed with that key must have been already buffered by the receivers. Since the sender and the receivers are loosely time synchronised, the receivers can estimate the scheduling of the packets and act accordingly.

The scheduling of the time intervals, their length, the packet size, and the key disclosure delays must be somehow known both to the sender and the receivers. This is achieved by either including those data into the first

*bootstrapping* packet sent by sender, or by assuming some predefined values. For the *bootstrapping* packet some common authentication method is used, such as public key cryptography [1].

Each packet includes, besides the payload, a Message Authentication Code (MAC), a disclosed key and a key commitment. The MAC included in each packet is obtained by applying a known MAC function with a symmetric key to the concatenation of the payload, the disclosed key and the key commitment. The key disclosed in the packet is the one used to compute the MAC signature of the packet sent in the previous interval, while the key commitment is the result of applying a hash function $F$ to the key which will be used to compute the MAC of the packet in the next interval. By knowing the key commitment and the hash function $F$, the receiver can check whether the key disclosed in the next packet is indeed the key used to sign the previous payload.

In the following, $S$ and $R$ are the sender and a receiver, respectively, $T_{int}$ is the interval length, and $\Delta$ is the maximum time synchronisation error between the sender and the receiver. In addition, $MAC$ is a cryptographically sound (i.e., non invertible) MAC function, $F$ is a cryptographically sound hash-function, and $PK(S)$ is the sender public key.

$0a.\ R \longrightarrow S : n_R$

$0b.\ S \longrightarrow R : \{F(k_1).n_R\}_{PK(S)}$

...

$i.\ S \longrightarrow R : D_i\,.\,MAC(k_i, D_i) \qquad$ with $D_i = (M_i.F(k_{i+1}).k_{i-1})$

Following is a detailed description of each step of the protocol. With step $0a$, the receiver starts a new session by sending a fresh nonce $n_R$ to the sender. With step $0b$, the sender replies to the receiver with the same nonce together with a commitment $F(k_1)$ to the first key used. The message is signed using the public key $PK(S)$ of the sender. The use of public key cryptography and of a fresh nonce allows to guarantee the receiver that the legitimate sender

[1] This requirement can indeed be relaxed as shown in [LN03]

is starting a new protocol session. In addition, the receiver gains knowledge of the first key commitment which will be used to authenticate the first key. The receiver is therefore guaranteed, by the replied and the use of, that the legitimate sender is starting a new protocol session and know the first key commitment used to authenticate the first key.

In each step $i$, with $i \geq 1$, every packet contains the concatenation of a message $D_i$ and of the MAC of the current message $D_i$ with respect to the key $k_i$. Message $D_i$ is, in turn, the concatenation of the following messages: the payload $M_i$, the commitment $F(k_{i+1})$ to the key used in the next interval, and the key $k_{i-1}$ used to generate the MAC of the message sent in the previous step $i-1$.

Notice that the packet sent in the protocol step $i = 1$ contains a bogus key, since the packet in Step $0b$ has no payload.

Since both functions $MAC$ and $F$ are cryptographically sound, after receiving a packet, a receiver cannot immediately verify the validity of the MAC and, therefore, cannot immediately authenticate the received packet as originating from the sender.

On the other hand, the receiver has to immediately check that the receiving time of the current packet is compatible with the scheduling, namely that the key used to sign the current packet is not yet disclosed. This is done by checking the following condition:

$$\lfloor (T_c + \Delta - T_0)/T_{int} \rfloor = i \tag{4.1}$$

where $T_c$ is the local receiver time and $T_0$ is the protocol start time.[2].

To fully authenticate a received packet, the receiver has to wait until the sender discloses the corresponding key in the next time interval. A receiver can authenticate packet $p_i$, received in step $i \geq 1$, if the previous packet $p_{i-1}$ and next packet $p_{i+1}$ have been received, and the following conditions (called the TESLA *security condition*) hold:

[2]In its general form, TESLA allows for authenticating a message after up to $d$ intervals (with $d \geq 1$). For simplicity sake the version of Condition (4.1) we consider here is a simplified version where we assume $d = 1$ (meaning the next interval).

1. $p_{i-1}$ is an authenticated packet;

2. the key disclosed in $p_{i+1}$ (i.e., $k_i$) is the same key committed to in packet $p_{i-1}$. In other words, if $c$ is the commitment received in $p_{i-1}$, it has to check whether $F(k_i) = c$;

3. the application of the MAC function $MAC$, with respect to the key disclosed in $p_{i+1}$, to the concatenation of the payload, the disclosed key and the key commitment of packet $p_i$, yields the MAC component of $p_i$.

Condition (1) is handled differently for the first packet since it is the only one which uses public key cryptography for authentication. Notice that, in this scheme the loss of a packet causes not only the loss of a useful payload, but also the loss of a key. This prevents the ability to authenticate the previously received packet.

**Second Scheme**

To overcome the problem mentioned above, and make the protocol loss tolerant, the sender can *chain* the used keys. In other words, the sender generates a random key $k_N$ and generates the other keys by successive applications of a cryptographically sound hash function $H$ to $k_N$. Therefore, the second key is $k_{N-1} = H(k_N)$, the third key is $k_{N-2} = H(H(k_N)) = H(k_{N-1})$, and so on. In this way, a receiver can compute any key at position $i$ in the chain just by knowing some key at position greater than $i$ in the chain.

The drawback of this scheme is that the sender must precompute all the keys in advance so as to to schedule their disclosure in the appropriate packet.

Being able to compute a key from any subsequently received key make superfluous including the commitment in each packet. Using the commitment included in the initial authenticated packet, $F(K_1)$, a receiver can check the validity of the $i$-sh key received by checking that $F(K_1) = F(H^i(K_N))$. The starting commitment, chaining the keys, work as a commitment to the entire key chain.

The protocol, written in A-B notation, is similar to the one for the first scheme. The only relevant differences are in the structure of the messages sent at steps $i \geq 1$, and in the definition of the TESLA security condition. Each step $i$ (with $i \geq 1$) is replaced with:

$$i'.\ S \longrightarrow R : D_i\,.\,MAC(k_i, D_i) \qquad \text{with } D_i = (M_i.k_{i-1})$$

Assuming that the receiver has received and authenticated packet $p_v$ (with $v < i$), and received packet $p_{i+d}$ (with $d > 0$), conditions (2) and (3) of the TESLA security condition for authenticating $p_i$ become:

(2′) the key disclosed in $p_{i+d}$ (i.e., $k_{i+d-1}$) is the $(i + d - 1)$-th key of the chain committed to in packet $p_0$. In other words, it is necessary to check whether $F(H^{i+d-1}(k_{i+d-1})) = c_0$, where $c_0$ is the key commitment enclosed in the message of step $0b$;

(3′) the application of the MAC function $MAC$, with respect to the key used in the (possibly lost) packet $p_{i+1}$ (which corresponds to $H^{d-1}(k_{i+d-1})$), to the message components sent in packet $p_i$, yields the MAC component of $p_i$.

**Timed Features in the TESLA Protocol**

While there are works that show TESLA safety in an untimed setting [Arc02], the use of a time friendly notation make its specification simpler. In particular we need the ability to:

- a way to model the splitting of the packet stream into intervals;
- a way to check the TESLA security condition (e.g.: the keys disclosure).

### 4.1.3 Zhou-Gollmann efficient Non Repudiation Protocol

Non-repudiation protocols are concerned with preventing a principal to deny, *repudiate*, having been involved in some communication.

In particular the Zhou-Gollmann efficient Non Repudiation Protocol (ZGNRP) [ZG97] is interesting for its peculiar dependence on time.

There are many variants of the protocol. All describe a protocol that allows two participants exchange messages while protecting each party of the transaction against the other party that, in the case of a dispute, denies the occurrence of an action. In this protocol, the principal, Alice, must send a message to another participant, Bob, the purpose of the protocol is to ensure that Alice can not, at *any time*, deny having sent the message to Bob and vice versa, Bob can not deny having received the message from Alice.

The protocol allow the principals to acquire irrefutable evidences of the actions.

What the protocol try to ensure is a fairness property: at any time the protocol end with a principal having an *advantage* over the other, in particular, with a principal unable to prove the receipt/send of a message.

The protocol require the presence of a Trusted Third Party (TTP) that acting as a delivery authority and is providing services needed to keep the fairness of the protocol.

Each correct run of the protocol allow each principals to collect the following transmission evidences:

- Non-repudiation of origin (NRO): Bob, communicating with Alice, has obtained an irrefutable evidence that the origin of the message is indeed Alice; Alice, in the case of a dispute, cannot deny to have sent the message to Bob;

- Non-repudiation of receipt (NRR): Alice, communicating with Bob, has obtained an irrefutable evidence that the message was received by Bob; Bob, in the case of a dispute, cannot deny to have received a message from Alice;

The fairness property is verified if every time a participant have the NRO evidence the other have the NRR evidence and vice versa.

The idea behind the protocol is to split the message $M$ in half, a key $K$ and a commitment $C = \{M\}_K$. The commitment is sent to Bob while the

key is sent to the TTP that is responsible for its distribution to the parties together with a *receipt*. The receipt (CON) provided by the TTP is part of the NRO and NRR and can be used to solve an eventual dispute.

An important temporal aspect of the protocol is the guarantees that the principals always finish the protocol after a certain, predetermined, finite amount of time $T$ during which the TTP assure to keep the key.

Let:

- $M$ the message to send;
- $K$ symmetric key;
- $C$ Commitment, $M$ encrypted with $K$;
- $S_A$ A private key;
- $S_B$ B private key;
- $S_{TTP}$ TPP private key;
- $f_{EOO}$ flag Evidence of Origin;
- $f_{EOR}$ flag Evidence of Receipt;
- $f_{SUB}$ flag Evidence of Submission;
- $f_{CON}$ flag Evidence of Confirmation;
- $L$ nonce;
- $T$ predetermined timeout;
- $EOO = s_{S_A}(f_{EOO}, B, L, T, C)$, $s_k(m)$ is the signature of $m$ using the private key $k$;
- $EOR = s_{S_B}(f_{EOR}, A, L, T, C)$;
- $sub_K = s_{S_A}(f_{SUB}, B, L, K)$;
- $con_K = s_{S_{TTP}}(f_{CON}, A, B, L, K)$.

The protocol, in A-B notation, is the following:

1 $A \rightarrow B : f_{EOO}, B, L, T, C, EOO$

2 $B \rightarrow A : f_{EOR}, A, L, EOR$

3 $A \rightarrow TTP : f_{SUB}, B, L, T, K, sub_K$

4 $B \leftrightarrow TTP : f_{CON}, A, B, L, K, con_K$

4' $A \leftrightarrow TTP : f_{CON}, A, B, L, K, con_K$

In step 1, Alice has sent the commitment to Bob, Bob has not the message still as he is lacking the key $K$ so the property of fairness continues to be maintained. Similarly in step 2, the sending to Alice of a message confirming the correct receiving of the commitment from Bob does not change the fairness because the key has not yet been sent and without such key Bob cannot access the message. After this initial commitment exchange, the following steps involve the TTP to distribute the key and the evidences. The sending of the key in step 3 by Alice to the TTP still does not give any advantage to the principals. The step 4 and 4' are the most important steps of the protocol; here Alice receive, from the TTP, the receipt stating the correct sending of the key and the commitment and Bob receive, from the TTP, the key and the receipt stating the correct receipt of the commitment and the key.

### Timed Features in the Zhou-Gollmann efficient Non Repudiation Protocol

To model correctly the ZGNRP protocol we need to:

- model the duration of the protocol. As already said the protocol have a finite predetermined duration to end while trying to keep its fairness;
- model the transmission channels.

The last point is of particular relevance in fact the communication channels, and ultimately the subsumed intruder model, can delay, lose or otherwise not correctly deliver the messages between the principals and the TTP causing situations where a principals does not know whether a protocol run is finished or not and, consequently, causing a violation of the fairness property.

We will show in a following chapter that the protocol if fair as long as the communication channels between the principals and the TTP are safe (i.e.: the delivery of the messages is guaranteed).

### 4.1.4 A Timed Specification Language

The non-repudiation Zhou-Gollmann protocol, the TESLA authentication protocol and the Wide Mouthed Frog protocol are all examples of protocols that cannot be correctly modelled without accounting for time. The formal framework generally employed to reason about time, in the context of finite state machines, is that of Timed Automata, supported by the framework of the temporal logics for the specification of its requirements.

However, the formalism of Timed Automata cannot be employed by protocol designer as a specification formalism in itself, being a too low level formalism, unsuited to express high-level specifications of security protocols. For this reason, in this thesis we propose a temporal extension of the, state of the art, specification language $HLPSL$ called *Timed* $HLPSL$ ($THLPSL$).

In particular, the proposed extension of $HLPSL$ introduces four kinds of temporal features: (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.

#### $THLPSL$ Syntax

The proposed extension $THLPSL$ allows 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.

Constraints on delay/timeout and message disclosure/expiration are expressed with respect to the occurrence of a transition executed by a participant in the protocol.

To allow for this extension, we introduce a new 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 new feature will allow for expressing time constraints relative to occurrences of events (referred to by transition labels) taking place within specific role instances. We also replace the original HLPSL constructs for channel declaration and transition schemas with two new constructs and extend the set of terms as follows:

**Timed channels:** the channel variable declaration has two additional parameters. The new operator takes the form `C:channel(dy,lb,up)`, specifying a Dolev-Yao channel with minimum transmission delay `lb` (a rational number in $\mathcal{Q}_{\geq 0}$) and maximum transmission delay `up` (a rational number in $\mathcal{Q}_{\geq 0} \cup \{\infty\}$);

**Timed transitions:** a new transition operator, replacing the original one, takes six parameters, and has the form `>>(t1,t2,lb,ub,RI,label)`, where

RI is a formal parameters of the current role of type role_instance, t1 and lb are rational numbers in $\mathcal{Q}_{\geq 0}$, t2 and ub rational numbers in $\mathcal{Q}_{\geq 0} \cup \{\infty\}$, and label is a transition label. It specifies a transition that will be enabled between time t1 and t2 relative to the execution of the transition labelled label within the role of role instance RI, that will complete between time lb and ub. Similarly a new transition operator, takes four parameters, and has the form ->(lb,ub,RI,label), where RI is a formal parameters of the current role of type, lb is a rational number in $\mathcal{Q}_{\geq 0}$, t2 and ub rational number in $\mathcal{Q}_{\geq 0} \cup \{\infty\}$, and label is a transition label. It specifies an urgent transition, a transition that will fire as soon as enabled, time is not allowed to pass in the starting state and that will complete between time lb and ub relative to the execution of the transition labelled label within the role of role instance RI;

**Timed messages:** the class of terms is extended to express timed messages by adding terms of the form X[dt,et,RI,label], where RI is a formal parameter of type role_instance, $X$ a variable of type text, text (fresh), key or the result of a hashing operator, dt is a rational number in $\mathcal{Q}_{\geq 0}$ and ut a rational number in $\mathcal{Q}_{\geq 0} \cup \{\infty\}$ and label is a transition label. It intuitively represents a term X that will be disclosed between time dt and et relative to the execution of the transition labelled label within role instance RI, and it will expire after the temporal bound et. Moreover, we add two predicates of the form EXP(X), DISC(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, respectively, expired, disclosed. We also assume to have an additional label start which represents a fictitious transition taken at time 0 to initialize the main role. Notice that the new construct for transition schema still allows for expressing untimed transitions. In particular, a transition without any temporal constraints (neither delay/time out nor duration constraints) will now take the form >>(0,$\infty$,0,$\infty$,RI,start), a transition having delay/time out but no duration constraints takes the form >>(t1,t2,0,$\infty$,RI,label), while a transition having no delay/time out constraints but duration constraints takes the form >>(0,$\infty$,lb,ul,RI,start).

### $THLPSL$ Semantics

In an attempt to link back to previous consolidated results and theories we provide the semantic of $HLPSL$ in terms of its translation into Extended Timed Automata. The translation into XTA ensures that our extension enjoys a well-defined semantics and allows the use of some automated testing tools (e.g.: the UPPAAL model checker) that make the formalism of automata timed their means of specification. For the sake of simplicity, we will assume that in a $THLPSL$ specification (a) no structured datatypes are used, (b) transitions are uniquely labelled and (c) each transition contains at most one send or one receive operation (d) there is only one main composition role instantiating in parallel all the transition roles.

