A Simple Language for Real-time Cryptographic Protocol Analysis ⋆

Roberto Gorrieri[1], Enrico Locatelli[1], and Fabio Martinelli[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy.
e_mail: {gorrieri,locatelli}@cs.unibo.it
[2] Istituto di Informatica e Telematica C.N.R., Pisa, Italy.
e_mail: Fabio.Martinelli@iit.cnr.it

Abstract. A real-time process algebra, enhanced with specific constructs for handling cryptographic primitives, is proposed to model cryptographic protocols in a simple way. We show that some security properties, such as authentication and secrecy, can be re-formulated in this timed setting. Moreover, we show that they can be seen as suitable instances of a general information flow-like scheme, called $tGNDC$, parametric w.r.t. the observational semantics of interest. We show that, when considering timed trace semantics, there exists a most powerful hostile environment (or enemy) that can try to compromise the protocol. Moreover, we hint some compositionality results. As a simple running example, we consider the timed version of the Wide Mouthed Frog protocol.

1 Introduction

In the last years there has been an increasing interest in the formal analysis of cryptographic protocols, as they have become the basic building blocks for many distributed services, such as home banking or electronic commerce. These analyses have been very successful in many cases, uncovering subtle inaccuracies in many specifications of cryptographic protocols. However, such analyses are usually restricted to very high abstractions of the real protocols, where concrete information about the timing of events are usually omitted (with the relevant exceptions of [12, 1]).

Our starting point is the work on $CryptoSPA$ [7, 5], which is an extension of SPA [3] (a CCS-like process algebra with actions belonging to two different levels of confidentiality), with some new constructs for handling cryptographic primitives. On such a language a general schema for the definition of security properties, called $GNDC$, has been proposed. It is based on the idea of checking the system against all the possible hostile environments. The general schema has the following form:

$$P \text{ satisfies } S^{\alpha}_{\triangleleft} \text{ iff } \forall X \in Env : P||X \triangleleft \alpha(P)$$

where the general property $S^{\alpha}_{\triangleleft}$ requires that the system P satisfies (via the behavioural preorder $\triangleleft$) a specification $\alpha(P)$ when composed in parallel with any possible hostile

⋆ Work partially supported by MURST Progetto "Metodi Formali per la Sicurezza" (MEFISTO), Microsoft Research Europe and by a CSP/CNR grant for the project "SeTAPS".

environment (or enemy) X. The problem of the universal quantification is overcome when it is possible to show that there exists the "most powerful" enemy; hence, one check against the most powerful enemy is as discriminating as an infinity of different checks against all the possible enemies. This lucky case occurs when the behavioural preorder $\lhd$ is a precongruence, e.g., for trace semantics.

The main goal of this paper is to show that the real-time information flow theory developed for $tSPA$ (a real-time extension of SPA reported in [6]), can be extended to $CryptoSPA$, yielding $timedCryptoSPA$ ($tCryptoSPA$ for short). The main expected results from such an effort are the following:

- A language for describing cryptographic protocols, where information about the concrete timing of events is necessary, e.g., because of the presence of timeouts or timestamps.
- A general scheme, called $tGNDC$, for describing uniformely the many security properties in a real-time setting; we will present two instances of such a general scheme, namely *timed authentication* and *timed secrecy*.
- Some specific results for the security properties based on semantics that are precongruences, such as the existence of a (real-time) most general enemy.

Moreover, we will hint some initial compositionality results, i.e., we will show some conditions under which secure real-time protocols can be safely composed.

As a running example, we will consider the timed version of the Wide Mouther Frog protocol (WMF for short). The choice has been driven by its simplicity; nonetheless, the WMF protocol is expressive enough to discuss some interesting issues, e.g., the timed version of authentication and secrecy.

The paper is organized as follows: in Section 2 we define the $tCryptoSPA$ syntax, operational and behavioural semantics as well as a simple case study. In Section 3 we define the general schema $tGNDC$, hence the notion of hostile environment (or enemy) and we present some general results, such as the existence of a real-time most general enemy. In Section 4 we present some security properties, namely $tNDC$, *timed authentication* and *timed secrecy*. Section 5 reports some initial results about conditions for safe composition of real-time security protocols. Finally in Section 6 we give some concluding remarks and comparison with related literature.

2 The Model

In this section we present the model we will use for the specification of cryptographic protocols and security properties. It is a real-time extension of the *Cryptographic Security Process Algebra* ($CryptoSPA$ for short) proposed in [7, 5], which is in turn an extension of *Security Process Algebra* (SPA for short) proposed in [3] where processes are explicitly given the possibility of manipulating messages. In $CryptoSPA$ it is possible to express qualitative ordering among events, while quantitative timing aspects cannot be expressed. Thus, we extend $CryptoSPA$ with operators that permit to express the elapsing of time, following the approach of [6].

2.1 The Language Syntax

We call the language *Timed Cryptographic Security Process Algebra* ($tCryptoSPA$ for short). Its syntax is based on the following elements:

- A set Ch of channels, partitioned into a set I of input channels (ranged over by c) and a set O of output channels (ranged over by $\overline{c}$, the output corresponding to the input c);
- A set $\mathcal{M}$ of messages;
- A set Var of variables, raged over by x;
- A set $Const$ of constants, ranged over by A.

The set $\mathcal{L}$ of tCryptoSPA terms (or processes) is defined as follows:

$$P ::= \mathbf{0} |\ c(x).P \mid \overline{c}e.P \mid \tau.P \mid tick.P \mid P + P \mid P|P \mid P \backslash L \mid$$

$$A(e_1, \ldots, e_n) \mid [\langle e_1, \ldots, e_r \rangle \vdash_{rule} x]P \mid \iota(P)$$

where $e, e', e_1, \ldots, e_r$ are messages or variables and L is a set of channels. Both the operators $c(x).P$ and $[\langle e_1 \ldots e_r \rangle \vdash_{rule} x]P$ bind the variable x in P.

Besides the standard (value-passing) CCS operators [11], we have an additional prefix action $tick$, used to model time elapsing, a delay operator $\iota(P)$, used to make lazy the initial actions of P, and the operator $[\langle m_1 \ldots m_r \rangle \vdash_{rule} x]P$ introduced in order to model message handling and cryptography. Informally, process $[\langle m_1 \ldots m_r \rangle \vdash_{rule} x]P$ tries to deduce a piece of information z from the tuple of messages $\langle m_1 \ldots m_r \rangle$ through one application of rule $\vdash_{rule}$; if it succeeds, then it behaves like $P[z/x]$, otherwise it is stuck. See the next subsection for a more detailed explanation of derivation rules.

The time model we use is known as the *fictitious clock* approach of, e.g., [8, 13]. A global clock is supposed to be updated whenever all the processes agree on this, by globally synchronizing on action $tick$. All the other actions are assumed to take no time. This is reasonable if we choose a time unit such that the actual time of an action is negligible w.r.t. the time unit. Hence, the computation proceeds in lock-steps: between the two global synchronizations on action $tick$ (that represent the elapsing of one time unit), all the processes proceed asynchronously by performing durationless actions.

Let $Def : Const \longrightarrow \mathcal{L}$ be a set of defining equations of the form $A(x_1, \ldots, x_n) \stackrel{def}{=} P$, where P may contain no free variables except $x_1, \ldots, x_n$, which must be distinct. Constants permit us to define recursive processes, but we have to be a bit careful in using them. A term P is *closed* w.r.t. Def if all the constants occurring in P are defined in Def (and, recursively, for their defining terms). A term P is *guarded* w.r.t. Def if all the constants occurring in P (and, recursively, for their defining terms) occur in a prefix context [11].