The intuition underlying the translation of Timed HLPSL into $XTA$ is the following:

- there is an automaton for each role instance, modelling the behaviour of the role instance within the protocol;
- there is an intruder automaton, modelled as a Dolev-Yao intruder;
- there is no direct synchronisation between role instances. Message exchange is modelled by synchronising the sender with the intruder via a channel, and then synchronising the intruder with the receiver via another channel.

The network of automata is therefore composed of $n+1$ automata, where $n$ is the number of role instances. The automaton for $i$-th role instance is denoted by $\mathrm{RI}_i$, while the automaton for the intruder will be denoted by $IA$.

Let $\mathcal{X}$ be a set of variables (including names for formal parameters), partitioned according to the builtin THLPSL types. Therefore, we have agent variables $\mathcal{X}_A$, text variables $\mathcal{X}_T$, key variables $\mathcal{X}_K$, channel variables $\mathcal{X}_C$, role instance variables $\mathcal{X}_{RI}$, and nonces variables $\mathcal{X}_{\mathcal{NC}}$. Let $SM_{\mathcal{X}}$ be the set of THLPSL terms built from $\mathcal{X}$, as defined in the previous sections. Given a term $T$, $VAR(T)$ denotes the set of the variables occurring in $T$, while $VAR'(T)$ the set of the variables occurring primed in $T$. Moreover, for

a term $T \in SM_{\mathcal{X}}$, $\overline{T}$ denotes the term obtained by replacing every occurrence of a primed variable $X'$ with the variable $X$. For a term $T \in SM_{\mathcal{X}}$, $\widetilde{T}$ denote the term obtained by removing temporization from every occurrence of a timed message occurring in $T$, eg.: substituting $m$ for $m[dt, et, RI, label]$ or $m^{\dagger}$.

We also define a collection of concrete, pairwise disjoint, domains, one for each THLPSL variable type: a set of agents $\mathcal{A}$, text messages $\mathcal{M}$, natural numbers $\mathcal{N}$, symmetric and public keys $\mathcal{K}$, hash functions $\mathcal{H}$, channels $\mathcal{C}$, and nonces $\mathcal{NC}$. Let $\Sigma$ be the union of those sets. Moreover, we add two a distinguished (w.r.t. $\Sigma$ ) sets, the set of role instances $\mathcal{RI}$ and labels $\mathcal{L}$.

Given the collection of concrete domains, we can define the set $SM_{\Sigma}$ of *ground messages* as the set obtained from THLPSL terms by (a) instantiating any variable occurring in a term by a concrete element in $\Sigma$, chosen in a type preserving way, and (b) adding a ground message of the form $m^{\dagger}$, for each $m \in \mathcal{M} \cup \mathcal{K} \cup \mathcal{NC}$, representing expired timed messages.

Let $\rho : \mathcal{X} \to SM_{\Sigma}$ be a partial valuation function associating variables to ground messages. Valuation functions are associated with role instances. Intuitively, a valuation function encodes the current set of ground messages the corresponding role instance has currently received or generated. Henceforth, we assume that, for each formal parameter $X$ of the role instance, $\rho(X)$ is set to the actual parameter associated with $X$ in the main composition role. Given a term $T$ without primed variables and a valuation $\rho$ defined on all the variables in $VAR(T)$, $T[\rho]$ denotes the ground message obtained from $T$ by substituting every occurrence of a variable $X \in VAR(T)$ with $\rho(X)$.

Recall that, for communication between HLPSL roles to successfully take place, it is required that the structure of the sent message $m$ matches the structure of the term $T$ specified in the receive action within a THLPSL transition. In particular, any unprimed variable $X$ in term $T$ requires that the value $\rho(X)$ is communicated in the matching part of $m$, while for any primed variable $X'$ any ground message of the same type of $X$ can be communicated. To capture this intuition, given a valuation function $\rho : \mathcal{X} \to SM_{\Sigma}$, we define a *matching relation* $\Rightarrow_{\rho} \subseteq SM_{\mathcal{X}} \times SM_{\Sigma} \times 2^{\mathcal{X} \times SM_{\Sigma}}$ which associates a term $T$ and a ground message $m$ with a partial valuation function $\rho_u$, binding

variables to ground messages, as follows:

- $(X', m) \Rightarrow_\rho \{(X, m)\}$, with $X \in \mathcal{X}$;
- $(X, m) \Rightarrow_\rho \emptyset$ if $\rho(X) = m$, with $X \in \mathcal{X}$;
- $(\{T\}_Z, \{m\}_k) \Rightarrow_\rho \rho_u$ if $\rho(Z) = k$ and $(T, m) \Rightarrow_\rho \rho_u$;
- $(T_1.T_2, m_1.m_1) \Rightarrow_\rho \rho_1 \cup \rho_2$ if $(T_1, m_1) \Rightarrow_\rho \rho_1$, $(T_2, m_2) \Rightarrow_\rho \rho_2$ and $\rho_1 \cup \rho_2$ is a partial function.

Notice that, in order to successfully match an encrypted term with an encrypted ground message (corresponding to decryption), the encryption key must be known ($\rho(Z) = k$ in the third clause above). Moreover, the matching operation cannot detach the timing attributes (possibly associated with a ground message) from a timed ground message.

To model generation of fresh nonces, we introduce a *nonce generation function* as an injective function $NC : RI \times \mathcal{N} \rightarrow \mathcal{NC}$, mapping a role instance and a natural number onto an element of $\mathcal{NC}$. Given a role instance $r$ and a nonce generation function $NC$, we define the *nonce assignment function* $AF^r_{\mathcal{NC}} : 2^{\mathcal{X}} \times \mathcal{N} \rightarrow 2^{\mathcal{X} \times \mathcal{NC}}$ such that $AF^r_{\mathcal{NC}}(\{X_1, ..., X_k\}, i) = \{(X_1, nc_{i+j_1}), ..., (X_k, nc_{i+j_k})\}$ and for all $1 \leq s \leq k$, the following must hold: (a) $1 \leq j_s \leq k$, (b) $nc_{i+j_s} = NC(r, i + j_s)$, and (c) $j_s \neq j_t$ if $s \neq t$.

Given a set of ground messages $M$, a role instance $r$ and a label $l$ occurring in the role of $r$, let $TM(M, r.l) = \{m[k, z, r, l] : m[k, z, r, l]$ occurs in some element of $M$ and $k > 0\}$ and $TM_0(M, r.l) = \{m[0, z, r, l] : m[0, z, r, l]$ occurs some element of $M\}$, which denote, respectively, the set of undisclosed and disclosed timed messages occurring in $M$ referring to label $l$ in role instance $r$. We also define:

$$Dis(M, r.l) = \min_{m[k,z,r,l] \in MT(M,r.l)} \{k\} \qquad Exp(M, r.l) = \min_{m[0,z,r,l] \in MT_0(M,r.l)} \{z\}$$

denoting, respectively, the minimum disclosure and expiration times of the timed messages in $M$ referring to label $l$ in role instance $r$. Given a valuation function $\rho$, let $M_\rho = \{m : \rho(x) = m$ for some $x \in \mathcal{X}\}$ be the set of ground messages in $\rho$.

Let us consider the XTA channels needed in the translation. Recalling that message exchange between role instances is modeled by synchronization with the intruder via a pair of channels, one for message sending and one for message delivery, and that channels in XTA do not convey values, for each THLPSL channel $c \in \mathcal{C}$ and ground message $m$, we add a XTA channel $c_s(m)$, denoting the *sending channel* for $m$, and the XTA channel $c_r(m)$, denoting the *receive channel* for $m$. Role instances are only allowed to execute output actions on sending channels and input actions on receive channels. Conversely, the intruder can perform input actions on sending channels and output actions on receive channels. Let $\Xi_g$ be the set of sending/receive channels.

As far as clocks are concerned, for each role instance $r$ and transition label $l$ in the role of $r$, we have a clock $CK(r.l)$. Let $CK_g$ be the set of clocks for the pairs $\langle r, l\rangle$, plus an additional clock $CK(start)$ used to refer to the label `start` in any role.

Given a main composition role of the form:

```
R1(actual_parameter) /\ ... /\ Rn(actual_parameter)
```

where $n$ roles are composed in parallel, we define a XTA $\langle \mathrm{RI}_1 \parallel \ldots \parallel \mathrm{RI}_n \parallel IA, \Xi_g, CK_g\rangle$ as follows.

**Modeling role instances.** The Timed Automaton $\mathrm{RI}_i$ for role instance $ri$, is the tuple $\mathrm{RI}_i = \langle L_i, L_{i0}, Ck_i, I_i, \delta_i\rangle$, where $L_i$ is the set of tuples either of the form $\langle \rho, i\rangle$ or $\langle l, \rho, i\rangle$, where $\rho$ is a partial valuation function of the role instance, $i$ is a natural number corresponding to the last index used for nonce generation by the role instance, and $l$ is a label in $\mathcal{L}$; $L_{i0}$ is the set of initial locations of the form $\langle \rho, 0\rangle$, where $\rho$ satisfies (with the usual meaning) the *init predicate* of the role. The set $Ck_i$ of local clocks contains a clock $d$, used to model the duration of transitions (as specified by the values $lb$ and $up$ in THLPSL transitions).

The intuition underlying the translation of the temporal features of THLPSL into XTA is the following:

- delay/timeout for a transition of the form `>>(t1,t2,lb,ub,role_inst,l)`

is modeled by guarding the XTA transitions with the clock constraint $t1 \leq CK(r.l) \leq t2$, where $r$ is the actual value of `role_inst` (notice that $CK(r.l)$ is reset whenever the transition labeled $l$ is performed by role instance $r$, conforming to the timed automata model we assume that $CK(r.l)$ is resetted at the start if the transition labeled $l$ of the role instance $r$ is still to be taken);

- duration of transitions of the form `>>(t1,t2,lb,ub,role_inst,l)` is modeled by splitting the transition into a sequence of two XTA transitions, the first one used to reset the clock $d$ measuring the required duration time, and the second transition guarded by the clock constraint $lb \leq d \leq ub$. Notice that, states of the form $\langle l, \rho, i\rangle$ corresponds to the intermediate states in the two step sequence above. To force the completion of the transition within the required bound `ub`, an invariant of the form $d \leq ub$ is associated to the intermediate state;

- urgency of transitions in the form `->(lb,ub,role_inst,l)` is modeled using invariant on the starting state;

- disclosure and expiration of a timed message of the form `m[dt,et,r,l]` is modeled by transitions guarded with the clock constraint $CK(r.l) = dt$ and $CK(r.l) = et$, respectively. Transitions disclosing the message simply substitute each occurrence of $m[dt, et, r, l]$ in the valuation $\rho$ with $m[0, et, r, l]$ (representing the disclosed message), while transitions for message expiration substitute every occurrence of $m[0, et, r, l]$ with the expired message $m^{\dagger}$.Clearly, in order to allow correct disclosure (resp., expiration) with respect to the flow of time, we need to associate invariants to each XTA state. The *Timed Messages Invariant* for states $\langle \rho, i\rangle$ and $\langle l, \rho, i\rangle$, in symbols $TMI(M_\rho)$, is the following:

$$\bigwedge_{r.l:MT(M_\rho,r.l)\neq\emptyset} CK(r.l) < Dis(M_\rho, r.l) \ \wedge \bigwedge_{r.l:MT_0(M_\rho,r.l)\neq\emptyset} CK(r.l) < Exp(M_\rho, r.l)$$

Intuitively, the invariant requires that the first disclosure (resp., expiration) to be executed is the one corresponding to the least disclosure

time $Dis(M_\rho, r.l)$ and expiration time $Exp(M_\rho, r.l)$ ranging over the sets of role instances and labels referenced in the timed messages in $M_\rho$;

- channels delays are modeled within the intruder automaton.

The relation $\delta_i$ and the invariant map $I_i$ are then defined as follows:

**State Transition** or each transition `lt. ps >>(t1,t2,lb,ub,RI,l) ps'`, we add the following XTA transitions:

$$\langle \rho, i \rangle \xrightarrow{\tau,\Delta,Clk} \langle lt, \rho', j \rangle \quad \text{where:}$$

- $\rho \models ps$,
- $\rho' \models ps'$;
- $Clk = \{d, CK(r.lt)\}$ with $r = \rho(RI)$ and $\Delta$ is $t1 \leq CK(r.l) \leq t2$ with $r = \rho(RI)$;

**Message Sending** for each transition `lt. ps >>(t1,t2,lb,ub,RI,l) C(T) /\ ps'`, we add the following XTA transitions:

$$\langle \rho, i \rangle \xrightarrow{c_s(m)!,\Delta,Clk} \langle lt, \rho', j \rangle \quad \text{where:}$$

- $\rho \models ps$ and $\rho(C) = c$ for the channel variable $C$,
- $\rho' \models ps'$, with $\rho'(X) = \rho(X)$, if $X \in VAR(T)$, and $\rho'(X) = \rho_s(X)$, if $X \in VAR'(T)$, and $\rho_s = AF^r_{\mathcal{NC}}(VAR'(T), i)$;
- $m = \overline{T}[\rho']$ and $j = i + |VAR'(T)|$;
- $Clk = \{d, CK(r.lt)\}$ with $r = \rho(RI)$ and $\Delta$ is $t1 \leq CK(r.l) \leq t2$ with $r = \rho(RI)$;

**Message Receive** for each transition `lt. ps /\ C(T) >>(t1,t2,lb,ub,RI,l) ps'`, we add the following XTA transitions:

$$\langle \rho, i \rangle \xrightarrow{c_r(m)?,\Delta,Clk} \langle lt, \rho', i \rangle \quad \text{where:}$$

- $\rho \models ps$ and $(T, m) \Rightarrow_\rho \rho_u$;
- $\rho' \models ps'$, with $\rho'(X) = \rho(X)$, if $X \in VAR(T)$, and $\rho'(X) = \rho_u(X)$, if $X \in VAR'(T)$;
- $Clk = \{d, CK(r.lt)\}$ with $r = \rho(RI)$ and $\Delta$ is $t1 \leq CK(r.l) \leq t2$ with $r = \rho(RI)$;

**Transition Duration** for each state of the form $\langle lt, \rho, i\rangle$, for some label $lt$, we add the following transitions:

$$\langle lt, \rho, i\rangle \xrightarrow{\tau,\Delta,\emptyset} \langle \rho, i\rangle$$

where $\Delta$ is $lb \leq d \leq ub$;

**Urgent Transition** for each transition `lt. ps ->(lb,ub,RI,l) ps'`, we add the following XTA transitions:

$$\langle \rho, i\rangle \xrightarrow{\tau,\Delta,Clk} \langle \rho', i\rangle \quad \text{where:}$$

- $\rho \models ps$;
- $\rho' \models ps'$;
- $Clk = \{d, CK(r.lt)\}$ with $r = \rho(RI)$ and $\Delta$ is $CK(r.l) \geq lb$ with $r = \rho(RI)$;

**Message Disclosure** for any pair $\langle r, l\rangle$ we add the following transitions:

$$\langle lt, \rho, i\rangle \xrightarrow{\tau,\Delta,\emptyset} \langle lt, \rho', i\rangle \text{ and } \langle \rho, i\rangle \xrightarrow{\tau,\Delta,\emptyset} \langle \rho', i\rangle \quad \text{where:}$$

- $\Delta$ is $CK(r.l) = k$, with $k = Dis(M_\rho, r.l)$;
- $\rho'(X) = \rho(X)[m[k, z, r, l]/m[0, z, r.l]]$ for any $X \in \mathcal{X}$, for some $m$;

**Message Expiration** for any pair $\langle r, l\rangle$ we add the following transitions:

$$\langle lt, \rho, i\rangle \xrightarrow{\tau,\Delta,\emptyset} \langle lt, \rho', i\rangle \text{ and } \langle \rho, i\rangle \xrightarrow{\tau,\Delta,\emptyset} \langle \rho', i\rangle \quad \text{where:}$$

- $\Delta$ is $CK(r.l) = z$, with $z = Exp(M_\rho, r.l)$

- $\rho'(X) = \rho(X)[m[0, z, r, l]/m^{\dagger}]$ for any $X \in \mathcal{X}$, for some $m$;

**Invariants** to each state of the form $\langle \rho, i \rangle$ not created by an urgent transition we associate the invariant $TMI(M_\rho)$ to enforce message disclosure/expiration, we associate $(CK(r.l) < ub) \wedge TMI(M_\rho)$ with $r = \rho(RI)$ otherwise; while to any state of the form $\langle lt, \rho, i \rangle$, such that $ub$ is the upper bound on the duration of the transition labeled $lt$, we associate the following invariant $(d \leq ub) \wedge TMI(M_\rho)$, for message disclosure/expiration and transition completion.

**Modeling the intruder.** By observing the traffic over the network, a Dolev-Yao intruder extends its knowledge, and, from its knowledge, can compose and send fraudulent messages to honest participants. To model this ability, we define the derivation relation $\vdash$, which determines the messages that the intruder is able to construct/deconstruct from a set $M$ of (known) messages. $\vdash \subseteq 2^{SM_\Sigma} \times SM_\Sigma$ is the smallest relation closed under the following rules:

- $M \vdash m$ if $m \in M$;
- $M \vdash m_1.m_2$ if $M \vdash m_1$ and $M \vdash m_2$;
- $M \vdash \{m\}_k$ if $M \vdash k$, $k \in \mathcal{K}$ and $M \vdash m$;
- $M \vdash m_1$ and $M \vdash m_2$ if $M \vdash m_1.m_2$;
- $M \vdash m$ if $M \vdash k$, $k \in \mathcal{K}$ and $M \vdash \{m\}_k$;
- $M \vdash h(m)$ if $h \in \mathcal{H}$ and $M \vdash m$;

In addition, for each $c \in C$ such that $c$ occurs as an actual parameter of type `channel(dy,lb,ub)`, $inf(c)$ denotes $lb$ and $sup(c)$ denotes $ub$, an $CK(c)$ is a new clock which is used to model channel delay. We denote the set of clocks associated to channels with $Ck_{IA}$ .

Recall that massage exchange between honest participant is modeled by a pair of synchronizations, one from the sender to the intruder and one from the intruder to a receiver. Since a Dolev-Yao intruder is allowed to block

messages, in our model the second synchronization is not guaranteed to take place. Therefore, the delay of a communication can be interpreted as the delay that the intruder introduces to complete the second synchronization. In addition the intruder is allowed to generate and autonomously send messages along channels not involved in a communication among participants. Therefore, we require fulfillment of delay constraints on channels only for the first type of communication, called Message Delivery. For the second type of communication (Autonomous Sending) no delay constraint is imposed. In order to model delay constraints on Message Delivery, the intruder stores the set of channels (*channels queue*) on which the first synchronization (from the sender to the intruder) has occured without the second synchronization having been completed. The channel queue does not store the messages sent along the channels, since the intruder is allowed to alter the content of communicated messages.

The intruder automaton $IA$ is the tuple $\langle L, L_0, Ck_{IA}, I, \delta \rangle$, where $L \subseteq 2^{SM_\Sigma} \times 2^{\mathcal{C}}$ is the set of intruder location. Each location is a pair $\langle M, Q \rangle$, where $M$ is a set of known messages and $Q$ a channel queue. $L_0 \subseteq L$ are the intruder initial locations (the intruder initial knowledge, as specified by the *knowledge* predicate, with the empty channel queue), $Ck_{IA}$ the set of clocks for channels, $I$ the invariant map, and $\delta$ the transition relation. The relation $\delta$ and the map $I$ are defined as follows:

**Message Receive** for each location $\langle M, Q \rangle \in L$, ground message $m \in SM_\Sigma$, and channel $c \in \mathcal{C}$ the following transition is added:

$$\langle M, Q \rangle \overset{c_s(m)?, \top, \{CK(c)\}}{\longrightarrow} \langle M \cup \{m\}, Q \cup \{c\} \rangle$$

**Message Delivery** for each location $\langle M, Q \rangle \in L$, ground message $m \in SM_\Sigma$ such that $M \vdash m$, and channel $c \in Q$ the following transition is added:

$$\langle M, Q \rangle \overset{c_r(m)!, \Delta, \emptyset}{\longrightarrow} \langle M, Q \setminus \{c\} \rangle$$

where $\Delta$ is $inf(c) \leq CK(c) \leq sup(c)$);

**Autonomous Sending** for each location $\langle M, Q \rangle \in L$, ground message $m \in$

$SM_\Sigma$ such that $M \vdash m$, and channel $c \in \mathcal{C} \setminus Q$ the following transition is added:

$$\langle M, Q\rangle \xrightarrow{c_r(m)!,\top,\emptyset} \langle M, Q\rangle$$

**Message Disclosure** for each pair $(r, l)$ such that $TM(M, r.l) \neq \emptyset$ the following transitions are added:

$$\langle M, Q\rangle \xrightarrow{\tau,\Delta,\emptyset} \langle M', Q\rangle \quad \text{where:}$$

- $\Delta$ is $CK(r.l) = k$, with $k = Dis(M, r.l)$;
- $M' = M[m[k, z, r, l]/m[0, z, r, l]]$, for some $m$.

**Message Expiration** for each pair $(r, l)$ such that $TM_0(M, r.l) \neq \emptyset$ the following transitions are added:

$$\langle M, Q\rangle \xrightarrow{\tau,\Delta,\emptyset} \langle M', Q\rangle \quad \text{where:}$$

- $\Delta$ is $CK(r.l) = z$, with $z = Exp(M, r.l)$;
- $M' = M[m[0, z, r, l]/m^\dagger]$, for some $m$.

**Invariant** the Timed Message Invariant $TMI(M)$ is associated to each location $\langle M, Q\rangle \in L$.

Notice that protocol specifications written in THLPSL are not guaranteed to lead to XTA having a finite number of locations. On the other hand, it is possible to define simple syntactic restrictions on the specification (e.g., restrictions on the presence of loops, finiteness of the concrete domains, bounds on the size of the set of messges the intruder can generate, etc.) which guarantee finiteness of the sets of locations of the translation. These restrictions still allow for specifying and verifying interesting classes of protocols.

# Chapter 5

# Timed Security Protocols Model Checking Framework

In this chapter we focus on the implementation of a model checking framework for the analysis of security protocols which employs the presented THLPSL language as a specification language. In section 5.1 we will introduce the Timed Protocol Model Checker (TPMC) framework and provide detail on its implementation. To illustrate how our framework applies, in section 5.2 we will also provide the specification in $THLPSL$ of the protocols presented in section 4.1. We will end in section 5.3 showing how the framework perform and, where possible, how its performance compare against state of the art protocols verifiers.

## 5.1 TPMC: Timed Security Protocols Model Checking Framework

The TPMC (Timed Protocols Model Checker) tool we developed for the analysis of timed security protocols 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 4.1.4. For the sake of ease of definition, such a semantics maps THLPSL specification onto

pure $XTA$s, 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, 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 expressiveness of UPPAAL $XTA$s, can be more efficiently employed for implementation purposes.

### 5.1.1 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. As we have seen in Chapter 2, UPPAAL $XTA$ are an extension of Timed Automata, where a parallel composition, in the style of CCS, of a collection of Timed Automata communicate by means of channels and the communication style is handshaking. Input symbols in a $XTA$ are replaced by channel names. 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 $a$. Moreover, $XTA$ can use (boolean and integer) variables and arrays. Therefore, the guard of a $XTA$ transition may also constraints values of variables and array elements besides clocks and the updates are also generalised allowing also assignments involving variables and arrays. Refer to Chapter 2 for a more detailed insight.

As previously said, the formal semantics of $THLPSL$ has be given in Section 4.1.4 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 sub message, compatibly with its knowledge of the necessary cryptographic keys. Conversely, known sub messages 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.

Therefore, the translation defined in Section 4.1.4 in order to give the semantics of the THLPSL, cannot be immediately exploited for implementation purposes, and a more succinct encoding of $THLPSL$ is required.

For a succinct encoding, the translation implemented in TPMC exploits the ability of UPPAAL 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. The net result of this encoding, is that most of the burden of managing the explosion of the space of states is left to the model checker.

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 synchronisations, 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 $XTA$ synchronisation 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 initialisation 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 $\rho_{ri}$ 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 $\chi^M_{ri}$ encodes the result of a structure preserving unification mechanism between message terms expected by the receiver and ground terms sent by a sender.

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 $\rho_{ri}$ and $\chi^M_{ri}$) of the message variables $\{X_1, ..., X_k\}$ occurring in the transition, as shown in Fig. 5.1. The first $XTA$ transition models the effect of the THLPSL transition, while the second one models its duration. With reference to Fig. 5.1, given $\theta =$

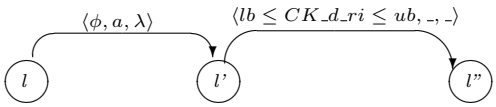



Figure 5.1: $XTA$ transitions encoding a timed transition.

$\{m_1, ..., m_k\}$ a possible instantiation of the message variable $\{X_1, ..., X_k\}$ according to $\rho_{ri}$:

- $l$ 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.
- $\phi$ 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 $\theta$.
- $\lambda$ 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 $\phi$ enables the transition when the current state: (i) assigns to the unprimed variables the ground messages assigned by $\theta$; and (ii) satisfies all the conjuncts in `MPred`. The update $\lambda$ 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 $\rho_{ri}$ and $\chi^M_{ri}$) 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 $\rho_{ri}$, and $\psi = \{u_1, ..., u_z\}$ a possible matching, according to $\chi^M_{ri}$, for the message variable $\{Y_1, ..., Y_z\}$ occurring primed in $M$, a pair of $XTA$ transitions as in Fig. 5.1 is added, where:

- $l$, $l'$, $l''$ and $\phi$ 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 $\theta$ and all the message variables in $\{Y_1, ..., Y_z\}$ by $\psi$;
- $\lambda$ 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. 5.1 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.

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

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
$\langle \mathrm{K[m]} = 1 \ \wedge \texttt{CK\_CHN} \geq lb\,,\ \texttt{C\_CHN\_r\_m!}, \_ \rangle$,
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

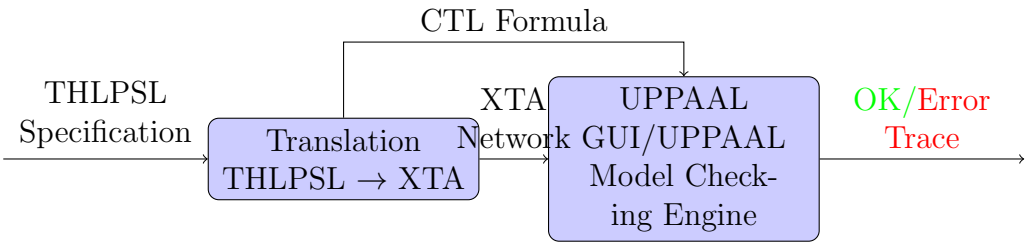



Figure 5.2: The architecture of the tool.

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:

$$(\texttt{F[lab\_ri]} \ \wedge \ \texttt{not D[m]}) \rightarrow \texttt{CK\_lab\_ri} \leq \texttt{dt}) \ \wedge$$
$$(\texttt{F[lab\_ri]} \ \wedge \ \texttt{D[m]} \ \wedge \ \texttt{not E[m]}) \rightarrow \texttt{CK\_lab\_ri} \leq \texttt{et})$$

### 5.1.2 Framework Architecture and Limits

The general architecture of the verification environment is depicted in Figure 5.2. The system is composed of two modules, a verification engine, namely the model checker UPPAAL [BY04], 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. The compiler, then, first pre-computes this set of the ground messages that can be generated during the protocol runs either by the legal participants to the protocol or by the intruder. Once computed the set of messages, the compiler generates: one automaton for each each role instance, modeling the behavior of the role instance; an automaton modeling the behavior of the intruder.

Notice that the full syntax of THLPSL, which allows for, e.g., integer variables and transition systems containing infinite loops generating fresh messages, can be used to specify system with an infinite number of locations. In addition, the language, similarly to HLPSL, does not restrict a protocol specification to a finite number of sessions, therefore allowing an infinite number of messages to be generated. These specification cannot be translated into a XTA network, since XTAs are constrained to a finite number of locations. Since decision problems are not decidable for the resulting protocol models (even in their untimed version), while the corresponding decision problems for XTAs are decidable, the THLPSL language is strictly more expressive than XTAs. To allow for automatic protocol analysis, our translation assumes THLPSL specifications with finite number of sessions, which do not allow for infinite generation of nonces within loops, and where integer variables take value in a bounded domain (e.g., this is the case for the variable used, in particular, to enforce a finite state control flow for each automaton).

As far as verification is concerned, the environment allows both to specify the CTL properties to be analyzed directly from the UPPAAL interface, and to specify a set of predefined security properties in a higher lever language within the goal section of a THLPSL specification. Currently, secrecy properties and (weak and strong) authentication properties are allowed within the goal section. In the latter case, the translator is responsible of automatically generating the corresponding CTL formula. Direct specification of CTL properties clearly requires knowledge of the logical formalism employed, therefore it is reserved to specialized users.

A *secrecy property* is a security property requiring that a given message term is kept secret by the protocol. Since the number of ground messages in

the protocol can be kept finite, this property can be encoded by a formula which checks, for every reachable state, that every ground message possibly instantiating the message term is never contained into the knowledge of the intruder.

An *authentication property* in THLPSL is specified using the `witness` and the `request` keywords. This kind of property cannot be directly encoded in the fragment of CTL allowed by UPPAAL. To encode it in UPPAAL, we use, for each ground message matching the message term involved in the authentication, two additional boolean variable, one for the request and one for the witness.

## 5.2 Timed Protocols Examples: Modelling in *THLPSL*

In this section we will propose possible encoding in *THLPSL* for the timed protocols presented in Section 4.1.

### 5.2.1 Wide Mouthed Frog Protocol

The temporal feature we identified in Section 4.1 as needed to model correctly the Wide Mouthed Frog Protocol were:

- the use of the timestamp exchanged by the principals;
- the timing in both principals action and channels.

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 parametrised 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 initialisation section, while the `transition` clause opens the section containing transition schemas. The transition schema, labelled `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.

### 5.2.2 TESLA Authentication Protocol

The modelling of the TESLA protocol in $THLPSL$ is not straightforward as the WMF protocols. In fact to model the TESLA protocol we need a way

to model the splitting of the packet stream into intervals and the condition in equation (4.1). While it is easy to model interval by assigning to each THLPSL transition a temporal constraint in the form of the enabling condition, the ability to account for the temporal displacement $\Delta$ between the sender and the receiver and the corresponding condition requires a suitable encoding.

We can account for the parameter $\Delta$ by combining temporal constraints on transitions and timed messages. The idea is to associate temporal constraints to the payload (i.e., payloads are modeled by timed messages) where the disclose/expiration time is shifted ahead, with respect the bounds of the interval time in which it is sent, of the amount of time $\Delta$.

In the following, we assume a value $\Delta = 2$ and a $T_{int} = 9$. The sender sends the first message between time 10 and 19. The validity of this message (i.e., the guarantee that the message has been sent by the sender) is between the sending time and the corresponding key disclosure time. On the other hand, due to the displacement $\Delta = 2$, the receiver must account for the maximum time synchronization error. Therefore, the receiver must consider the packet as valid, if received between time 10 and time 21. The complete specification of the sender is the following:

```
role Sender(S,R:agent, RCV,SND:channel(dy, 0, inf), F,MAC:function,
                       KP:public_key, RI:role_instance) played_by S
  def=  local State : nat, NB : text,
        M1,M2,M3 : text(fresh),
        K1, K2, K3, K4: symmetric_key, KB : symmetric_key
  init  State = 0
  transition
   req.   State = 0 /\ RCV(NB') >>(0,1,0,0,RI,start) State' = 1
   setup. State = 1 >>(1,10,0,0,RI,start)  State' = 2 /\
                                           SND(NB.F(K1)_(KP))
   s1.    State = 2 >>(10,19,0,0,RI,start) State' = 3
          /\ SND(M1'[10,21,start,RI].F(K2).KB.MAC(K1.M1'.F(K2).KB))
          /\ witness(S,R,m1,M1)
   s2.    State = 3 >>(19,28,0,0,RI,start) State' = 4
```

```
          /\ SND(M2'[19,30,start,RI].F(K3).K1.MAC(K2.M2'.F(K3).K1))
   s3.    State = 4 >>(28,37,0,0,RI,start) State' = 5
          /\ SND(M3'[28,39,start,RI].F(K4).K2.MAC(K3.M3'.F(K4).K2))
end role
```

Notice that the specification considers just four iterations of the protocol loop. This is essentially due to the decidability issues discussed at the end of the previous section, and to the fact that in this thesis we are only interested in checking the time dependent packet authentication mechanism implemented in TESLA.