The set $Act = \{c(m) \mid c \in I\} \cup \{\overline{c}m \mid \overline{c} \in O\} \cup \{\tau\} \cup \{tick\}$ of actions (τ is the internal, invisible action, $tick$ is the special action used to model time elapsing), ranged over by a (with abuse of notation); we let l range over $Act \backslash \{tick\}$. We call $\mathcal{P}$ the set of all the CryptoSPA *closed* terms (i.e., with no free variables), that are closed and guarded w.r.t. Def. We define $sort(P)$ to be the set of all the channels syntactically occurring in the term P. Moreover, for the sake of readability, we always omit the termination $\mathbf{0}$

at the end of process specifications, e.g., we write a in place of $a.\mathbf{0}$. We often replace constructive rules (encryption and pairing) with the resulting messages, e.g., we use $\overline{c}\{m\}_k$ as a shortcut for $[\langle m, k\rangle \vdash_{enc} x]\overline{c}x$.
We give an informal overview of $tCryptoSPA$ operators:

- $\mathbf{0}$ is a process that does nothing;
- $c(x).P$ represents the process that can get an input m on channel c behaving like P where all the occurrences of x are replaced by m (written $P[m/x]$);
- $\overline{c}m.P$ is the process that can send m on channel c, then behaving like P;
- $\tau.P$ is the process that executes the invisible action τ and then behaves like P;
- $tick.P$ is a process willing to let one time unit pass and then behaving as P;
- $P_1 + P_2$ (*choice*) represents the nondeterministic choice between the two processes P_1 and P_2; time passes when both P_1 and P_2 are able to perform a *tick* action – and in such a case by performing $tick$ a configuration where both the derivatives of the summands can still be chosen is reached – or when only one of the two can perform $tick$ – and in such a case the other summand is discarded; moreover, τ prefixed summands have priority over $tick$ prefixed summands.
- $P_1 \,|\, P_2$ (*parallel*) is the parallel composition of processes that can proceed in an asynchronous way but they must synchronize on complementary actions to make a communication, represented by a τ. Both components must agree on performing a *tick* action, and this can be done even if a communication is possible;
- $P \backslash L$ is the process that cannot send and receive messages on channels in L, for all the other channels it behaves exactly like P;
- $A(e_1, \ldots, e_n)$ behaves like the respective defining term P where all the variables $x_1, \ldots, x_n$ are replaced by the messages $m_1, \ldots, m_n$;
- $[\langle e_1, \ldots, e_r\rangle \vdash_{rule} x]P$ is the process used to model message handling and cryptography. The process $[\langle e_1, \ldots, e_r\rangle \vdash_{rule} x]P$ tries to deduce an information z from the tuple of messages $\langle e_1, \ldots, e_r\rangle$ through the application of rule $\vdash_{rule}$; if it succeeds then it behaves like $P[z/x]$, otherwise it is stuck. The set of rules that can be applied is defined through an inference system (e.g., see Figure 1);
- $\iota(P)$ (*idling*) allows process P to wait indefinitely. At every instant of time, if process P performs an action l, then the whole system proceeds in this state, while dropping the idling operator.

2.2 The Operational Semantics of tCryptoSPA

In order to model message handling and cryptography we use a set of inference rules. Note that $tCryptoSPA$ syntax, its semantics and the results obtained are completely parametric with respect to the inference system used. We present in Figure 1 the same inference system of [7]. This inference system can combine two messages obtaining a pair (rule $\vdash_{pair}$); it can extract one message from a pair (rules $\vdash_{fst}$ and $\vdash_{snd}$); it can encrypt a message m with a key k obtaining $\{m\}_k$ and finally decrypt a message of the form $\{m\}_k$ only if it has the same key k (rules $\vdash_{enc}$ and $\vdash_{dec}$).

In a similar way, the inference system can contain rules for handling the basic arithmetic operations and boolean relations among numbers, so that the value-passing CCS **if-then-else** construct can be obtained via the $\vdash_{rule}$ operator.

$$\frac{m \quad m'}{(m,m')}(\vdash_{pair}) \quad \frac{(m,m')}{m}(\vdash_{fst}) \quad \frac{(m,m')}{m'}(\vdash_{snd})$$

$$\frac{m \quad k}{\{m\}_k}(\vdash_{enc}) \quad \frac{\{m\}_k \quad k}{m}(\vdash_{dec})$$

Fig. 1. An example inference system for shared key cryptography.

Example 1. Natural numbers may be encoded by assuming a single value 0 and a function $S(y)$, with the following rule:

$$\frac{x}{S(x)} inc$$

So we will often use $n + 1$ to indicate the application of rule inc with n as argument. Similarly, we can define summations and other operations on natural numbers. ■

Example 2. We do not explictly define equality check among messages in the syntax. However, this can be implemented through the usage of the inference construct. E.g., consider rule $equal \stackrel{def}{=} \frac{x \quad x}{Equal(x,x)}$. Then $[m = m']A$ (with the expected semantics) may be equivalently expressed as $[m \quad m' \vdash_{equal} y]A$ where y does not occur in A. Similarly, we can define inequalities, e.g., $\leq$, among natural numbers. ■

We consider a function $\mathcal{D}$, from finite sets of messages to sets of messages, such that $\mathcal{D}(\phi)$ is the set of messages that can be deduced from ϕ. We assume that $\mathcal{D}$ is decidable and that cryptography is completely reliable, i.e., that a crypted message can be deciphered only by knowing the suitable decryption key.

The operational semantics of a $tCryptoSPA$ term is described by means of the *labelled transition system* (*lts*, for short) $\langle \mathcal{P}, Act, \{\stackrel{a}{\longrightarrow}\}_{a \in Act} \rangle$, where $\{\stackrel{a}{\longrightarrow}\}_{a \in Act}$ is the least relation between $tCryptoSPA$ processes induced by the axioms and inference rules of Figure 2. Such a relation is well-defined even if negative premises occur in a rule for the idling operator and in one rule for $+$, because the relation is *strictly stratifiable* [9].
Note that $tCryptoSPA$ is $tick$*-deterministic* i.e., the time elapsing never moves a process to two different states. The proof of the following proposition can be easily given by inspecting the operational rules. In particular, the first two rules of the idling operator and the rules for nondeterministic choice are the key rules enforcing time determinacy.

Proposition 1. *For every tCryptoSPA process P we have:*
If $P \stackrel{tick}{\longrightarrow} P'$ *and* $P \stackrel{tick}{\longrightarrow} P''$ *then* $P' = P''$.

Example 3. In $tCryptoSPA$ there are processes, such as $\mathbf{0}$, that do not allow time to proceed; hence, as rule $|_3$ for parallel composition forces a global synchronisation on $tick$ actions, the effect of composing a process P with $\mathbf{0}$ is to prevent P from letting time pass. In other words, $\mathbf{0}$ acts as a time annihilator for its parallel context. On the contrary, $\iota(\mathbf{0})$ is process that, even if functionally terminated, let time to proceed indefinitely. Hence, $\iota(\mathbf{0})$ acts as a neutral element for parallel composition. ■

$$(input)\frac{m \in \mathcal{M}}{c(x).P \xrightarrow{c(m)} P[m/x]} \qquad (output)\frac{}{\overline{c}m.P \xrightarrow{\overline{c}m} P}$$

$$(internal)\frac{}{\tau.P \xrightarrow{\tau} P} \qquad (tick)\frac{}{tick.P \xrightarrow{tick} P}$$

$$(|)_1\frac{P_1 \xrightarrow{l} P_1'}{P_1 \,|\, P_2 \xrightarrow{l} P_1' \,|\, P_2} \qquad (|)_2\frac{P_1 \xrightarrow{c(x)} P_1' \quad P_2 \xrightarrow{\overline{c}m} P_2'}{P_1 \,|\, P_2 \xrightarrow{\tau} P_1' \,|\, P_2'}$$

$$(|)_3\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \xrightarrow{tick} P_2'}{P_1 \,|\, P_2 \xrightarrow{tick} P_1' \,|\, P_2'} \qquad (\backslash L)\frac{P \xrightarrow{c(m)} P' \quad c \notin L}{P\backslash L \xrightarrow{c(m)} P'\backslash L}$$

$$(+_1)\frac{P_1 \xrightarrow{l} P_1'}{P_1 + P_2 \xrightarrow{l} P_1'} \qquad (+_2)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \xrightarrow{tick} P_2'}{P_1 + P_2 \xrightarrow{tick} P_1' + P_2'}$$

$$(+_3)\frac{P_1 \xrightarrow{tick} P_1' \quad P_2 \not\xrightarrow{tick} \quad P_2 \not\xrightarrow{\tau}}{P_1 + P_2 \xrightarrow{tick} P_1'}$$

$$(Def)\frac{P[m_1/x_1, \ldots, m_n/x_n] \xrightarrow{a} P' \quad A(x_1, \ldots, x_n) \stackrel{def}{=} P}{A(m_1, \ldots, m_n) \xrightarrow{a} P'}$$

$$(\mathcal{D})\frac{\langle m_1, \ldots, m_r \rangle \vdash_{rule} m \quad P[m/x] \xrightarrow{a} P'}{[\langle m_1, \ldots, m_r \rangle \vdash_{rule} x]P \xrightarrow{a} P'}$$

$$(\mathcal{I}_1)\frac{P \not\xrightarrow{tick}}{\iota(P) \xrightarrow{tick} \iota(P)} \qquad (\mathcal{I}_2)\frac{P \xrightarrow{tick} P'}{\iota(P) \xrightarrow{tick} \iota(P')} \qquad (\mathcal{I}_3)\frac{P \xrightarrow{l} P'}{\iota(P) \xrightarrow{l} P'}$$

Fig. 2. Structured Operational Semantics for tCryptoSPA (symmetric rules for $+_1, +_3, |_1, |_2$ and $\backslash L$ are omitted)

Example 4. Consider a process $P = \iota(a) \,|\, \iota(\overline{a})$ that can perform any sequence (possibly empty) of $tick$ actions followed by a τ. It is worth-observing that, contrary to $tSPA$ [6], we do not assume maximal communication progress, i.e., τ's do not have priority over $tick$ actions or, equivalently, a process cannot idle if it can perform a τ. Hence in $tSPA$ process P can perform only the sequence τ. ■

Example 5. We can easily model timeout constructs in $tCryptoSPA$.
Assume $n_1 \leq n_2$ and define a process

$$Time\_out(n_1, n_2, A, B) = tick^{n_1}.\iota(A) + tick^{n_2}.\tau.B$$

By looking at the rules for choice and idling, we see that $Time\_out(n_1, n_2, A, B)$ first performs a sequence of n_1 $tick$ actions; then, the system may perform other $n_2 - n_1$ $tick$ actions, unless A resolves the choice by performing an action; instead if A does nothing, after n_2 units of time, through the execution of action τ, the process is forced to act as B. Note that rule $+_3$ is responsible for preventing the selection of process A at timeout expiration. This semantics for the $+$ operator is different from the one in $tSPA$ (a $tick$ action can be performed *only if* both summands can do so) and is motivated by the

$$
\begin{aligned}
Clock(n) &= tick.Clock(n+1) + \overline{time}\, n.Clock(n) \\
A(K) &= time(T_a).tick.\overline{c}_{AS}\{B, T_a, K\}_{K_{as}}.\iota(\mathbf{0}) \\
B &= \iota(c_{BS}(y)[\langle y, K_{bs}\rangle \vdash_{dec} z]time(T_s).tick.[\langle z\rangle \vdash_{snd} t] \\
&\quad [t+1 \leq T_s][\langle z\rangle \vdash_{trd} w]\tau) \\
S &= tick.c_{AS}(u).[\langle u, K_{as}\rangle \vdash_{dec} z]time(T_s).tick.[\langle z\rangle \vdash_{snd} t][t+1 \leq T_s] \\
&\quad [\langle z\rangle \vdash_{trd} x]\overline{c}_{BS}\{A, T_s, x\}_{K_{bs}}.tick \\
P(K_{ab}) &= (Clock(0) \,|\, A(K_{ab}) \,|\, B \,|\, S)\backslash\{c_{AS}, c_{BS}, time\}
\end{aligned}
$$

Fig. 3. $tCryptoSPA$ specification for the WMF protocol

need of a more flexible way of programming the choice between different components as, e.g., in the $Clock(n)$ process of our running example, when the maximal progress assumption for parallel composition is dropped. ■

2.3 The Wide-Mouthed Frog Protocol

As a simple example, we recall the Wide-Mouthed Frog Protocol (WMF, for short). The aim of the protocol is to establish a session key from an initiator A to a responder B by means of a server S and using shared key cryptography. The protocol, as reported in [2], is as follows:

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

where T_a and T_s are timestamps, K_{as} (K_{bs}) is the key shared between A and S (B and S), and K_{ab} is the new session key. Both the server S and the responder B check the timestamp. If it is too old, the message (hence, the session key) is ignored.

In order to model this protocol in $tCryptoSPA$, we use a process $Clock$, counting the time elapsing, which other processes can ask for the current time value. Moreover, we use $[\langle m, m', m''\rangle \vdash_{trd} x]$ as a shortcut to extract m'' and also $[t \leq t']P$, assuming to have an inference system that handles the boolean relations on arithmetic expressions. With c_{XY} we denote the channel used by X to communicate with Y and *vice-versa*.

The lts of $P(K_{ab})$ is reported below, where, for the sake of clarity, each τ labelled transition is decorated also with the channel on which the synchronization took place and, possibly, with the modified value of the timestamps:

$$
P(K_{ab}) \stackrel{\tau, time, T_a=0}{\longrightarrow} . \stackrel{tick}{\longrightarrow} . \stackrel{\tau, c_{AS}}{\longrightarrow} . \stackrel{\tau, time, T_s=1}{\longrightarrow} .
$$

$$
\stackrel{tick}{\longrightarrow} . \stackrel{\tau, c_{BS}}{\longrightarrow} . \stackrel{\tau, time, T_s=2}{\longrightarrow} . \stackrel{tick}{\longrightarrow} . \stackrel{\tau}{\longrightarrow}
$$

This protocol takes two time units to perform the distribution of the key. At the third time unit, A and B are ready to use the session key.

3 A General Schema for the Definition of Timed Security Properties

In this section we propose a general schema for the definition of timed security properties. We call it *Timed Generalized NDC* ($tGNDC$ for short), since it is a real-time generalization of *Generalized NDC* ($GNDC$ for short) [7], which is in turn a generalization of *Non Deducibility on Compositions* (NDC for short) [3]. The main idea is the following: a system P is $tGNDC^{\alpha}_{\lhd}$ iff for every hostile environment (or enemy) X the composition of the system P with X satisfies the timed specification $\alpha(P)$. Essentially $tGNDC^{\alpha}_{\lhd}$ guarantees that the timed property α is satisfied, with respect to the $\lhd$ timed behavioural relation, even when the system is composed with any possible enemy.

This section is organised as follows. We first define timed trace semantics as the behavioural semantics of interest. Then, we discuss the issue of hostile environments, showing that it is necessary to restrict their initial knowledge. Finally, we present the $tGNDC$ schema and some general results on it, some of which are independent of the chosen behavioural notion.