In the local variable declaration section, the variable `State` represents the current state of the sender. The fresh text variables `M1`, `M2` and `M3` are the payloads of the second, third and fourth message (the first message has no payload). The keys `K1`, `K2`, `K3` and `K4` are the keys used to compute the MACs of the messages, and variable `NB` represents the nonce received by the receiver and is used to ensure freshness of the session. The key `KB` is the bogus key disclosed in the message in the first step. The `init` clause opens the variable initialization section, while the keyword `transition` opens the section containing transition schemas. Note that non initialized non nonces variables (i.e.: of type text(fresh)) like $K1$ are considered implicitly initialized to a unique value. The first transition labeled `req` is a receive transition that models the receipt of the fresh nonces sent by the receiver. It takes from a state where variable `State`$= 0$ to a state where `State`$= 1$, and all the other variables remain unchanged. The additional effect of the transition is that the content of the channel `RCV` is assigned to the variable called `NB`. The transition schema, labeled `setup`, is a send transition. Besides performing the state change from 1 to 2, it sends the message term `{F(K)}_(KP)` over the channel `SND`. This is the first message of the sender used to setup the protocol run. This message contains the commitment to the first key (`K1`) encrypted with its public key (`KP`). The transition scheme labeled `s1`, `s2`, `s3` send the messages to be authenticated. Let us consider the transition `s1` (the other are similar). The term `M1'[10,21,start,RI]` represents a fresh message, with disclosure/expiration interval between time 10 and 21, relative to the start of the protocol (label `start`). Timing messages is therefore used

to bind each payload to the temporal interval where the corresponding key has not yet been disclosed. Note also that each transition is timed. For instance, the decoration `>>(10,19,0,0,RI,start)` of transition `s1` means that the the transition is enabled only between time 10 and 19 from the start of the protocol run. This time constraints models the assumption that the interval size $T_{int}$ is of length 9. The use of timed messages to model intervals is exploited in the receiver to model the TESLA security condition. A parametric receiver is, then, defined as follows:

```
role Receiver(S,R:agent, RCV,SND:channel(dy, 0, inf), F,MAC:function,
              KP:public_key, RI:role_instance) played_by R
  def= local State : nat, NB : text(fresh),
        M1, M2, M3 : text, MAC1, MAC2, MAC3 : text,
        KB, K1, K2  : symmetric_key, FK1, FK2, FK3, FK4 : text
  init  State = 0
  transition
   req.      State = 0 >>(0,inf,0,0,RI,start) State' = 1 /\ SND(NB')
   setup.    State = 1 /\ RCV(NB.FK1'_(KP)) >>(0,inf,0,0,RI,start)  State' = 2
   step1.    State = 2 /\ RCV(M1'.FK2'.KB'.MAC1') >>(0,inf,0,0,RI,start)
             State' = 3  /\ request(R,S,m1,M1)
   verstep1. State = 3  /\ not EXP(M1) =|> State' = 5
   step2.    State = 5  /\ RCV(M2'.FK3'.K1'.MAC2') >>(0,inf,0,0,RI,start)
             Statea' = 6
   verstep2. State = 6  /\ not EXP(M2) /\ FK1=F(K1) /\ MAC1=MAC(K1.M1.FK2.KB) =|>
             State' = 7
   step3.    State = 7  /\ RCV(M3'.FK4'.K2'.MAC3') >>(0,inf,0,0,RI,start)
             State' = 8
   verstep3. State = 8  /\ not EXP(M3)  =|> State' = 9
   verL.     State = 9  /\ FK2=F(K2) =|> State' = 10
   verMAC2.  State = 10 /\ MAC2=MAC(K2.M2.FK3.K1) =|> State' = 11
end role
```

The transition `req` sends a fresh nonce to the sender in order to start the protocol run. The transition `setup` is a receive transition, where a message with a structure compatible with `{NB.FK1'}_(KP)` is received along the

channel RCV. The ground value received for component NB must match the ground message associated to variable NB in the previous transition (where it was generated). The side effect is that the primed variable FK1' is assigned to the value of the corresponding term in the received message.

Since the sender and the receiver are not necessarily synchronized (due to possible local time displacement and delays caused by the intruder) the receiver must be allowed to receive messages in time intervals different from the one scheduled by the sender. Therefore, transitions of the receiver are not constrained to specific time bounds.

Transition verstep1 is responsible for checking the condition in equation (4.1) for the first message. The check is modeled by the predicate not EXP(M1), which verifies whether the payload contained in the received message is not yet expired (therefore, ensuring that it has been sent before the corresponding key was disclosed). This transition is an instantaneous transition, as the check must be performed as soon as the message is received.

Transition step2 receives the second message, while verstep2 performs both the checks of condition (4.1) for the newly received message and the authentication of the previously received one. This second check corresponds to conditions (2) (i.e., FK1=F(K1)) and (3) (i.e., MAC1=MAC(K1.M1.FK2.KB)) of the TESLA security condition. Analogously for the remaining transitions.

For the sake of exposition, below we show a specification of a simplified version of the second schema of TESLA, where the receiver is able to recover the loss of the second message only. The main differences with respect to the first schema are in the structure of the messages sent and received and, according to the description given in the previous section, in the way the receiver authenticates the packets.

As to the sender specification, the only difference is that instead of using the keys K1, K2, K3 and K4, it uses H(H(H(H(K)))), H(H(H(K))), H(H(K)), and H(K), respectively.

```
role Sender(S,R:agent, RCV,SND:channel(dy, 0, inf), F,H,MAC:function,
            KP:public_key, RI:role_instance) played_by S
  def= local State : nat, NB : text,
```

```
      M1,M2,M3,M4 : text(fresh), K : symmetric_key, KB : symmetric_key
  init  State = 0
  transition
   req.   State = 0 /\ RCV(NB')>>( 0,1,0,0,RI,start) State' = 1
   setup. State = 1 >>( 1,10,0,0,RI,start) State' = 2
          /\ SND( NB.F(H(H(H(H(K)))))_(KP) )
   s1.    State = 2 >>(10,19,0,0,RI,start) State' = 3
          /\ SND( M1'[10,21,start,RI].KB.MAC(H(H(H(H(K)))).M1'.KB)
          /\ witness(S,R,m1,M1)
   s2.    State = 3 >>(19,28,0,0,RI,start) State' = 4 /\
          SND( M2'[20,30,start,RI].H(H(H(H(K)))).
               MAC(H(H(H(K))).M2'.H(H(H(H(K)))))
   s3.    State = 4 >>(28,37,0,0,RI,start) State' = 5 /\
          SND( M3'[30,39,start,RI].H(H(H(K))).MAC(H(H(K)).M3'.H(H(H(K))))
end role
```

As to the receiver, recall that in this version it also needs to check if a packet is lost and, if this is the case, perform the corresponding recover phase.

```
role Receiver(S,R:agent, RCV,SND:channel(dy, 0, inf), F,H,MAC:function,
              KP:public_key, RI:role_instance ) played_by R
  def= local State, HC : nat, NB : text(fresh),
      M1, M2, M3 : text, MAC1, MAC2, MAC3 : text,
      K : symmetric_key, KB, HK, HHK, HHHK, FHHHHKP : text
  init  State = 0  /\ HC = 0
  transition
   req.       State = 0 >> (0,inf,0,0,RI,start) State' = 1 /\ SND(NB')
   setup.     State = 1 /\ RCV(NB.FHHHHK'_(KP)) >>(0,inf,0,0,RI,start)
              State' = 2
   step1.     State = 2 /\ RCV(M1'.KB'.MAC1') >>(0,inf,0,0,RI,start)
              State' = 3  /\ request(R,S,m1,M1)
   verstep1.  State = 3 /\ not EXP(M1) =|> State' = 4
   step2.     State = 4 /\ RCV(M2'.HHHHK'.MAC2') >>(0,inf,0,0,RI,start)
              State' = 5
   lost2.     State = 4 >>(0,inf,0,0,RI,start) State' = 5 /\ HC' = 1
```

```
   verstep2.  State = 5 /\ HC = 0/\ not EXP(M2)
              /\ FHHHHK=F(HHHHK))
              /\ MAC1=MAC(HHHHK.M1.KB)=|> State' = 7
   step3.     State = 7 /\ RCV(M3'.HHHK'.MAC3') >>(0,inf,0,0,RI,start)
              State' = 8
   verstep3.  State = 8 /\ not EXP(M3)  =|> State' = 9
   verL.      State = 10 /\ FHHHHK=F(H(HHHK)) =|>
              State' = 11
   verMAC2.   State = 11 /\ MAC2=MAC(HHHK.M2.HHHHK) =|>
              State' = 12
   recovery2. State = 12 /\ HC = 1
              /\ MAC1=MAC(H(HHHK).M1.KB) =|> State' = 13
end
```

The check of the packet loss is modeled using two transition, `step2` and `lost2`, and additional flag `HC` which witnesses the possible loss of a packet. Transition `step2` is executed if the expected message is delivered in the correct interval. If no packet loss occurs, the receiver proceeds similarly to the first schema. If, on the other hand, the enabling interval of `step2` elapses without a message receipt, `lost2` is executed, setting the flag `HC` to 1. In this case, the receiver has to delay the authentication of the first message after the authentication of the third message (transition `recovery2`).

The `witness` and the `request` operators, occurring in the transitions `step1` of the receiver and `s1` of the sender, respectively, in both version of the specifications, are used to specify the *authentication property*. These operator in the roles of the parties which must authenticate each other, and are parameterized by the name of the parties involved in the authentication process and the message term with respect to the authentication must occur (e.g., a term corresponding to a password). Intuitively, the witness models the challenge for the authentication, while the matching request models the response to the challenge. In the first model, the transitions labeled `s1` in the Sender and `step1` in the Receiver contain a pair of witness and request terms which are used to specify the authentication property. Intuitively, this witness and request pair requires that pairs of instances of roles `S` and `R`,

instantiated with the same values for the parameters of agent type, must agree on the value of the message term `M1`. In other words, whenever the receiver `R` performs a request on a ground message instantiating `M1`, that message must have been previously sent (i.e., witnessed) by the sender `S`.

The above specifications are indeed safe. Therefore, the TESLA security condition, together with Condition (4.1), allows the receivers to correctly authenticate the packets. On the other hand, if one weakens the specification by avoiding, for instance, to check Condition (4.1) on the first packet (transition `verstep1` in the first schema), the same property becomes false.

### 5.2.3 Zhou-Gollmann efficient Non Repudiation Protocol

The encoding of the Zhou-Gollman efficient Non Repudiation Protocol in $THLPSL$ is the straightforward translation of its encoding in A-B notation. The only details missing from the A-B notation, are the encoding of the protocol duration $T$ and the channels timings. Following is the encoding in $THLPSL$ of the protocol with $T = 5$ and instant channels.

```
role Alice( A, B, TTP : agent,  SND, RCV : channel(dy, 0, 0),
            SNDTTP, RCVTTP : channel(dy, 0, inf ),
            SA, SB, STTP : public_key, K : symmetric_key, M : text )
                                                played_by A def=

    exists  State : nat,
        L : text(fresh),
        % Costanti
        FNRO : nat,
        FNRR : nat,
        FSUB : nat,
        FCON : nat,
        FGETA : nat,
        T : nat
```

```
    init State = 0 /\
        % Initialisation
        FNRO = 1
        FNRR = 2 /\
        FSUB = 3 /\
        FCON = 4 /\
        FGETA = 5 /\
        T = 5

    transition
    alice1. State = 0
        =>>(0,5)
    State'=1  /\ SND(FNRO.B.L'.T.M_(K).FNRO.B.L'.T.M_(K)_inv(SA)_(SB))

    alice2. State = 1  /\ RCV( FNRR.A.L.FNRR.A.L.T.M_(K)_(SB)_inv(SA) )
        =>>(0,5)
    State'=2

    alice3. State = 2
        =>>(0,5)
    State' = 3  /\ SNDTTP( FSUB.B.L.T.K.FSUB.B.L.T.K_inv(SA)_(STTP) )

    alice4. State = 3
        =>>(0,5)
    State' = 4  /\ SNDTTP(FGETA)

    alice5. State = 4  /\ RCVTTP( FCON.A.B.L.K.FCON.A.B.L.K_(STTP)_inv(SA) )
        =>>(0,5)
    State' = 5
end role

role Bob( A, B, TTP : agent,    SND, RCV : channel(dy, 0, 0),
          SNDTTP, RCVTTP : channel(dy, 0, inf ),
          SA, SB, STTP : public_key, K : symmetric_key, M : text )
                                          played_by B def=
```

```
exists  State : nat,
    L : text,
    Y : message,
    W : symmetric_key,
    % Constanta
    FNRO : nat,
    FNRR : nat,
    FSUB : nat,
    FCON : nat,
    FGETB : nat,
    T : nat

init State = 0 /\
    % Initialisations
    FNRO = 1 /\
    FNRR = 2 /\
    FSUB = 3 /\
    FCON = 4 /\
    FGETB = 6 /\
    T = 5

transition
bob1. State = 0 /\ RCV( FNRO.B.L'.T.Y'.FNRO.B.L'.T.Y'_(SA)_inv(SB) )
    =>>(0,5)
State' = 1

bob2. State = 1
    =>>(0,5)
SND( FNRR.A.L.FNRR.A.L.T.Y_inv(SB)_(SA) ) /\ State'=2

bob3. State = 2
    =>>(0,5)
State' = 3 / SNDTTP(FGETB)
```

```
    bob4. State = 3 /\ RCVTTP( FCON.A.B.L.W'.FCON.A.B.L.W'_(STTP)_inv(SB) )
        =>>(0,5)
    State' = 4
end role

role TServer( A, B, TTP : agent,
              SNDTTPA, RCVTTPA : channel(dy, 0, 0),
              SNDTTPB, RCVTTPB : channel(dy, 0, 0),
              SA, SB, STTP : public_key ) played_by TTP def=

    exists  State : nat,
        L : text,
        W : symmetric_key,
        % Constants
        FSUB : nat,
        FCON : nat,
        FGETA : nat,
        FGETB : nat,
        T : nat

    init State = 0 /\
        % Initialisation
        FSUB = 3 /\
        FCON = 4 /\
        FGETA = 5 /\
        FGETB = 6 /\
        T = 5

    transition
    ttp1. State=0 /\ RCVTTPA(FSUB.B.L'.T.W'.FSUB.B.L'.T.W'_(SA)_inv(STTP))
        =>>(0,5)
          State' = 1

    ttp2a. State = 1 /\ RCVTTPA(FGETA) =>>(0,5) State' = 2
    ttp2b. State = 1 /\ RCVTTPB(FGETB) =>>(0,5) State' = 3
```

```
    ttp3a. State = 2
        =>>(0,5)
    SNDTTPA(  FCON.A.B.L.W.FCON.A.B.L.W_inv(STTP) _(SA)) /\ State' = 1

    ttp3b. State = 3
        =>>(0,5)
    SNDTTPB(  FCON.A.B.L.W.FCON.A.B.L.W_inv(STTP) _(SB)) /\ State' = 1
end role

role Env( ) def=
composition
    Alice(Ai, Bi, TTPi, Sndi, Rcvi, Sndttpia, Rcvttpia,
                            SAi, SBi, STTPi, Ki, Mi)  /\
    Bob(Ai, Bi, TTPi, Rcvi, Sndi, Sndttpib, Rcvttpib,
                            SAi, SBi, STTPi, Ki, Mi) /\
    TServer( Ai, Bi, TTPi, Rcvttpia, Sndttpia, Rcvttpib, Sndttpib,
                            SAi, SBi, STTPi )
end role

goal

end goal

Env()
```

Differently from the other two protocols we can see, in the encoding of the TTP rule, the use of looping. The algorithm used in our tool, generally, cannot handle loops in protocols, this case is peculiar since the TTP does not generate any nonces. This particular use of loops, i.e.: without nonces generation, result in a finite state automaton and as such is handled.