3.1 Behavioural semantics

Here we define the semantic preorder and equivalence we will use to formalize security properties, *timed trace* preorder and equivalence, the timed version of the classic untimed semantics.
The expression $P \stackrel{a}{\Rightarrow} P'$ is a shorthand for $P(\stackrel{\tau}{\longrightarrow})^* P_1 \stackrel{a}{\longrightarrow} P_2(\stackrel{\tau}{\longrightarrow})^* P'$ where $(\stackrel{\tau}{\longrightarrow})^*$ denotes a (possibly empty) sequence of transitions labelled τ. Let $\gamma = a_1, \ldots, a_n \in (Act\backslash\{\tau\})^*$ be a sequence of actions; then $P \stackrel{\gamma}{\Rightarrow} P'$ iff there exist $P_1, \ldots, P_{n-1} \in \mathcal{P}$ such that $P \stackrel{a_1}{\Rightarrow} P_1 \stackrel{a_2}{\Rightarrow}, \ldots, P_{n-1} \stackrel{a_n}{\Rightarrow} P'$.

Definition 1. *For any $P \in \mathcal{P}$ the set $T(P)$ of timed traces associated with P is defined as follows $T(P) = \{\gamma \in (Act\backslash\{\tau\})^* \mid \exists P'.P \stackrel{\gamma}{\Rightarrow} P'\}$. The timed trace preorder, denoted by $\leq_{ttrace}$, is defined as follows: $P \leq_{ttrace} Q$ iff $T(P) \subseteq T(Q)$. P and Q are* timed trace equivalent*, denoted by $P =_{ttrace} Q$, if $T(P) = T(Q)$.*

As an example, it is easy to see that $T(P(K_{ab})) = \{tick, tick\ tick, tick\ tick\ tick\}$.

3.2 Hostile Environments

Here we characterize the notion of admissible hostile environments similarly to what done in [7] for the untimed setting. Such a characterization is necessary to analyze protocols where some information is assumed to be secret, as in cryptographic protocols. A hostile environment, or *enemy*, is a process which tries to attack a protocol by stealing and faking information which is transmitted on *public* channels in C. Such an agent is modeled as a generic process X which can communicate only through channels in C, imposing some constraints on the initial data that are known by the enemy and requiring that such a protocol is *weakly time alive*, i.e., the agent may always perform $tick$ eventually. Otherwise X could prevent time from elapsing when composed in parallel with some system, since in a compound system time can pass iff all components let it pass. So the enemy could block the time flow and we want to avoid this unrealistic case. Let $Der(P)$ be the set of all derivates of P, i.e., all the P's reachable from P.

Definition 2. *A process P is directly weakly time alive* iff $P \stackrel{tick}{\Longrightarrow} P'$. *$P$ is weakly time alive* iff *for all $P' \in Der(P)$, we have P' is directly weakly time alive.*

Now, let $ID(P)$ be the set of messages that appear in P and $\phi \subseteq \mathcal{M}$ be the initial knowledge we would like to give to the enemies, i.e., the public information such as the names of the entities and the public keys, plus some possible private data of the enemies (e.g., their private key or nonces). For some enemy X, we want that all the messages in $ID(X)$ are deducible from ϕ. We thus define the set $t\mathcal{E}_C^{\phi_I}$ of timed hostile processes as:

$$t\mathcal{E}_C^{\phi} = \{X \in \mathcal{P} \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \mathcal{D}(\phi) \text{ and } X \text{ is weakly time alive}\}$$

3.3 The $tGNDC$ Schema

In this section we formally define the $tGNDC_{\lhd}^{\alpha}$ family of properties. We will use $A||_C B$ as a shortcut for $(A \,|\, B)\backslash C$. The proposed family of security properties is as follows.

Definition 3. *P is $tGNDC_{\lhd}^{\alpha}$ iff*

$$\forall X \in t\mathcal{E}_C^{\phi_I} \; : \; (P||_C X) \lhd \alpha(P)$$

where $\lhd \in \mathcal{P} \times \mathcal{P}$ is a timed behavioral preorder and $\alpha \; : \; \mathcal{P} \mapsto \mathcal{P}$ is a function between processes defining the property specification for P as the process $\alpha(P)$.

We propose a sufficient criterion for a static characterization (i.e., not involving the universal quantification $\forall$) of $tGNDC_{\lhd}^{\alpha}$ properties. We will say that $\lhd$ is a *precongruence* w.r.t. $||_C$ if it is a preorder and for every $P, Q, Q' \in \mathcal{P}$ if $Q \lhd Q'$ then $P||_C Q \lhd P||_C Q'$. Thus it is easy to prove the following:

Proposition 2. *If $\lhd$ is a precongruence w.r.t. $||_C$ and if there exist a process $Top \in t\mathcal{E}_C^{\phi_I}$ such that for every process $X \in t\mathcal{E}_C^{\phi_I}$ we have $X \lhd Top$, then $\forall \alpha$:*

$$P \in tGNDC_{\lhd}^{\alpha} \quad \text{iff} \quad (P||_C Top) \lhd \alpha(P)$$

Proof. ($\Leftarrow$)If $\lhd$ is a precongruence w.r.t. $||_C$, then $X \lhd X'$ implies $(P||_C X) \lhd (P||_C X')$. Thus if for every process $X \in t\mathcal{E}_C^{\phi_I}$ we have $X \lhd Top$ then we will also have $(P||_C X) \lhd (P||_C Top)$ for every process $X \in t\mathcal{E}_C^{\phi_I}$ and so, as by hypothesis $(P||_C Top) \lhd \alpha(P)$, we obtain $(P||_C X) \lhd (P \,|\, Top) \lhd \alpha(P), \; \forall X \in t\mathcal{E}_C^{\phi_I}$, i.e., P is $tGNDC_{\lhd}^{\alpha}$.
($\Rightarrow$)By definition of $tGNDC_{\lhd}^{\alpha}$, since $Top \in t\mathcal{E}_C^{\phi_I}$.

In particular if the hypotheses of the proposition above hold it is sufficient to check that $\alpha(P)$ is satisfied when P is composed with the most general hostile environment Top. This permits to make only one single check, in order to prove that a property holds whatever attacker we choose. We also have the following corollary for the congruence induced by $\lhd$:

Corollary 1. *Let $\lhd$ be a precongruence w.r.t. $||_C$ and let $\equiv\ =\ \lhd \cap \lhd^{-1}$. If there exist two processes $Bot, Top \in t\mathcal{E}_C^{\phi_I}$ such that for every process $X \in t\mathcal{E}_C^{\phi_I}$ we have $Bot \lhd X \lhd Top$ then*

$$P \in tGNDC_{\equiv}^{\alpha} \quad \text{iff} \quad (P||_C Bot) \equiv (P||_C Top) \equiv \alpha(P)$$

Given these very general results, we show that they are instanciable in the model we presented so far. Indeed, this is the case, at least for the trace preorder $\leq_{ttrace}$, which is a precongruence.

Proposition 3. *Timed trace preorder is a precongruence w.r.t. $||_C$.*

Proof. Given $P, Q, Q' \in \mathcal{P}$ such that $Q \leq_{ttrace} Q'$, we have to prove $P||_C Q \leq_{ttrace} P||_C Q'$. This can be done by induction on the length of the timed traces generated by $P||_C Q$.

Note that in the $tSPA$ model, timed trace preorder is not a precongruence, since the semantic rules enforce the so called *maximal communication progress*, i.e., when a communication is possible it must start immediately, and it is not possible to perform a $tick$ [6].

Now we single out the minimal element Bot and the maximum element Top in $t\mathcal{E}_C^{\phi_I}$ w.r.t. $\leq_{ttrace}$. As for Bot it is clear that the minimum set of traces is generated by the weakly time alive process that does nothing, that is generated by process $\iota(\mathbf{0})$. As a matter of fact, $(P \mid \iota(\mathbf{0})) =_{ttrace} P$ for timed trace equivalence and most other equivalences. We thus define the Top element using a family of processes $Top_{ttrace}^{C,\phi}$ each representing an instance of the enemy with knowledge ϕ:

$$Top_{ttrace}^{C,\phi} = \sum_{\substack{c \in C \\ m \in \mathcal{M}}} \iota(c(m).Top_{ttrace}^{C,\phi\cup\{m\}}) + \sum_{\substack{c \in C \\ m \in \mathcal{D}(\phi) \cap \mathcal{M}}} \iota(\overline{c}m.Top_{ttrace}^{C,\phi})$$

The initial element of the family is Top_{ttrace}^{C,ϕ_I} as ϕ_I is the initial knowledge of the enemy. This may accept any input message to be bound to the variable x which is then added to the knowledge set that becomes $\phi_I \cup \{x\}$, and may output only messages that can pass on the channel c and that are deducible from the current knowledge set ϕ via the deduction function $\mathcal{D}$. Furthermore it can let time pass. Note that τ summands are not considered, as inessential for trace preorder. As done in [5] we prove the following:

Proposition 4. *If $X \in t\mathcal{E}_C^{\phi}$ then $X \leq_{ttrace} Top_{ttrace}^{C,\phi}$.*

Proof. By induction on the length of the timed traces generated by X.

4 Some Timed Security Properties

In this section we show how to redefine three timed security properties as suitable instances of the $tGNDC_{\lhd}^{\alpha}$ schema, by suitably instancing function α. As for the behavioural semantics $\lhd$, we will always consider the timed trace semantics. The three properties we consider are:

– The timed version of Non Deducibility on Compositions [3], which has been proposed to study information flow security; we will show that $tNDC$ is the strongest property in the $tGNDC^{\alpha}_{\triangleleft}$ family, under some mild assumptions.
– A timed notion of authentication, called *timed agreement* [12], as the natural transposition in a timed setting of the agreement property [10], according to which agreement must be reached within a certain deadline, otherwise authentication does not hold.
– a timed notion of secrecy, we call *timed secrecy*, according to which a message is secret only within a time interval and after the deadline it can become a public piece of information.

4.1 Timed Non Deducibility on Compositions

We start with $tNDC$ since $tGNDC^{\alpha}_{\triangleleft}$ is a generalization of such a property. Its underlying idea is that the system behavior must be invariant w.r.t. the composition with every hostile environment. Indeed, there is no possibility of establishing a communication (i.e. sending information). In the $CryptoSPA$ untimed setting the NDC [1] idea can thus be defined as follows:

Definition 4. *$P \in NDC$ if and only if $\forall X \in \mathcal{E}_C^{\phi_I}$, we have $(P||_C X) =_{trace} P\backslash C$.*

where $=_{trace}$ is trace preorder and the only difference with the definition given in SPA is that the knowledge of process X is bounded by ϕ_I. Now we present *timed NDC* ($tNDC$, for short) ([6]) which is the natural extension of NDC to a timed setting.

Definition 5. *$P \in tNDC$ if and only if $\forall X \in t\mathcal{E}_C^{\phi_I}$ we have $(P||_C X) =_{ttrace} P\backslash C$.*

where the difference is that we use the timed hostile environment and timed trace preorder. Note that $tNDC$ corresponds to $tGNDC^{P\backslash C}_{=_{ttrace}}$. It is also possible to apply Corollary 1 obtaining the following static characterization.

Proposition 5. *$P \in tNDC$ if and only if $(P||_C Top^{C,\phi_I}_{ttrace}) =_{ttrace} P\backslash C$.*

Now we suggest that $tNDC$ is the most restrictive $\alpha(P)$ hence inducing the strongest property for timed trace semantics. The most restrictive $\alpha(P)$ should return an *encapsulation* of protocol P, i.e., a version of P which is completely isolated from the environment, corresponding to the execution of P in a perfectly secure network where only the honest parties are present. In our process algebra setting, this corresponds to the *restriction* of all public channels in C along which protocol messages are sent.

Note that for every process P we have $(P \,|\, \iota(\mathbf{0})) \setminus C =_{ttrace} P \setminus C$. This means that P restricted on C is equivalent to the protocol in composition with the enemy that does nothing. Note also that, by definition, $\iota(\mathbf{0}) \in t\mathcal{E}_C^{\phi}$ for every ϕ. So it is very natural to consider α functions and processes P such that:

$$P \setminus C \leq_{ttrace} \alpha(P)$$

[1] As for $tGNDC^{\alpha}_{\triangleleft}$, also NDC and $tNDC$ are implicitly parametric w.r.t. the set C of public channels and the set ϕ_I of initial knowledge. We usually omit these parameters when clear from the context.

This simply means that the protocol P is correct (as it satisfies its specification $\alpha(P)$) at least when it is not under attack. This condition can be somehow seen as a reasonable criterion for any *good* protocol: it must be correct at least when it is not under attack! Under this mild assumption, it is clear that $P \in tNDC$ implies $P \in tGNDC^{\alpha}_{\leq_{ttrace}}$.

Another way to avoid universal quantification over all the admissible enemies is to show the equivalence between $tNDC$ and *Timed Strong Nondeterministic Non-Interference* ($tSNNI$, for short); such equivalence result holds in the untimed case [3], but that does not hold for $tSPA$ [6] because of the maximal communication assumption of that language.

A $CryptoSPA$ process is $SNNI^{\phi}_{C}$ if $P \backslash C$, where all actions in C are forbidden, behaves like the system P where all the actions in C are *hidden* (i.e., transformed into internal τ actions). To express this second system we need to introduce first the *hiding* operator $P/^{\phi}C$:

$$\frac{P \xrightarrow{a} P'}{P/^{\phi}C \xrightarrow{a} P'/^{\phi}C}(a \notin C \cup \overline{C})$$

$$\frac{P \xrightarrow{\overline{c}m} P' \quad c \in C \cup \overline{C}}{P/^{\phi}C \xrightarrow{\tau} P'/^{\phi \cup \{m\}}C}$$

$$\frac{P \xrightarrow{c(m)} P' \quad c \in C \cup \overline{C} \quad m \in \mathcal{D}(\phi)}{P/^{\phi}C \xrightarrow{\tau} P'/^{\phi}C}$$

Now we are ready to define the property *timed* $SNNI^{\phi}_{C}$ as follows.

Definition 6. *A process is $tSNNI^{\phi}_{C}$ if $P \backslash C =_{ttrace} P/^{\phi}C$.*

It is rather intuitive that $P/^{\phi}C$ can be seen as $P||_{C}Top$, where Top is the top element of the trace preorder for $CryptoSPA$. Hence, such a static characterization can be seen as a corollary of the existence of a top element in the trace preorder (together with the fact that trace preorder is a precongruence).

Proposition 6. *For every $P \in \mathcal{P}$ we have that $(P||_{C}Top^{C,\phi}_{ttrace}) =_{ttrace} P/^{\phi}C$*

Proof. We have to prove $(P||_{C}Top^{C,\phi}_{ttrace}) \leq_{ttrace} P/^{\phi}C$ and the symmetric relation $(P||_{C}Top^{C,\phi}_{ttrace}) \leq_{ttrace} P||_{C}Top^{C,\phi}_{ttrace})$. These can be done by induction on the length of the traces generated by the left-hand-side processes.

Proposition 7. $P \in tNDC^{\phi}_{C}$ iff $P \in tSNNI^{\phi}_{C}$.

Proof. By Proposition 5 and Proposition 6.

4.2 Timed Agreement

We now present the *Timed Agreement* Property [10, 12]:

"A protocol guarantees to an initiator A *Timed Agreement* with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, in the last $n\ ticks$ (where n is a prefixed timeout value), and B was acting as a responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds, and each such run of A corresponds to a unique run of B."