Differently from the other two protocols the propertie to check is not an authentication or a secrecy one so need some appropriate encoding directly in CTL. The non repudiation property of the protocols, ultimately a fairness property, can be encoded as a reachability property. In detail we want that

when the Alice automaton is in any state generated by the label `alice5`, i.e.: Alice got its receipt (NRR), Bob is in a state generated by the label `bob4`, i.e.: Bob got its receipt (NRO).

The violation of fairness by Bob against Alice can be encoded in the CTL fragment used by UPPAAL as:

$$E <> (Bob1.Bob1_4 and !TServer2.TSe2_3 and TServer2.d > 5)$$

where:

- Bob1.Bob1_4 is the state where Bob got the secret (i.e.: Bob final state);
- TServer2.TSe2_3 is the state where the TTP got the Alice request;
- TServer2.d s the clock counting the duration of the protocol (5 time unit in the example).

The symmetric fairness violation by Alice against Bob:

$$E <> (Alice0.Ali0_5 and !TServer2.TSe2_2 and TServer2.d > 5)$$

where:

- Alice0.Ali0_5 is the state where Alice got the secret (i.e.: Alice final state);
- TServer2.TSe2_3 is the state where the TTP got the Bob request;
- TServer2.d s the clock counting the duration of the protocol (5 time unit in the example).

The encoding of those properties is, of course, not automatic and require expertise in the peculiar encoding used by the framework but is a demonstration of its flexibility. Adding support for goals different from the usual secrecy and authentication is a possible extension of the framework.

Checking the protocol using the above formulae we see that the properties is violated when the channels are not instantaneous.

### TTPs Variant

An interesting different variant of the protocol require the TTP to always wait for the request from both the principals before answering. This variant is safe way even when not using instantaneous channels.

```
role TServer( A, B, TTP : agent,
              SNDTTPA, RCVTTPA : channel(dy, 0, 0),
              SNDTTPB, RCVTTPB : channel(dy, 0, 0),
              SA, SB, STTP : public_key ) played_by TTP def=

    exists  State : nat,
        L : text,
        W : symmetric_key,
        % Constants
        FSUB : nat,
        FCON : nat,
        FGETA : nat,
        FGETB : nat,
        T : nat

    init State = 0  /\
        % Initialisation
        FSUB = 3  /\
        FCON = 4  /\
        FGETA = 5  /\
        FGETB = 6  /\
        T = 5

    transition
    ttp1. State=0 /\ RCVTTPA(FSUB.B.L'.T.W'.FSUB.B.L'.T.W'_(SA)_inv(STTP))
        =>>(0,5)
    State' = 1
```

```
    ttp2a. State = 1  /\ RCVTTPA(FGETA) =>>(0,5) State' = 2
    ttp2ap. State = 2 /\ RCVTTPB(FGETB) =>>(0,5) State' = 3

    ttp2b. State = 1 /\ RCVTTPB(FGETB) =>>(0,5) State' = 6
    ttp2a. State = 6 /\ RCVTTPA(FGETA) =>>(0,5) State' = 3

    ttp3a. State = 3
        =>>(0,5)
    SNDTTPA(  FCON.A.B.L.W.FCON.A.B.L.W_inv(STTP) _(SA)) /\ State' = 4

    ttp3b. State = 4
        =>>(0,5)
    SNDTTPB(  FCON.A.B.L.W.FCON.A.B.L.W_inv(STTP) _(SB)) /\ State' = 5
end role
```

The TTP now wait both request before send the receipts. We can see that the protocol is safe using the modified fairness formula:

$$E <> (Alice0.Ali0_5 and! TServer2.TSe2_5 and TServer2.d > 5)$$

or its symmetric.

## 5.3 Experimental results

The verification environment is implemented in C++ and integrates the compiler from THLPSL specifications 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 Table 5.1. The experiments have been run on a 3.0GHz Pentium IV with 1Gb of memory running Linux (Slackware 11.0). The column Inst. in the two tables reports the number of protocol sessions allowed in the corresponding test.

Table reports the time, expressed in seconds, spent by the tool for the two versions of the TESLA protocol and their flawed versions, as presented

| Protocol | Inst | CT | VT | St#/Tr#/M |
|---|---|---|---|---|
| TESLA 1 | 1 | .028 | 1.79 | 27/3458/10 |
| TESLA 2 | 1 | 2.37 | 20.90 | 20/4716/10 |
| TESLA 1 Flawed | 1 | 2.74 | 29.73 | 27/3246/10 |
| TESLA 2 Flawed | 1 | 2.37 | 58.73 | 20/4328/10 |
| ZG DY Chn. | 1 | .01 | .33 | 19/48/23 |
| ZG Oper. | 1 | .01 | .4 | 19/48/23 |
| ZG $\text{Oper}_{Fix}$ | 1 | .01 | .1 | 19/48/23 |
| ZG Res. | 1 | .01 | .1 | 21/47/23 |
| WMF | 1-1-3 | .01 | .44 | 16/31/6 |
| $\text{WMF}_{Fix}$ | 1-1-3 | .01 | .03 | 14/18/6 |
| WMF | 2-2-5 | .06 | 129.27 | 28/158/14 |
| $\text{WMF}_{Fix}$ | 2-2-5 | .06 | 111.31 | 24/82/14 |

Table 5.1: Experimental results for Timed Protocols (times are in seconds).

| Protocol | Inst | CT | VT | St#/Tr#/M# | AVISPA OFMC | AVISPA SatMC |
|---|---|---|---|---|---|---|
| NSPK | 3 | .056 | .33 | 17/114/47 | .090 | .853 |
| NSPK | 17 | 18.961 | 55.535 | 93/2955/604 | - | 25.608 |
| $\text{NSPK}_{Fix}$ | 3 | .070 | .039 | 17/98/42 | .090 | .897 |
| $\text{NSPK}_{Fix}$ | 5 | .168 | 21.674 | 29/231/74 | 13.638 | 1.343 |
| ISO1 | 2 | .012 | .015 | 9/12/13 | .059 | .502 |
| ISO1 | 32 | 57.405 | 2.727 | 129/2112/103 | 19.416 | - |
| PBK | 2 | .03 | .036 | 17/96/29 | .102 | .896 |
| PBK | 19 | 9.722 | 344.575 | 117/5547/176 | - | 55.618 |
| $\text{PBK}_{Fix}$ | 2 | .036 | .065 | 17/88/37 | .191 | 1.022 |
| $\text{PBK}_{Fix}$ | 19 | 9.359 | 250.253 | 117/3644/203 | - | 15.659 |

Table 5.2: Experimental results for Untimed Protocols (times are in seconds).

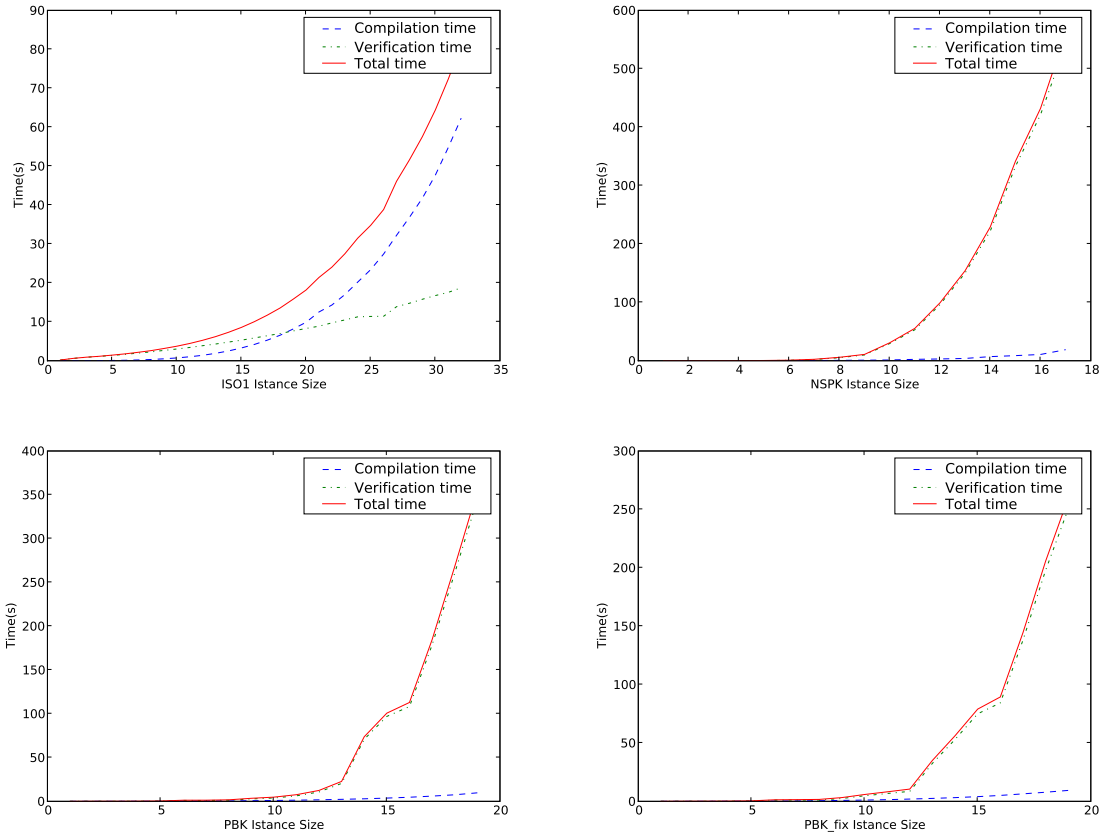



Figure 5.3: Tool performances in seconds against protocol sessions.

in the previous chapter, four version of the Zhou-Gollmann non repudiation protocol, testing the protocol under different assumptions on the communication channel reliability (i.e., resilient, operational and unreliable channels), and the original and fixed versions (as proposed by Lowe [Low97]) of the Wide Mouthed Frog protocol. The column Inst. for the Wide Mouthed Frog protocol reports the number of instances of the participants involved.

In order to allow for a comparison with state-of-the-art verification tools for security protocols, we ran our tool on some untimed protocols. Table 5.1 shows the experimental results both of our tool and of two verification engines included in the AVISPA suite, namely OFMC and SATMC. The untimed protocol analyzed (taken from the AVISPA library of protocols [ABB+05, Vig06]) 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 from two participants. Clearly, the bigger the number of sessions, the higher the number of automata and of ground messages sent/received, leading to a growth in the state space to be analyzed. The property checked for all the protocols is strong authentication. We only report the results for the minimal and maximal instance of the protocols we tried to analyze (the absence of a value for the time spent by a tool, indicates that it did not terminate 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 that our tool scales better as the number of sessions increases. Figure 5.3 also shows how our tool scales against the number of sessions on some untimed protocols. Both compilation and verification times are reported.

Verification and compilation times are usually not correlated. For example in the case of the ISO1 protocol the exchanged messages are heavily structured, so increasing the number of sessions the compiler needs to consider a exponentially growing number of (sub) message (combinations). The verification time for the same protocol scales much better since, while the compiler have to fully generate the automata, the model checker does not need to explore the full state space but only the portion needed 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 of the compiler is quite limited. On the other hand, the verification time increases rapidly, since the model checker have to work on a timed automaton where timing constraints are actually presents, while for ISO1 (and all the untimed protocols) timing is absent.

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.

# Chapter 6

# Decidability of the Protocol Insecurity Problem

In this chapter we focus on the computational complexity of the verification security protocols. In section 6.1 we will introduce the complexity of the problem of verification of security problem we will present the results already present in bibliography and then move to present, in section 6.2 the formalism used to specify timed protocols and an example of a protocol specification. Section 6.3 describes a possible extension of the Dolev-Yao intruder model to the timed setting. In section 6.4 the semantics of the specification language and the notion of attack is defined, together with some crucial properties which ensure that the space of attacks to a protocols is finite and polynomially bounded by the size of the protocol. Finally, in section 6.5 an NP decision procedure for timed protocols insecurity problem is proposed.

## 6.1 Introduction to the Decidability of the Protocol Insecurity Problem

In most cases, the shift from an untimed to a timed model causes a significant growth in complexity (for instance, the reachability problem for Büchi automata is linear, while it is PSPACE for Timed Automata). However, the experimental results with THLPSL do not exhibit a significant growth in

| | With fresh terms | Without fresh terms |
|---|---|---|
| No bound | Undecidable | Undecidable |
| Unbounded n. of sessions + unb. depth messages | Undecidable | DEXPTIME-complete |
| Bounded number of sessions + bounded steps + unbounded depth messages | NP-complete | NP-complete |

Table 6.1: Complexity Results

complexity. The question, thus, arises of what the actual computational complexity of verification of THLPSL specifications is. In literature many results about the decidability and complexity of the protocols insecurity problem have been settled. While the general case (no bound on messages structure and on the number of sessions) has been proved to be undecidable [EG82], for weaker fragments, e.g. by finitely bounding the number of sessions and/or of the messages, there are interesting decidability results [RT03], [DLMS99], [DY83]. In particular, the complexity of the protocol insecurity problem for finite number of sessions has been proved to be NP–Complete by M. Rusinowitch and M. Turuani in [RT03]. Table 6.1 resume the main results.

In the following sections we extend the specification formalism of [RT03] to allow the description of timed dependent security protocols. In particular, we introduce in this framework the temporal features of THLPSL presented in the previous chapters. Moreover, we propose a more powerful threat model, by allowing the intruder to affect some temporal feature of messages. Notice, that to the best of our knowledge, this is the first temporal extension of the Dolev-Yao intruder model in the literature.

The main result is that, under the assumption of a finite number of sessions, adding temporal features to the protocols specifications and assuming a timed intruder model do not change the complexity of the insecurity problem, which remains NP–Complete. This result justifies the experimental results we obtained with THLPSL and UPPAAL.

## 6.2 Modelling Timed Protocols

We extend the protocol model presented in [RT03] with temporal features. Similarly to many other specification languages, the model specifies the actions of the protocols principals as a partially ordered list of steps, which relate what a principal expects to receive and what the principal sends as a reaction. We start by defining the structure of messages and terms involved in principals communication.

### Names and Message primitives

We have a finite set of atomic timed messages, *Atoms*. This set includes also the set *Names* of principal names and the set *Keys* of atomic keys. The elements of the set *Atoms* can be composed by means of three primitives:

- pairing: $\langle \_, \_ \rangle$;
- symmetric, asymmetric encryption: $\{\_\}^s_k$ $\{\_\}^p_k$.

Elements of *Keys* are used only for asymmetric encryption, while any element of *Atoms* can be used as the key for symmetric encryption. Given a $k \in Keys$, with $k^{-1}$ we denote its inverse key. We note also that there is no explicit hashing operator, as it can be simulated using public key encryption.

Given a set of atoms *Atoms*, the set of structured messages over *Atoms*, denoted by $Msg[Atoms]$ is generated by the following grammar:

$$msg ::= Atoms \mid \langle msg, msg \rangle \mid \{msg\}^p_{Keys} \mid \{msg\}^s_{msg}$$

Moreover, given a finite set of (message) variables, *Var*, the set of message terms over *Atoms*, denoted by $Term[Atoms] \supseteq Msg[Atoms]$, is given by the following grammar:

$$trm ::= Atoms \mid Var \mid \langle trm, trm \rangle \mid \{trm\}^p_{trm} \mid \{trm\}^s_{trm}$$

In other words, $Msg[Atoms]$ is the set of ground atoms in $Term[Atoms]$.

Given a set of atoms *Atoms*, a variable substitution over *Atoms*, $\rho$ : $Var \rightarrow Term[Atoms]$, is a function mapping variables to terms. A variable substitution is called a *ground substitution* when it maps variables to messages (ground terms). We denote with $t\rho$ the application of the substitution $\rho$ to the term $t$ and, for any set $E$ of structured terms, $E\rho$ denotes the set $\{t\rho \mid t \in E\}$.

### Timed Signature

Actions of principals can be temporally constrained. To this purpose consider a relative time model where each timed constraint on actions is relative to the execution of some protocol step, a transition/event from now on. The general form of a timing constraint is $[c, C]_{label}$ where $c$ and $C$ are numerical constants (e.g., in $\mathcal{Q}_{\geq 0}$), and *label* is the label identifying a transition/event within the protocol specification. Intuitively, this constraint holds between the time interval bounded by $c + t$ and $C + t$, where $t$ is the time when the transition/event labeled *label* has occurred. The label `start` is a special label which denotes the initialization event of the protocol.

A *timed signature* can be associated to atomic messages (in *Atoms*), which specifies the disclosure and the expiration time of the atom relative to a transition/event. We use the decoration $m[d, e]_{label}$ to denote that message $m$ has disclosure time $d$ and expiration time $e$, relative to the event *label*. $Labels_T$ denotes the set of all timed signatures.

A *time labeling function* $\varphi : Atoms \rightarrow Labels_T$ assigns a timed signature to each element of *Atoms*.

When not otherwise stated, a timed message will have the timed signature $[0, \infty]_{start}$, denoting a message that is always disclosed and never expires. Disclosure and expiration times are properties of a message stated at creation time by some principal and cannot be altered afterwords. Principals can, however, always check for known messages disclosure or expiration.

### Protocol Specification

A protocol specification is a partially ordered set of send/receive actions performed by the principals.

To each principal name $A \in Names$ we associate a finite set of labels and a set of protocol steps, indexed over a partially ordered set $(W_A, <_{W_A})$. Let $\mathcal{I} = \{(A, i) \mid A \in Names,\ i \in W_A\}$ be the set of labels. Formally, a protocol specification $P_\varphi$ over $\mathcal{I}$ is a pair:

$$\langle \{(\iota, T_\iota, R_\iota \xrightarrow[[rd,Rd][sd,Sd]]{[te,Te]_{label}} S_\iota)\}_{\iota \in \mathcal{I}}, \varphi \rangle \text{ , where:}$$

- $(\iota, T_\iota, R_\iota \xrightarrow[[rd,Rd][sd,Sd]]{[te,Te]_{label}} S_\iota)\}_{\iota \in \mathcal{I}}$ is a family of protocol transitions/events indexed over $\mathcal{I}$;
- $\varphi$ is a time labeling function

A protocol step is specified by transitions/events of the form $(\iota, T_\iota, R_\iota \xrightarrow[[rd,Rd][sd,Sd]]{[te,Te]_{label}} S_\iota)$, where:

- $\iota$ is the label;
- $T_\iota$ is a set of Boolean predicates of the form $DSC(x)$, $EXP(x)$, $\neg DSC(x)$, $\neg EXP(x)$, with $x \in Var \cup Atoms$. The conjunction of the predicates in $T_\iota$ specifies the expiration and disclosure constraint of the transition;
- $R_\iota$ is a term in $Term[Atoms] \cup \{Init, \varepsilon\}$ that will be received by the principal. $Init$ and $\varepsilon$ are special terms used to denote the start up of the protocol and the empty message, respectively;
- $S_\iota$ is a term in $Term[Atoms] \cup \{End, \varepsilon\}$ that will be sent by the principal. $End$ and $\varepsilon$ are special terms used to denote the end of the protocol and the empty message, respectively;
- the decoration $\xrightarrow[[rd,Rd][sd,Sd]]{[te,Te]_{label}}$ of the transition specifies the constraints of the protocol step:

– $[te, Te]_{label}$ is the triggering condition. It requires that the transition takes place between time $te$ and $Te$, counting from occurrence of the event $label \in \mathcal{I}$;

– $[rd, Rd]$ is the receipt delay, which can be used to model delays either due to the receive channel or to principal internal activities;

– similarly, $[sd, Sd]$ is the send delay, and can be used to model delays either due to the send channel or to principal internal activities;

**The Specification Of The Wide Mouthed Frog Protocol**

As a specific example consider again the Wide Mouthed Frog presented in section 3.2.

1 $A \rightarrow S : A, \{T_a, B, K_{ab}\}_{K_{as}}$

2 $S \rightarrow B : \{T_s, A, K_{ab}\}_{K_{bs}}$

Below is a possible specification of the protocol, the principals names are $A$ for Alice, $B$ for Bob, $S1$, $S2$ and $S3$ for three instances of the server since the language does not allow the use of cycles or multiple instantiation (both allowed in the $THLPSL$ language). The protocol specification is the following :

$$((A, 1), \emptyset, Init \xrightarrow[[0,0][0,0]]{[0,\infty]_{start}} \langle A, \{T_a, B, K_{ab}\}^s_{K_{as}} \rangle)$$

The first step of Alice is also the starting event of the protocol, as indicated by $Init$. The timing condition of this step specifies an instantaneous transition (both delays are $[0, 0]$) that is always enabled ($[0, \infty]_{start}$). Alice sends the message $\langle A, \{T_a, B, K_{ab}\}^s_{K_{as}} \rangle$ containing the timestamp, the identity of the intended recipient Bob and the key $K_{ab}$.

Let us consider the three instances $S1$, $S2$ and $S3$ of the server. Each instance performs two steps responsible for receiving the timestamp and the key send by some principal and, after testing that the timestamp is not yet expired, for sending the key with a new timestamp to the other principal. The transitions are very similar to the ones of Alice. The transitions of $S1$

are the following:

$$((S1,1),\emptyset,\langle A,\{s_1,B,s_2\}^s_{K_{as}}\rangle \xrightarrow[[0,5][0,0]]{[0,\infty]_{start}} \varepsilon)$$
$$((S1,2),\{\neg EXP(s_1)\},\varepsilon \xrightarrow[[0,0][0,0]]{[0,\infty]_{start}} \langle\{T_{s1},A,s_2\}^s_{K_{bs}}\rangle)$$

The first one receives the message, supposedly from Alice, and does not sent anything. The transition is always enabled and takes at most 5 time units to complete. The second one checks for the non expiration of the timestamp sent by the principals, and sends a message containing the key and a new fresh timestamp. $S3$ performs the same kind of transitions, the difference with $S2$ is that $S2$ expects to be communicating with Bob.

$$((S2,1),\emptyset,\langle B,\{s_3,A,s_4\}^s_{K_{bs}}\rangle \xrightarrow[[0,5][0,0]]{[0,\infty]_{start}} \varepsilon)$$
$$((S2,2),\{\neg EXP(s_3)\},\varepsilon \xrightarrow[[0,0][0,0]]{[0,\infty]_{start}} \langle\{T_{s2},B,s_4\}^s_{K_{as}}\rangle)$$
$$((S3,1),\emptyset,\langle A,\{s_5,B,s_6\}^s_{K_{as}}\rangle \xrightarrow[[0,5][0,0]]{[0,\infty]_{start}} \varepsilon)$$
$$((S3,2),\{\neg EXP(s_5)\},\varepsilon \xrightarrow[[0,0][0,0]]{[0,\infty]_{start}} \langle\{T_{s3},A,s_6\}^s_{K_{bs}}\rangle)$$

The steps performed by Bob are specified as follows:

$$((B,1),\emptyset,\langle\{b_1,A,b_2\}^s_{K_{bs}}\rangle \xrightarrow[[0,5][0,0]]{[0,\infty]_{start}} \varepsilon)$$
$$((B,2),\neg EXP(b_1),EXP(b_2),\varepsilon \xrightarrow[[0,0][0,0]]{[0,0]_{(B,1)}} Secret)$$
$$((B,3),\neg EXP(b_1),\neg EXP(b_2),\varepsilon \xrightarrow[[0,0][0,0]]{[0,0]_{(B,1)}} End)$$

The first step of the principal $B$ is the receive transition. The timestamps $b1$ from the server and $b2$ from principal $A$ are tested for expiration. Notice that the language is mainly suited to express secrecy. In order to model authentication as a secrecy property, we introduce the atom $Secret$, which will be released if authentication fails. Therefore, the second and third transitions check for authentication based on validity of the timestamps. The second transition tests if the timestamp $b1$ (the one originating from the

server) is expired while timestamp $b2$ (originating from $A$) is not. If so, the secret is revealed (sending the special message *Secret*), thereby leading to a violation of the secrecy. Otherwise, transition three correctly ends the protocol by sending the message *End*. Note the timing of the three transitions: the first one is always enabled and takes at most 5 time units to complete (the time signature is $[0, 5]$) modeling a 5 unit delay due to the transmission channel; the other two transitions are both instantaneous and must be taken immediately after the first one.

The timed labeling function $\varphi$ is defined as follow:

$$\begin{aligned}\varphi = \ & \{(T_a, [0,5]_{(A,1)}), (K_{ab}, [0,11]_{(A,1)}), (T_{s1}, [0,5]_{(S1,2)}), \\ & (T_{s2}, [0,5]_{(S2,2)}), (T_{s3}, [0,5]_{(S3,2)})\}\end{aligned}$$

We assume the usual ordering of the label, i.e.: $1 \leq_{W_B} 2 \leq_{W_B} 3$, $1 \leq_{W_{S1}} 2$, $1 \leq_{W_{S2}} 2$, $1 \leq_{W_{S3}} 2$.

## 6.3 Intruder Model

In the Dolev-Yao model [DY83] the intruder has many degrees of freedom, being able to eavesdrop, divert and memorize messages, compose and decompose them, decrypt and encrypt messages with known keys, and generate new messages. In the timed model we propose, the DY intruder is extended with the ability to create, starting from a known timed atom, new timed atoms whose timing is obtained by shifting in the future the disclosure and expiration times of the original atom. This ability is enabled when the intruder sends a message containing a timed atom.

Following the notation used in [RT03], the intruder can be specified by means of a set of rewriting rules on sets of messages. A rule has of the form $L$: $l \to r$, where $l, r$ are sets of messages. A rule $L$: $l \to r$ is applicable to the set of messages $E \in Msg[Atoms]$, if $l \subseteq E$, and the result of the application is the set $E' = (E \setminus l) \cup r$. This is denoted by $E \to_L E'$. The rules modeling a standard D-Y intruder are the following:

Composition Rules:

- $L_c(\langle m_1, m_2\rangle) : m_1, m_2 \to m_1, m_2, \langle m_1, m_2\rangle$;
- $L_c(\{m\}^p_K) : m, K \to m, K, \{m\}^p_K$;
- $L_c(\{m_1\}^s_{m_2}) : m_1, m_2 \to m_1, m_2, \{m_1\}^s_{m_2}$.

Decomposition Rules:

- $L_d(\langle m_1, m_2\rangle) : \langle m_1, m_2\rangle \to m_1, m_2, \langle m_1, m_2\rangle$;
- $L_d(\{m\}^p_K) : \{m\}^p_K, K^{-1} \to m, K^{-1}, \{m\}^p_K$;
- $L_d(\{m_1\}^s_{m_2}) : \{m_1\}^s_{m_2}, m_2 \to m_1, m_2, \{m_1\}^s_{m_2}$.

To model the additional ability of the intruder to generate a fresh timed atom $new(a)$ starting from a known timed atom $a$, we introduce the following rule:

- $L(new(a)) : a \to a, new(a)$.

The atom $new(a)$ is syntactically distinguishable from $a$ and, as we shall see in the semantics of protocol runs, the disclosure and expiration bounds are induced by $\varphi(a)$. The temporal difference between $new(a)$ and $a$ is the transition/event with respect to which the disclosure and expiration times are computed, which will correspond to the first transition/event receiving it during a protocol run. We will write $new^i(a)$ to denote the atom obtained by applying rule $L(new(a))$ to the atom $new^{i-1}(a)$, taking $new^0(a) = a$.

A *derivation* is a sequence of rule applications $E_0 \to_{L_1} E_1 \to_{L_2} \ldots \to_{L_k} E_k$. We write $E \to^* E'$ whenever there is a derivation starting from $E$ and leading to $E'$. We say that a message $m$ is *forged* from $E$, in symbols $m \in forge(E)$, if $E \to^* E'$ and $m \in E'$. Intuitively, $forge(E)$ contains all the messages that can be generated or deduced by the intruder from the set $E$ of known messages.

## 6.4 Protocol Executions and Attacks

In this section we give the semantics of the protocol specification language. An *environment* for a protocol is a pair $\langle E, N\rangle$, where $N$ is a finite set

of timed atoms including *Atoms*, and $E$ a set of messages contained in $Msg[N]$, representing the set of messages sent over the communication channel. A *correct execution order* $\pi$ for a protocol $P_\varphi$ is a one-to-one mapping $\pi : \mathcal{I} \rightarrow \{1, 2, \ldots, |\mathcal{I}|\}$ such that for each $A \in Names$ if $i <_{W_A} j$ then $\pi((A, i)) < \pi((A, j))$. Intuitively, $\pi$ defines the execution order of the steps of the protocol according to the partial ordering $<_{W_A}$. A *time sequence* $\tau : \mathcal{I} \times \{r, s, c\} \rightarrow \mathcal{R}^{\geq 0}$ is a mapping which defines the times at which a particular protocol step takes place. Intuitively, $\tau(i, r)$ corresponds to the time when the reception of a message at step $i$ occurs, $\tau(i, s)$ to the time when the message in step $i$ is sent, and $\tau(i, c)$ to the completion time of step $i$. To ensure monotonicity of the time sequence the following conditions must be satisfied:

- for all $i \in \mathcal{I},\ \tau(i, r) \leq \tau(i, s) \leq \tau(i, c)$;
- for all $i, j \in \mathcal{I},\ \pi(i) < \pi(j)$ implies $\tau(i, c) \leq \tau(j, r)$.

A *protocol run* $\Xi$ for $P_\varphi$ is a tuple $\langle \pi, \rho, \tau, \langle \langle E_0, N_0 \rangle, \langle E_1, N_1 \rangle, \ldots, \langle E_v, N_v \rangle \rangle \rangle$, where $v \leq | \mathcal{I} |$, $\pi$ is a correct execution order, $\rho$ is a ground substitution over $N_v$ and $\tau$ is a time sequence, and $\langle \langle E_0, N_0 \rangle, \ldots, \langle E_v, N_v \rangle \rangle$ is a sequence of environments, satisfying the following properties:

- $Init \in E_0$, $Atoms \subseteq N_0$ and $N_i \subseteq N_{i+1}$;
- for all $1 \leq k \leq v, R_{\pi^{-1}(k)}\rho \in E_{k-1}$ and $S_{\pi^{-1}(k)}\rho \in E_k$;
- $\tau$ satisfies the following temporal constraints, according to the temporal decoration of the transitions/events and to the associated disclosure/expiration guards:

  1. for all $1 \leq k \leq v$, if the $\pi^{-1}(k)$ transition is

  $$(\pi^{-1}(k), T_{\pi^{-1}(k)}, R_{\pi^{-1}(k)} \xrightarrow[[rd,Rd][sd,Sd]]{[te,Te]_{label}} S_{\pi^{-1}(k)})$$

  , then:

(a) $te \leq \tau(\pi^{-1}(k), r) - \tau(label, c) \leq Te$: $te$ and $Te$ serve as delay and timeout for the transition, which is enabled within the time interval $[te, Te]$ starting from the completion of transition/event named $label$;

(b) $rd \leq \tau(\pi^{-1}(k), s) - \tau(\pi^{-1}(k), r) \leq Rd$: $rd$ and $Rd$ serve as bounds for the receiving channel delay;

(c) $sd \leq \tau(\pi^{-1}(k), s) - \tau(\pi^{-1}(k), c) \leq Sd$: $sd$ and $Sd$ serve as bounds for the sending channel delay;

2. for all Boolean predicate $\psi \in T_{\pi^{-1}(k)}$:

   (a) if $\psi$ has the form $DSC(t)$ (respectively $\neg DSC(t)$) with $\varphi_i^{\Xi}(t\rho) = [d, e]_{label'}$, then $(\tau(\pi^{-1}(k), r) - \tau(label', s) \geq d)$ (respectively, $(\tau(\pi^{-1}(k), r) - \tau(label', s) < d)$);

   (b) if $\psi$ has the form $EXP(t)$ (respectively, $\neg EXP(t)$) with $\varphi_i^{\Xi}(t\rho) = [d, e]_{label'}$, then $(\tau(\pi^{-1}(k), r) - \tau(label', s) \geq e)$, (respectively, $(\tau(\pi^{-1}(k), r) - \tau(label', s) < e)$).

   where the function $\varphi_i^{\Xi}$ is the extension of the timed labeling function $\varphi$ to $Msg[N_i]$ induced by $\Xi$ at step $i$ and is defined, for all $m \in Msg[N_i]$ as follows:

$$\varphi_0^{\Xi}(m) = \varphi(m) \quad \text{if } m \in Atoms$$

$$\varphi_i^{\Xi}(m) = \begin{cases} \varphi_{i-1}^{\Xi}(m) & \text{if } m \in N_{i-1} \\ [d, e]_{\pi^{-1}(i)} & \text{if } m = new(b) \in N_i \setminus N_{i-1} \\ & \text{and } \varphi_{i-1}^{\Xi}(b) = [d, e]_{label} \\ [0, \infty]_{start} & \text{otherwise.} \end{cases}$$

Notice that the disclosure/expiration times of messages in conditions 2.(a)-(b) are measured with respect to the sending time associated with the transition/event specified in their time signature, and that to non atomic (composed) messages the trivial time signature ($[0, \infty]_{start}$) is associated.