As done in [7] for the non real-time version of *Agreement*, what we do is to have for each party an action representing the running of a protocol and another one representing the completation of it. We consider an action $commit\_res(B, A, d)$ representing a correct termination of B as a responder, convinced to communicate with A that agrees on data d. On the other hand we have an action $running\_ini(A, B, d)$ that represents the fact that A is running the protocol as initiator, apparently with B, with data d. If we specify these two actions in the protocol, the *Timed Agreement* property requires that when B executes $commit\_res(B, A, d)$ then A has previously executed $running\_ini(A, B, d)$ and at most $n\ tick$ actions, where n is the prefixed timeout value, occurred between these actions. We assume that the actions representing the running and the commit are correctly specified in the protocol. We can see them as output actions over two particular channels $running\_ini$ and $commit\_res$. We assume that d can assume values in a set D. Let $NotObs(P) = sort(P)\backslash(C \cup \{running\_ini, commit\_res\})$ be the set of channels in P that are not public and are different from $running\_ini$ and $commit\_res$, i.e., that will not be observed. Function $\alpha^{t(n)}_{tAgree}$ can be thus defined as follows:

$$P'(x,y) = \sum_{\substack{d \in D \\ 0 \leq\ i\ \leq n}} \overline{running\_ini}(x,y,d).tick_1.\ldots.tick_i.\overline{commit\_res}(y,x,d)$$

$$P'' = \sum_{c \in NotObs(P)} \iota(c(x).P'') + \sum_{\substack{c \in NotObs(P) \\ m \in Msg(c)}} \iota(\overline{c}m.P'')$$

$$\alpha^{t(n)}_{tAgree}(P) = P'' \mid P'(A,B)$$

Note that P'' is essentially the process that executes every possible action over channels in $sort(P)$ which are not in C and are different from $running\_ini$ and $commit\_res$, or let time pass. Given P, $\alpha^{t(n)}_{tAgree}(P)$ represents the most general system which satisfies the *Timed Agreement* property and has the same sort of P. In fact in $\alpha^{t(n)}_{tAgree}(P)$ action $\overline{running\_ini}(x, y, d)$ always precedes $\overline{commit\_res}(y, x, d)$ for every datum d, and every combination of the other actions of P can be executed. Finally the number of $tick$ actions is at most n. In order to analyze more than one session, it suffices to consider an extended α which has several processes P' in parallel.
We want that even in the presence of a hostile process, P does not execute traces that are not in $\alpha^{t(n)}_{tAgree}(P)$. So we can give the following definition:

Definition 7. *P satisfies Timed Agreement* iff *P is $tGNDC^{\alpha^{t(n)}_{tAgree}(P)}_{\leq_{ttrace}}$, i.e.,*

$$\forall\, X \in t\mathcal{E}^{\phi_I}_C \;:\; (P||_C X) \leq_{ttrace} \alpha^{t(n)}_{tAgree}(P)$$

Example 6. We consider here the Wide Mouthed Frog protocol presented in Section 2.3. In this specification we consider a server that can receive requests from both A and B as initiators. Note that time parameter n is 3.

$Clock(n) = tick.Clock(n+1) + \overline{time}\, n.Clock(n)$

$A(K) = time(T_a).\overline{running\_ini}(A, B, K).tick.\overline{c}_{AS}\{B, T_a, K\}_{K_{as}}.\iota(\mathbf{0})$

$B = \iota(c_{BS}(y)[\langle y, K_{bs}\rangle \vdash_{dec} z]time(T_b).tick.[\langle z\rangle \vdash_{snd} t][t+1 \leq T_b]$
$[\langle z\rangle \vdash_{trd} w]\overline{commit\_res}(B, A, w))$

$S = tick.c_{AS}(u).[\langle u, K_{as}\rangle \vdash_{dec} z]time(T_s).tick.[\langle z\rangle \vdash_{snd} t][t+1 \leq T_s][\langle z\rangle \vdash_{trd}$
$x]\overline{c}_{BS}\{A, T_s, x\}_{K_{bs}}.S \;\;+\;\; tick.c_{BS}(u).[\langle u, K_{bs}\rangle \vdash_{dec} z]time(T_s)$
$.tick.[\langle z\rangle \vdash_{snd} t][t+1 \leq T_s][\langle z\rangle \vdash_{trd} x]\overline{c}_{AS}\{B, T_s, x\}_{K_{as}}.S$

$P(K_{ab}) = (Clock(0) \,|\, A(K_{ab}) \,|\, B \,|\, S)\backslash\{c_{as}, c_{bs}, time\}$

Unfortunately, this protocol could be attacked by an intruder P in the following way:

1. $A \rightarrow S : A, \{B, T_a, K_{ab}\}_{K_{as}}$
2. $S \rightarrow E(B) : \{A, T_a+1, K_{ab}\}_{K_{bs}}$
3. $E(B) \rightarrow S : B, \{A, T_a+1, K_{ab}\}_{K_{bs}}$
4. $S \rightarrow E(A) : \{B, T_a+3, K_{ab}\}_{K_{as}}$
5. $E(A) \rightarrow S : A, \{B, T_a+3, K_{ab}\}_{K_{as}}$
6. $S \rightarrow B : \{B, T_a+5, K_{ab}\}_{K_{bs}}$

Indeed, the protocol P, in parallel with the enemy E, produces the following trace:

$$\overline{running\_ini}(A, B, K_{ab}).tick.tick.tick.tick.tick.\overline{commit\_res}(B, A, K_{ab})$$

that is not in $\alpha^{t(n)}_{tAgree}$ since the number of $ticks$ is greater than $n = 3$. Note that, in fact, the actions $\overline{running\_ini}(A, B, K_{ab})$ and $\overline{commit\_res}(B, A, K_{ab})$ are correctly executed, but the enemy forces B to accept an old key. In an untimed setting this wolud be a correct execution of the protocol. The problem here is, on the one hand that the message produced by S can easily be used by the enemy to start a new run of the protocol having as initiator the intended receiver of the previous run, and on the other hand that when B receives the key, he has no way to know when A forged it.

4.3 Timed Secrecy

We now present the *Timed Secrecy* Property:

> "A protocol guarantees to an initiator A the property of *Timed Secrecy* on a set of data items ds within a time n if, whenever a data item in ds becomes public, at least n ticks passed since A started the protocol"

As done for *Timed Agreement*, what we do is to have an action representing the running of a protocol and another one representing that a secret is revealed. We consider an action $running\_ini(A, d)$ that represents the fact that A is running the protocol as initiator, with data d. On the other hand we have an action $public(d)$ representing that data item d is made public. If we specify these two actions in the protocol, the *Timed Secrecy* property requires that when someone executes $public(d)$ then A has executed $running\_ini(A, d)$ and at least n $tick$ actions, where n is the prefixed timeout value, occurred between them. We assume that the actions representing the running and the publication are correctly specified in the protocol. We can see the first as an output action over a particular channel $running\_ini$. The second action, following the approach of [4] is performed by a particular process called *Key Expired Notifier* (KEN, for short) that reads from a public channel c not used in the protocol and performs the output of what it has read on the channel $\overline{public}$:

$$KEN = c(x).\overline{public}(x)$$

We assume that d can take values in a set of secret values D. Let $NotObs(P) = sort(P) \backslash (C \cup \{c, running\_ini, public\})$ be the set of channels in P that are not public and are different from $running\_ini$ and $public$, i.e., that will not be observed. Function $\alpha_{tSec}^{t(n)}$ can be thus defined as follows:

$$P'(x) = \sum_{d \in D} \overline{running\_ini}(x, d).tick_1. \ldots .tick_n.(\iota(\overline{public}(d).\iota(\mathbf{0})) \; + \iota(\tau.\iota(\mathbf{0})))$$

$$P'' = \sum_{c \in NotObs(P)} \iota(c(x).P'') + \sum_{\substack{c \in NotObs(P) \\ m \in Msg(c)}} \iota(\overline{c}m.P'')$$

$$\alpha_{tSec}^{t(n)}(P) = P'' \,|\, P'(A)$$

Given P, $\alpha_{tSec}^{t(n)}(P)$ represents the most general system which satisfies the *Timed Secrecy* property and has the same sort of P. In fact in $\alpha_{tSec}^{t(n)}(P)$ action $\overline{public}(d)$ is always executed at least n $ticks$ after $\overline{running\_ini}(x, d)$ for every datum d, and every combination of the other actions of P can be executed. In order to analyze more than one session, it suffices to consider an extended α which has several processes P' in parallel.