When a timed sequence $\tau$ satisfies conditions 1.(a)-(c) and 2.(a)-(b) we say that $\tau$ is *compatible* w.r.t. $\pi$ and $\rho$. A *correct protocol execution* is a protocol run where $End \in E_v$;

Given a protocol $P_\varphi$ and a secret message $Secret$ and assuming the intruder has initial knowledge $S_0 \subseteq Msg[Atoms \cup \{Charlie\}]$, an *attack* is a protocol run $\langle \pi, \rho, \tau, \langle \langle E_0, N_0 \rangle, \langle E_1, N_1 \rangle, \ldots, \langle E_k, N_k \rangle \rangle \rangle$ such that:

- $S_{\pi^{-1}(i)}\rho \in E_{i+1}$, for any $1 \leq i < k$;
- $R_{\pi^{-1}(i)}\rho \in forge(S_0, S_{\pi^{-1}(1)}\rho, \ldots, S_{\pi^{-1}(i-1)}\rho)$, for any $1 \leq i \leq k$;
- $N_i \supseteq N_{i-1} \cup \{new(a) \mid new(a) \in Subterm(R_{\pi^{-1}(i)}\rho)\}$;
- $Secret \in forge(S_0, S_{\pi^{-1}(1)}\rho, \ldots, S_{\pi^{-}1(k)}\rho)$.

As an example, consider the specification of the WMF protocol given in section 6.2. A possible protocol execution is $\langle \pi, \rho, \tau, \langle E_0, N_0 \rangle, \ldots, \langle E_5, N_5 \rangle \rangle$, where:

- $\pi = \{((A,1),1), ((S1,1),2), ((S1,2),3), ((B,1),4), ((B,2),5)\}$;
- $\rho : \{(b1, T_{s1}), (b2, K_{ab}), (s1, T_a), (s2, K_{ab})\}$;
- $N_i = Atom$ for all $i = 0, \ldots, 5$ and
  - $E_0 = \{Init, A, B\}$;
  - $E_1 = E_2 = E_0 \cup \{\langle A, \{T_a, B, K_{ab}\}^s_{K_{as}} \rangle\}$;
  - $E_3 = E_4 = E_2 \cup \{\langle \{T_{s1}, A, K_{ab}\}^s_{K_{bs}} \rangle\}$;
  - $E_5 = E_4 \cup \{End\}$;
- the function $\tau$ is reported in the left-hand side of Figure 6.1.

An example of attack to the protocol is $\langle \pi, \rho, \tau, \langle E_0, N_0 \rangle, \ldots, \langle E_9, N_9 \rangle \rangle$, where:

- $\pi = \{((A,1),1), ((S1,1),2), ((S1,2),3), ((S2,1),4), ((S2,2),5), ((S3,1),6), ((S3,2),7), ((B,1),8), ((B,2),9)\}$;

| label | r | s | c |
|---|---|---|---|
| (A,1) | 0 | 1 | 1 |
| (S1,1) | 1 | 1 | 2 |
| (S1,2) | 2 | 3 | 3 |
| (B,1) | 3 | 3 | 4 |
| (B,3) | 4 | 4 | 4 |

| label | r | s | c |
|---|---|---|---|
| (A,1) | 0 | 1 | 1 |
| (S1,1) | 1 | 1 | 2 |
| (S1,2) | 2 | 2 | 2 |
| (S2,1) | 2 | 3 | 4 |
| (S2,2) | 4 | 5 | 5 |
| (S3,1) | 5 | 6 | 6 |
| (S3,2) | 7 | 8 | 8 |
| (B,1) | 8 | 9 | 10 |
| (B,2) | 11 | 11 | 11 |

Figure 6.1: The time sequences of a correct execution (left-hand side) and of an attack (right-hand side).

- $\rho = \{(b1, T_{s3}), (b2, K_{ab}), (s1, T_a), (s2, K_{ab}), (s3, T_{s1}), (s4, K_{ab}), (s5, T_{s2}), (s6, K_{ab})\}$;
- $N_i = Atom$ for all $i = 0, \ldots, 9$ and
  - $E_0 = \{Init, A, B\}$;
  - $E_1 = E_2 = E_0 \cup \{\langle A, \{T_a, B, K_{ab}\}^s_{K_{as}}\rangle\}$;
  - $E_3 = E_4 = E_2 \cup \{\langle \{T_{s1}, A, K_{ab}\}^s_{K_{bs}}\rangle\}$;
  - $E_5 = E_6 = E_4 \cup \{\langle \{T_{s2}, A, K_{ab}\}^s_{K_{bs}}\rangle\}$;
  - $E_7 = E_8 = E_6 \cup \{\langle \{T_{s3}, A, K_{ab}\}^s_{K_{bs}}\rangle\}$;
  - $E_9 = E_8 \cup \{Secret\}$;
- the function $\tau$ is reported in the right-hand side of Figure 6.1.

Following [RT03], we show that for each attack there exists an equivalent attack in a suitable normal form, whose size is polynomially bounded by the size of the protocol specification. This is an essential property to assess the decidability and complexity of the insecurity problem. In the rest of the section we shall show that this property holds also in the timed extension proposed in this thesis. We first introduce the notion of measure of attacks as introduced in [RT03].

Let $Charlie \in S_0$ be an atom known only to the intruder. The size of a term $t$, denoted by $|t|$, is inductively defined as follows:

- $|Charlie|= 0$;
- $|t|= 1$, if $t$ is an atom;
- $|\langle x, y\rangle|=|\{x\}_y|= 1+|x|+|y|$.

The measure of an attack is given with respect to the size of the messages received during the protocol execution. More formally, we denote the size of an attack $\langle \pi, \rho, \tau, \langle\langle E_0, N_0\rangle, \langle E_1, N_1\rangle, \ldots, \langle E_k, N_k\rangle\rangle\rangle$ by the multiset of natural numbers $\{|R_1\rho|, \ldots, |R_k\rho|\}$. An ordering on multisets of naturals can be defined as follows. The ordering relation $\gg$ over multisets is the smallest ordering such that $X \cup \{s\} \gg Y \cup \{t_1, \ldots, t_n\}$ if $X = Y$ and $s > t_i$ for all $i = 1, \ldots, n$. For example, $\{3, 1, 1, 1\} \gg \{2, 2, 2, 1\}$. Given the ordering relation $\gg$, we can define the notion of *normal attack*.

**Definition 6.4.1 (Normal Attack)** *Given a protocol*

$$P = \{(\iota, T_\iota, R'_\iota \xrightarrow[[rd,Rd][sd,Sd]]{[te,Te]_{label}} S'_\iota), \varphi\}_{\iota\in\mathcal{I}}$$

*, an attack* $\langle \pi, \rho, \tau, \langle\langle E_0, N_0\rangle, \langle E_1, N_1\rangle, \ldots, \langle E_k, N_k\rangle\rangle\rangle$ *is* normal *if the multiset* $\{|R_1\rho|, \ldots, |R_k\rho|\}$ *is minimal with respect to* $\gg$*, with* $R_i = R'_{\pi^{-1}}(i)$ *and* $S_i = S'_{\pi^{-1}}(i)$.

In the following, we assume the standard DAG representation of terms (e.g., see [RT03]), and, for a term $t$ we write $|\ t\ |_{DAG}$ to denote the size of the graph representing $t$. Notice that the DAG representation of a term is unique, and its size is linear in the number of distinct subterms. This allows for a compact encoding of a protocol terms by merging possible repeated subterms, preventing the exponential blowup in the representation of protocol runs.

Given an attack of length $k$, let $\mathcal{P} = \{R_i | i = 1, \ldots, k\} \cup \{S_i | i = 0, \ldots, k\}$, with $R_i = R'_{\pi^{-1}}(i)$ and $S_i = S'_{\pi^{-1}}(i)$, denote the set of message terms involved in the run. The same result on normal attacks proved in [RT03] can, then, be proved to hold also in the timed setting.

**Theorem 6.4.2 ([RT03])** *If* $\rho$ *is a ground substitution in a normal attack, then for all* $x \in Var$, $|x\rho|_{DAG} \leq |\mathcal{P}|_{DAG}$.

The theorem above still holds in our setting since the structure of terms introduced in this thesis is the same as the one given by [RT03]. The only notable difference is, indeed, the introduction of the terms of the form $new^j(a)$. Since $new^j(a)$, for any $j$, is still an atom and, therefore, its size is 1, their occurrence as subterms in an attack cannot affect its size. As a consequence, the same proof reported in [RT03] can be trivially adapted to the present case. In [RT03], Theorem 6.4.2 is enough to ensure that the space of normal attacks to a protocol is finite. This does not necessarily hold if generation of fresh atoms by the intruder is allowed (see rule $L(new(a))$ in the intruder model definition). On the other hand, the following lemma can be proved.

**Lemma 6.4.3** *Let $\rho$ be a ground substitution in an attack $\Xi$, such that the atom $new^j(a)$ occurs as a subterm of $R_{\pi^{-1}}(i)\rho$ for some $i$. If $x\rho \neq new^j(a)$ for all $x \in Var$, then $\Xi$ is not a normal attack.*

**Proof** For the sake of space, we only report a sketch of the proof of Lemma 6.4.3. The idea is that we can define a new ground substitution $\rho'$, by substituting every occurrence of the atom $new^j(a)$ in $\rho$ with the atom $Charlie$ (in other words, $x\rho' = x\rho[new^j(a) \leftarrow Charlie]$, for all $x \in Var$). A new attack $\Xi'$ is then obtained from $\Xi$, by substituting $\rho$ with $\rho'$. $\Xi'$ can be proved to be still an attack. The reason is the following. First, every disclosure/expiration guard in the new attack $\Xi'$ is still satisfied. This holds as no transition guard can test the timing of the atom $Charlie$, since $x\rho \neq new^j(a)$, for all $x \in Var$, and, therefore, $new^j(a)$ was never tested for disclosure or expiration in $\Xi$. Second, the substitution of $new^j(a)$ with $Charlie$ (recall that they are both atoms) cannot prevent the derivation in $\Xi'$ of any term occurring in $\Xi$, which does not contain $new^j(a)$ as subterm. The same holds of the derivation of new atoms of the form $new^l(a)$, with $l > j$. Indeed, they can be derived from $a$, and if $a$ is derivable in $\Xi$, then so it is in $\Xi'$. Third, since no variable is assigned to $new^j(a)$ by $\rho$ and the substitution with $Charlie$ is uniform, $\Xi'$ satisfies that for all $1 \leq j \leq k, R_{\pi^{-1}(j)}\rho' \in E_{j-1}$ and $S_{\pi^{-1}(j)}\rho' \in E_j$. The new attack $\Xi'$ is clearly smaller than $\Xi$, since $\mid Charlie \mid < \mid new^j(a) \mid$ and, therefore, $\{\mid R_1\rho \mid, \ldots, \mid R_k\rho \mid\} \gg \{\mid R_1\rho' \mid, \ldots, \mid R_k\rho' \mid\}$. Hence, $\Xi$ cannot be a normal attack.

As a consequence of Lemma 6.4.3, the cardinality of the set of timed atoms occurring in a normal attack of length $v$ cannot be greater than the number of atoms occurring in the protocol specification plus the number of variables. In other words, in any normal attack $\Xi$, $\mid N_i \mid \leq \mid Var \mid + \mid Atom \mid$, for any $i = 1, \ldots, \mid v \mid$. This allows us to state the following theorem.

**Theorem 6.4.4** *For every normal attack $\Xi$, there exists a normal attack $\Xi'$, where no atom of the form $new^j(a)$, with $j >\mid Var \mid$, occurs.*

Since $Var$, $Atom$ and $\{new^j(a) \mid a \in Atom, j \leq \mid Var \mid\}$ are all finite sets, Theorems 6.4.2 and 6.4.4 ensure that when searching for attacks we can limit ourselves to the finite space of normal attacks which contain a finite number of new atoms smaller than the size $\mid \mathcal{P} \mid_{DAG}$ of the protocol.

## 6.5 Complexity of the Timed Insecurity Problem

In this section we show that the problem of checking insecurity of a timed protocol is an NP–Complete problem as in the untimed case. Following the procedure in [RT03] for untimed protocols, to show that the insecurity problem belongs to NP we need to show that we can guess a run of the protocol, namely a correct protocol execution order $\pi$ of length $\mid \mathcal{I} \mid$, a possibly empty set of new timed atoms, and a ground substitution $\rho$, and check, in polynomial time, that it is actually an attack as defined in section 6.4. The following non-deterministic procedure will do the job:

1. Guess a correct execution order $\pi : \mathcal{I} \to \{1, \ldots, v\}$. Let $\mathcal{R}_i = \mathcal{R}'_{\pi^{-1}(i)}$ and $\mathcal{S}_i = \mathcal{S}'_{\pi^{-1}(i)}$ for $i \in \{1, \ldots, v\}$;

2. Guess a monotone sequence of sets $N_0 = Atoms \subseteq N_1 \subseteq \ldots \subseteq N_v \subseteq \{new^j(a) \mid a \in Atom, j \leq \mid Var \mid\}$;

3. Guess a ground substitution $\rho : Var \to Msg[N_v]$, such that for all $x \in V, x\rho$ has DAG-size $\leq n$;

4. For each $i \in \{1, \ldots, v+1\}$ guess an ordered list $l_i$ of $n$ rules whose principal terms have DAG-size $\leq n$;

5. For each $i \in \{1, \ldots, v\}$ check that $l_i$ applied to $\{S_j\rho | j < i\} \cup \{S_0\}$ generates $R_i\rho$;

6. Check that $l_{v+1}$ applied to $\{S_j\rho | j < v+1\} \cup \{S_0\}$ generated $Secret$;

7. Check whether there exists a time sequence $\tau$ which is *compatible* w.r.t. $\pi$ and $\rho$.

If all these checks are successful, then answer YES, and the protocol is insecure. Notice that, with the exception of steps 2. and 7., the other steps are essentially the same as in the untimed case [RT03] and each can be checked in polynomial time. Step 2. can easily be performed in polynomial time too. In the rest of this section we shall show that step 7. can also be checked in polynomial time. This will prove that the problem belongs to NP. NP–Hardness results immediately from the fact the untimed security problem can be easily encoded as a trivial timed insecurity problem.

The existence of a time sequence compatible w.r.t. $\pi$ and $\rho$ can be encoded as a satisfiability problem of a conjunction of Difference Logic constraints.

We shall now show how to build a conjunction of DL constraints which expresses the existence of a time sequence compatible w.r.t. $\pi$ and $\rho$.

For each execution step $i \in \{1, ..., v\}$, we introduce three numerical variables $\mathcal{X}_{\pi^{-1}(i),r}, \mathcal{X}_{\pi^{-1}(i),s}$ and $\mathcal{X}_{\pi^{-1}(i),c}$, which represent the receive time, the send time and the completion time of the transition labeled $\pi^{-1}(i)$, respectively. Moreover, we need one more variable $\mathcal{X}_{Start,c}$ to model the initialization time and assume that $\pi^{-1}(0) = Start$.

We shall split the task of building the desired DL formula into five subformula schemata, each encoding one of the required properties of a compatible time sequence.

First, we need to enforce the monotonicity conditions on a time sequence. Given a label $\iota$ the first monotonicity condition is expressed by the formula:

$$\Phi_\iota^{Mon} = (\mathcal{X}_{\iota,r} - \mathcal{X}_{\iota,s} \leq 0) \wedge (\mathcal{X}_{\iota,s} - \mathcal{X}_{\iota,c} \leq 0)$$

Given two labels $\iota, \iota'$, the second monotonicity condition, stating that the event $\iota$ precedes event $\iota'$, is expressed by the formula:

$$\Phi^{Pr}_{\iota,\iota'} = \mathcal{X}_{\iota,c} - \mathcal{X}_{\iota',r} \leq 0$$

Let us now consider the constraints on $\tau$ induced by the transition/event temporal constraints (conditions 1.(a)–(c) in the definition of protocol run). Given $i$, let $\pi^{-1}(i) = \iota$ and $R_\iota \xrightarrow[[rd,Rd][sd,Sd]]{[te,Te]_{\iota'}} S_\iota$ be the corresponding transition in $\mathcal{P}$. Then we build the formula:

$$\Phi^{Tr}_\iota = \left( \begin{array}{l} te \leq \mathcal{X}_{\iota,r} - \mathcal{X}_{\iota',c} \leq Te \wedge \\ rd \leq \mathcal{X}_{\iota,s} - \mathcal{X}_{\iota,r} \leq Rd \wedge \\ sd \leq \mathcal{X}_{\iota,c} - \mathcal{X}_{\iota,s} \leq Sd \end{array} \right)$$

The DL formula $\Phi^S_\iota$ requires that the time decoration of the transition labeled $\iota$, namely the enabling condition with respect to label $\iota'$ expressing the receive delay and the send delay, are both satisfied.

Finally, for every transition/event $\iota$ we need to encode the disclosure and expiration constraints in $T_\iota$ (conditions 2.(a)–(b) in the definition of protocol run). Let $T^{Dsc^+}_\iota = \{t\rho \mid DSC(t) \in T_\iota\}$ and $T^{Dsc^-}_\iota = \{t\rho \mid \neg DSC(t) \in T_\iota\}$. Then, the following DL formula

$$\begin{array}{rl} \Phi^{Dsc}_\iota = & \bigwedge_{\substack{m \in T^{Dsc^+}_\iota \\ \phi^{\Xi}_i(m) = [d,e]_{\iota'}}} (d \leq \mathcal{X}_{\iota,r} - \mathcal{X}_{\iota',s}) \wedge \\ \bigwedge & _{\substack{m \in T^{Dsc}_\iota \\ \phi^{\Xi}_i(m) = [d,e]_{\iota'}}} (\mathcal{X}_{\iota,r} - \mathcal{X}_{\iota',s} < d) \end{array}$$

requires that all the disclosure predicates for the timed atoms in the guard of the transition labeled $\iota$ are satisfied. Similarly, let $T^{Exp^+}_\iota = \{t\rho \mid EXP(t) \in$

$T_\iota\}$ and $T_\iota^{Exp^-} = \{t\rho \mid \neg EXP(t) \in T_\iota\}$. Then, the following DL formula

$$\Phi_i^{Exp} = \bigwedge_{\substack{m \in T_\iota^{Exp^+} \\ \phi_i^\Xi(m) = [d,e]_{\iota'}}} (e \leq \mathcal{X}_{\iota,r} - \mathcal{X}_{\iota',s}) \wedge \bigwedge_{\substack{m \in T_\iota^{Exp} \\ \phi_i^\Xi(m) = [d,e]_{\iota'}}} (\mathcal{X}_{\iota,r} - \mathcal{X}_{\iota',s} < e)$$

requires that all the expiration predicates for the timed atoms in the guard of the transition labeled $\iota$ are satisfied.

Finally, we build the formula:

$$\Phi^\Xi = \bigwedge_{1 \leq i \leq v} \left( \begin{array}{c} \Phi^S_{\pi^{-1}(i)} \wedge \Phi^{Disc}_{\pi^{-1}(i)} \wedge \Phi^{Exp}_{\pi^{-1}(i)} \wedge \\ \Phi^{Mon}_{\pi^{-1}(i)} \wedge \Phi^{Pr}_{\pi^{-1}(i-1),\pi^{-1}(i)} \end{array} \right)$$

It is immediate to see that satisfiability of $\Phi^\Xi$ ensures the existence of a time sequence $\tau$ which is compatible with the execution of an attack. Notice also that formula $\Phi^\Xi$ is a conjunction of difference constraints and its length is clearly linear in the size of the protocol. Thus, checking the existence of a compatible time sequence can be solved in polynomial time.

# Chapter 7

# Conclusions and Perspectives

In this thesis we contributed to the research area of security protocols analysis on the following two points:

**Verification Framework for Timed Security Protocols**

This work has revisited, and further extended the protocol specification language High Level Specification Language (HLPSL) with Timed High Level Specification Language (THLPSL), a timed extension, which explicitly allows to model temporal features of time sensitive protocols. The semantics of THLPSL was given in terms of Extended Timed Automata (XTA), the input language of well known model checker UPPAAL. This allowed us to develop an prototype verification tool, Timed Protocol Model Checker (TPMC) where a THLPSL specification is first translated into XTA and then is given in input to UPPAAL, together with a property encoding the security goal, for verification. The environment permit the explicit modelling of temporal features of protocols while remaining sufficiently high level to be used by protocol designers and engineers. We presented a number of optimisations that allowed to keep the size of the resulting XTA as small as possible, in particular to reduce the number of clocks and the number of locations of the intruder automaton, increasing the scalability of our approach and allowing the verification on a number of timed and untimed security protocols, showing encouraging performances.

**Theoretical Complexity Results for the Timed Protocol Security Problem**

Driven by the results obtained by the implementation of the TPMC checking framework for timed security protocol we investigated the complexity aspects of their insecurity problem. Using a specification language similar to the one used by Rusinowitch and Turuani [RT03] and an extended Dolev-Yao intruder model, we have shown that the insecurity problem for timed security protocols with a finite number of sessions is NP-Complete. The result shows that we can add time to protocol specification and verify security at no additional computational cost. The verification procedure proposed suggests that state-of- the-art tools developed for untimed protocols, can easily be extended to cope with the temporal dimension. Indeed, the test for compatibility can be performed, e.g. using a decision procedure for DL, independently from the secrecy tests.

## 7.1 Future Development

Needless to say, there is a lot of room for improvement in the design and implementation of the verification framework. At the current stage, the environment still suffers from some limitations both on the specification language side and on the verification engine side. In particular, the kind of security goals which can be tested are limited to secrecy and authentication only, while it would be useful, in the context of timed protocols, to allow for time dependent security properties, such as properties which must be satisfied within some time bounds. Moreover would be interesting researching new kind of intruders/channels models, such as models where its possible for the intruder to alter time for some parts of the protocols. (e.g.: timestamps). As to the verification engine, the current tool builds on top of a model checker for Timed Automata, which, in order to ensure termination, does not allow for parametrised temporal constrains. This limits the possibility to specify and verify protocols where participants can negotiate temporal constants to rule the evolution of the protocol itself. To cope with these limitation, we are

currently investigating further extensions of the environment such as defining suitable extensions of THLPSL to express time sensitive security goals and parametric temporal constraints, and integrating analysis techniques based, e.g., on constraint based decision procedure to overcome the limitations of our current verification engine. Also, in the light of the complexity results presented, a more radical move from the Timed Automata framework to a SAT Modulo Theory/Difference Logic model checking engine could increase the scalability of the tool allowing for partial order reduction like techniques.

In the formal front future work could include the investigation of the impact on complexity of strengthening the abilities of the intruder to alter the time signature of messages. Moreover the current result is limited to secrecy (and authentication) problem. It would be interesting to investigate the complexity of checking timed dependent properties (e.g.: fairness).

# Bibliography


[132] RFC 1321. The MD5 algorithm.

[ABB+05] Alessandro Armando, David Basin , Yohann Boichut, Yannick Chevalier, and al Et. The AVISPA tool for the automated validation of internet security protocols and applications. In Etessami Kousha and Rajamani Sriram, editors, *Computer-Aided Verification , Edinburgh, Scotland, UK, 06/07/05-10/07/05*, pages 1–5. Springer-Verlag, juillet 2005.

[AC04] Alessandro Armando and Luca Compagna. SATMC: A SAT-based model checker for security protocols. In José Júlio Alferes and João Alexandre Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 730–733. Springer, 2004.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, (126):183–235, 1994.

[AFH99] Alur, Fix, and Henzinger. Event-clock automata: A determinizable class of timed automata. *TCS: Theoretical Computer Science*, 211, 1999.

[Arc02] M. Archer. Proving correctness of the basic TESLA multicast stream authentication protocol with TAME*. In *Workshop on Issues in the Theory of Security*, Portland, OR, January 2002.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical report, DIGITAL, Systems Research Center, N 39, February 1989.

[BCCZ] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. pages 193–207.

[BCP06] Massimo Benerecetti, Nicola Cuomo, and Adriano Peron. Timed HLPSL for specification and verification of time sensitive protocols. In *Proceedings of Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'06), Seattle, August 15-16, 2006*, 2006.

[BCP07] Massimo Benerecetti, Nicola Cuomo, and Adriano Peron. TPMC: A model checker for time-sensitive security protocols. In *HPCS*, pages 742–749, 2007.

[BCP09a] Massimo Benerecetti, Nicola Cuomo, and Adriano Peron. An environment for the specification and verification of time dependent security protocols. *International Journal of Computers and Applications, 31 (3)*, 2009.

[BCP09b] Massimo Benerecetti, Nicola Cuomo, and Adriano Peron. TPMC: A model checker for time sensitive security protocols. *JCP Journal of Computers 4 (5) Special Issue on Security and High Performance Computer Systems*, pages 366–377, 2009.

[BCP10] Massimo Benerecetti, Nicola Cuomo, and Adriano Peron. Timed protocols insecurity problem is NP-complete. In Waleed W. Smari and John P. McIntire, editors, *HPCS*, pages 274–282. IEEE, 2010.

[BD00] Berard and Dufourd. Timed automata and additive clock constraints. *IPL: Information Processing Letters*, 75, 2000.

[BDFP04] Bouyer, Dufourd, Fleury, and Petit. Updatable timed automata. *TCS: Theoretical Computer Science*, 321, 2004.

[BFST02] R. Barbuti, N. De Francesco, A. Santone, and L. Tesei. A notion of non-interference for timed automata. *FUNDINF: Fundamenta Informatica*, 51, 2002.

[BHK08] Y. Boichut, P. C. Héam, and O. Kouchnarenko. Approximation-based tree regular model-checking. *Nordic Journal of Computing*, 14(3):216–241, Fall 2008.

[BHR06] Bouyer, Haddad, and Reynier. Timed petri nets and timed automata: On the discriminating power of zeno sequences. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2006.

[BL02] P. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In *Proc. of ESORICS'02, LNCS 2502*, pages 146–161, 2002.

[BMV05] David A. Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec*, 4(3):181–208, 2005.

[BP09] Patricia Bouyer and Antoine Petit. On extensions of timed automata. In Kamal Lodaya, Madhavan Mukund, and R. Ramanujam, editors, *Perspectives in Concurrency Theory*, IARCS-Universities, pages 35–63. Universities Press, January 2009.

[BPDG98] Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inform*, 36(2-3):145–182, 1998.

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jorg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets (4th ACPN'03)*, volume 3098 of *Lecture Notes in Computer Science (LNCS)*, pages 87–124. Springer-Verlag (New York), Eichstatt, Germany, September 2003, selected revised paper 2004.

[CES86] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifica-

tions. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999. To appear.

[CJ] John Clark and Jeremy Jacob. A survey of authentication protocol literature.

[CJM00] Clarke, Jha, and Marrero. Verifying security protocols with brutus. *ACMTSEM: ACM Transactions on Software Engineering and Methodology*, 9, 2000.

[Com00] Douglas E. Comer. *Internetworking with TCP/IP*, volume 1: Principles, Protocols and Architecture. Prentice-Hall, Englewood Cliffs, New Jersey, 4th edition, January 2000.

[CSHM07] 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.

[CV02] Chevaher and Vigneron. Automated unbounded verification of security protocols. In *CAV: International Conference on Computer Aided Verification*, 2002.

[DG04] G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In *Proc. of TACAS 2004, Barcelona, Spain*, pages 342–356, 2004.

[DGP97] Volker Diekert, Paul Gastin, and Antoine Petit. Removing epsilon-transitions in timed automata. In *14th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *lncs*, pages 583–594, Lübeck, Germany, 27 February–March 1 1997. Springer.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[DLMS99] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols (FMSP)*, 1999.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *JACM*, 7(3):201–215, 1960.

[DY83] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[EG82] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *CRYPTO*, page 315, 1982.

[EJ01] Donald E. Eastlake and Paul E. Jones. US secure hash algorithm 1 (SHA1). Internet RFC 3174, September 2001.

[FIP76] *Federal Information Processing Standard 46 – the Data Encryption Standard*, 1976.

[GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata, April 22 1997.

[GLM03] Gorrieri, Locatelli, and Martinelli. A simple language for real-time cryptographic protocol analysis. In *ESOP: 12th European Symposium on Programming*, 2003.

[Hal94] N. M. Haller. The S/KEY one-time password system. In *ISOC*, 1994.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput*, 111(2):193–244, June 1994.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice Hall, 1985.

[JRV00] Florent Jacquemard, Michaël Rusinowitch, and Laurent Vigneron. Compiling and verifying security protocols, 2000.

[Koy90] R. Koymans. Specifying real-time properties with metric temporal logic. *J. Real-Time Systems*, 2, 1990.

[Lam81] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–771, November 1981.

[Lam90] Leslie Lamport. A temporal logic of actions. Technical Report 57, DEC Systems Research Center, Palo Alto, California, April 1990.

[LN03] D. Liu and P. Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2003)*, pages 263–276, San Diego, CA, February 2003. Internet Society.

[Low97] G. Lowe. A family of attacks upon authentication protocols. Technical report, Department of Mathematics and Computer Science, University of Leicester, January 1997.

[Low98] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1-2):53–84, 1998.

[McM93] K. L. McMillan. *Symbolic Model Checking*. PhD thesis, Boston, 1993.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Mil00] Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

[NPP04] M. Napoli, M. Parente, and A. Peron. Specification and verification of protocols with time constraints. *Electr. Notes Theor. Comput. Sci*, 99:205–227, 2004.

[PCTS02] Adrian Perrig, Ran Canetti, J. D. Tygar, and Dawn Song. The TESLA broadcast authentication protocol, August 06 2002.

[Pnu77] A. Pnueli. The temporal logic of programs. In *focs77*, pages 46–57, 1977.

[Ram74] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. Project MAC 120, MIT, 1974.

[RSA78] R. L. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RT94] R. Alur and T.A. Henzinger. A Really Temporal Logic. *JACM*, 41(1):181–204, January 1994.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theor. Comput. Sci*, 1-3(299):451–475, 2003.

[Son99] Dawn Xiaodong Song. Athena: A new efficient automatic checker for security protocol analysis. In *CSFW*, pages 192–202, 1999.

[Sta02] Stallings. The advanced encryption standard. *CRYPTOLOGIA: Cryptologia*, 26, 2002.

[Vig06] L. Viganò. Automated security protocol analysis with the AVISPA tool. *Electr. Notes Theor. Comput. Sci*, 155:61–86, 2006.

[ZG97] Zhou and Gollmann. An efficient non-repudiation protocol. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.