We want that, even in the presence of a hostile process, P does not execute traces that are not in $\alpha_{tSec}^{t(n)}(P)$. So we can give the following definition:

Definition 8. *P satisfies Timed Secrecy* iff *P is $tGNDC_{\leq_{ttrace}}^{\alpha_{tSec}^{t(n)}(P)}$, i.e.,*

$$\forall\, X \in t\mathcal{E}_C^{\phi_I} \;:\; (P||_C X) \leq_{ttrace} \alpha_{tSec}^{t(n)}(P)$$

Example 7. We consider the following protocol:

1. $A \rightarrow S : A, \{B, T_a, K_{ab}, Exp\}_{K_{as}}$
2. $S \rightarrow B : \{A, T_s, K_{ab}, Exp'\}_{K_{bs}}$

which is a variant of the Wide Mouthed Frog Protocol in which we add a field Exp used to state how long the key will last before expiring. After that time the key is no longer valid and will be made public by B. Both the server S and B check the value Exp and upgrade it according to the amount of time passed. c is the public channel used to throw away expired keys while KEN is the process that we use to write on the private channel $public$.
We assume that we have two servers that update the field Exp in a different way. In particular server S_2 decreases – at double speed w.r.t. S_1 – the key expiration time, so that it is possible to have secure keys that will last less.
$Clock(n) = tick.Clock(n+1) + \overline{time}\, n.Clock(n)$

$A(K) = time(T_a).\overline{running\_ini}(A, K).tick.\overline{c}_{AS_1}\{B, T_a, K, n\}_{K_{as}}.\iota(\mathbf{0})$

$B = \iota(c_{BS}(y)[\langle y, K_{bs}\rangle \vdash_{dec} z]time(T_b).tick.[\langle z\rangle \vdash_{snd} t_s][\langle z\rangle \vdash_{trd} w][\langle z\rangle \vdash_{frt} t]$
$[t - (T_b - t_s) \leq 0]\overline{c}w)$

$KEN = \iota(c(x).\overline{public}(x))$

$S = S_1 \mid S_2$

$S_1 = \iota(c_{AS_1}(u).[\langle u, K_{as}\rangle \vdash_{dec} z]time(T_s).tick.[\langle z\rangle \vdash_{snd} t_a][\langle z\rangle \vdash_{trd} x]\ [\langle z\rangle \vdash_{frt}$
$t][1 < t - (T_s - t_a)]\overline{c}_{BS}\{A, T_s, x, t - (T_s - t_a)\}_{K_{bs}}.tick)$

$S_2 = \iota(c_{AS_2}(u).[\langle u, K_{as}\rangle \vdash_{dec} z]time(T_s).tick.[\langle z\rangle \vdash_{snd} t_a][\langle z\rangle \vdash_{trd} x]\ [\langle z\rangle \vdash_{frt}$
$t][0 < t - (T_s - t_a)]\overline{c}_{BS}\{A, T_s, x, t - 2 * (T_s - t_a)\}_{K_{bs}}.tick)$

$P(K_{ab}) = (Clock(0) \mid A(K_{ab}) \mid B \mid S \mid KEN)\backslash\{c_{as}, c_{bs}, c, time\}$

An enemy E, by using server S_2, could make the key appear as expired to the intended recipient, by simply redirecting to server S_2 the message intended for server S_1. We suppose that $n = 4$. The enemy could attack the protocol in the following way:

1. $A \rightarrow E(S_1) : A, \{B, 0, K_{ab}, 4\}_{K_{as}}$
2. $E(A) \rightarrow S_2 : A, \{B, 0, K_{ab}, 4\}_{K_{as}}$
3. $S_2 \rightarrow B : \{A, 1, K_{ab}, 2\}_{K_{bs}}$

As B receives the third message and upgrades the value of Exp, it becomes 1 and so the key will be made public even if only 3 $ticks$ passed.
The protocol P composed in parallel with enemy E could produce the following trace:

$$\overline{running\_ini}(A, K_{ab}).tick.tick.tick.\overline{public}(K_{ab})$$

that is not in $\alpha_{tSec}^{t(n)}$ since the number of $ticks$ is less than $n = 4$.

5 Compositionality results

In this section we illustrate a compositional proof method for establishing that a system enjoys a particular $tGNDC$ property, namely $tSNNI^{\phi}_{C}$. However, remember that, as it is equivalent to $tNDC$, this property implies all the other ones based on trace semantics, that are the most frequently used.

Within the SPA theory, $SNNI$ is compositional, i.e. if $P, Q \in SNNI$ then $(P \,|\, Q) \in SNNI$. Unfortunately, the same does not hold when considering enemies with limited knowledge, as for $tSNNI^{\phi}_{C}$. For instance, consider the processes:

$$P = \overline{c_1} m_1.c_2(x)[x = m_2].\overline{c_3} m_2$$
$$Q = \overline{c_1} m_2.c_2(x)[x = m_1].\overline{c_3} m_1$$

Now, assuming $C = \{c_1, c_2\}$ and $\phi = \emptyset$, we have that $P, Q \in tSNNI^{\phi}_{C}$. However, $P \,|\, Q \notin tSNNI^{\phi}$. As a matter of fact, $(P \,|\, Q)\backslash C$ is equivalent to **0**, while $(P \,|\, Q)/^{\phi} C$ may perform both $\overline{c_3} m_1$ and $\overline{c_3} m_2$.

However, if we strengthen the assumptions we can get a compositional rule for establishing that a process belongs to $tSNNI^{\phi}_{C}$. The *stability* assumption we make is that the process cannot increment its knowledge.

Definition 9. *We say that a process P is stable w.r.t. ϕ, whenever if $P/^{\phi} C \Longrightarrow P'/^{\phi'} C$ then $\mathcal{D}(\phi) = \mathcal{D}(\phi')$.*

Thus, the following proposition holds.

Proposition 8. *Assume that $P, Q \in tSNNI^{\phi}_{C}$ and that P, Q are stable w.r.t. ϕ. Then $(P \,|\, Q) \in tSNNI^{\phi}_{C}$ and $P \,|\, Q$ is stable w.r.t. ϕ.*

Proof. First, we show that $P \,|\, Q$ is stable. For the sake of simplicity, we assume that $\phi = \mathcal{D}(\phi)$. By contradiction, consider a computation s.t.

$$P \,|\, Q/^{\phi} C \Longrightarrow P'' \,|\, Q''/^{\phi''} C$$

and $\mathcal{D}(\phi'') \neq \mathcal{D}(\phi) = \phi$. Then, it means that $P \,|\, Q$ outputs a message m s.t. $m \notin \phi$. Let m_1 be the first of such messages sent during the computation under consideration. Assume, without loss of generality, that m_1 has been sent by P. Thus, we must have:

$$P \,|\, Q/^{\phi} C \Longrightarrow P' \,|\, Q'/^{\phi} C \longrightarrow P'' \,|\, Q''/^{\phi''} C$$

where during the prefix of the computation Q emitted only message in ϕ. So, the trace may be simulated by P in composition with Top^{ϕ}_{C}. But this implies that $P/^{\phi} C \Longrightarrow P''/^{\phi''} C$ and $\mathcal{D}(\phi'') \neq \mathcal{D}(\phi)$. This leads to a contradiction since we assumed P stable w.r.t. ϕ.

Now, we show that $P \,|\, Q$ is in $tSNNI^{\phi}_{C}$. Clearly, since $=_{ttrace}$ is a congruence w.r.t. the parallel composition we have that traces $T((P \,|\, Q)\backslash C) \subseteq T((P \,|\, Q \,|\, Top^{\phi}_{C})\backslash C)$. On the other hand, suppose that $(P \,|\, Q \,|\, Top^{\phi}_{C})\backslash C$ performs a trace that does not belong to $T((P \,|\, Q)\backslash C)$. Consider the shortest of such traces:

$$P \,|\, Q/^{\phi} C \stackrel{\gamma}{\Longrightarrow} P' \,|\, Q'/^{\phi} C \stackrel{\overline{c} m}{\longrightarrow} P'' \,|\, Q''/^{\phi''} C$$

where the output is due to process P (without loss of generality). The part of the computation performed by P is γ_1. Now, the process P may have received only messages in ϕ from Top and Q in the channels in C, since $P \,|\, Q$ is stable, and each possible message in $\mathcal{M}$ on the channels not restricted. Thus, $P/^{\phi}C \stackrel{\gamma_1}{\Longrightarrow} P'/^{\phi}C \stackrel{\overline{c}m}{\longrightarrow}$. However, also $P\backslash C \stackrel{\gamma_1 \overline{c}m}{\Longrightarrow}$ since $P \in tSNNI_C^{\phi}$. This leads to a contradiction, since in this case also $P \,|\, Q$ may produce the previous computation. ■

Example 8. We consider a simple example of the application of the principle above. Consider the processes $P = \overline{c}\{m\}_k.0$ and $Q = c?x.[x \quad k \vdash_{dec} z]\tau$. Consider also the process $KEN' = c?x.[x = m].\overline{public}\, m$. Assuming $\phi = \{\{m\}_k\}$ and $C = \{c\}$, we can establish that $P, Q, KEN' \in tSNNI_C^{\phi}$ and are all stable; hence $P \,|\, Q \,|\, KEN' \in tSNNI_C^{\phi}$, which means that $P \,|\, Q$ keeps m secret. As a matter of fact, $(P \,|\, Q \,|\, KEN')\backslash C =_{ttrace} \mathbf{0}$.

We have another compositionality principle for the $tGNDC^{\alpha}_{\leq_{ttrace}}$ schema, again under the assumption that the involved processes are stable.

Proposition 9. *Given the set of initial knowledge ϕ and the set of public channels C, assume $P_i \in tGNDC^{\alpha_i(P_i)}_{\leq_{ttrace}}$, with $i = 1, 2$, and P_1, P_2 are stable w.r.t. ϕ. It follows that $(P_1 \,|\, P_2) \in tGNDC^{\alpha_1(P_1) \,|\, \alpha_2(P_2)}_{\leq_{ttrace}}$ and $P_1 \,|\, P_2$ is stable w.r.t. ϕ.*

Proof. The proof follows by noticing that $(P_1 \,|\, P_2)/^{\phi}C \leq_{ttrace} P_1/^{\phi}C \,|\, P_2/^{\phi}C$. Then, by hypothesis and Proposition 6, we have $P_i/^{\phi}C \leq_{ttrace} \alpha_i(P_i)$, with $i = 1, 2$, and the thesis then follows because $\leq_{ttrace}$ is a pre-congruence w.r.t. $|$.

Example 9. Consider the following family of processes $S(i)$, each sending a message (m_i, i) after every time unit:

$$S(i) = \iota(tick.[(m_i, i) \quad pk_A \vdash_{enc} z].\overline{c}z.S(i+1))$$

Consider also a family of receivers of this kind:

$$R(i) = tick.c(y)[y \quad pk_A^{-1} \vdash_{dec} t][t \vdash_{snd} t_2][t_2 = i][t \vdash_{fst} t_1]\overline{out}\, t_1.R(i+1)$$

Basically, we have that $(S(0) \,|\, R(0))\backslash C$, where $C = \{c\}$, is trace included into $Spec(0)$ where

$$Spec(i) = tick.\overline{out}\, m_i.Spec(i+1)$$

Consider $\phi = \{\{(m_i, i)\}_{pk_A} \;|\; 0 \leq i\} \cup \{pk_A\}$ as the intruder's knowledge set. Then, we have that $S(0)$ and $R(0)$ are stable w.r.t. ϕ and $S(0) \in tNDC_C^{\phi}$, with $S(0)\backslash C =_{ttrace} \iota(\mathbf{0})$, and $R(0) \in tGNDC^{Spec(0)}_{\leq_{ttrace}}$. By Proposition 9, we have that

$$(S(0) \,|\, R(0) \,|\, Top_C^{\phi})\backslash C \leq_{ttrace} \mathbf{0} \,|\, Spec(0) \leq_{ttrace} Spec(0)$$

It is worth-while observing that the stability condition is not only sufficient for it, but it is also strictly necessary to get any general compositionality result. Without it, it is always possible for the enemy to interfere with the correct execution of the two subprotocols P and Q by, e.g., providing P with knowledge of Q.

One may wonder if the stability condition is too restricitve, as in the example above the potential enemies must know in advance the set of exchanged messages. As a matter of fact, it seems that for integrity purposes the compositional proof principle can be successfully applied.

6 Conclusions

We have shown how to extend the $GNDC$ schema to a real time setting while preserving the properties of the untimed schema. In particular, we have shown the existence of a ”most powerful” timed enemy, at least for the timed trace semantics. We have also shown how to express uniformly in this general schema some timed security properties, such as timed Non Deducibility on Compositions, (one definition of) timed authentication, timed secrecy and, in the final example, also timed integrity. We have also introduced a compositional proof principle that allows us to compose safely two real-time security protocols, preserving the security properties they enjoy.

Related literature on real-time security include the prominent papers [12, 1]. The former paper presents tock-CSP – a real-time language very similar to tSPA – that is used to specify real-time crytpographic protocols; timed agreement is defined and analysed on the WMF protocol. The attack reported in Section 4.2 is essentially the same reported there; moreover, the definition of timed agreement is very similar. The main differences consists of a different treatment of timed operators, in the absence of a mechanism for handling crypto-primitives, in the lack of a uniform schema, and in the absence of compositionality results. The latter paper [1] is mainly concerned with the model checking of the interesting case study of TESLA, a protocol for stream broadcasting over the internet. The main focus is on showing that it is possible to give a finite model for the unbounded supply of fresh cryptographic keys used during the protocol. However, the so-called *security condition* of the protocol reminds somehow timed agreement, as we have defined in Section 4.3.

Future work will be devoted to study other security properties in a timed setting, such as non repudiation, for which apparently there is the need for using semantics more discriminating than timed trace semantics. Moreover, the extension of SPA with the primitive for handling cryptography and with real-time is rather plain and follows classic techniques: integration of a deduction system and of the fictitious clock approach. Hence, it is quite reasonable to envisage that existing tools can be enhanced to accomodate $tCryptoSPA$ specifications; such tools would help in automatic (or semi-automatic) analyses of real-time cryptographic protocols.

References

1. P. Broadfoot, G. Lowe. Analysing a Stream Authentication Protocol using Model Checking In Procs. ESORICS'02, LNCS, Springer, 2002.
2. J. Clark and J. Jacob. A Survey of Autentication Protocol Literature: Version 1.0. http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz

3. R. Focardi and R. Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(1):5–33, 1995.
4. R. Focardi, R. Gorrieri and F.Martinelli. Secrecy in Security Protocols as Non Interference *Electronic Notes in Theoretical Computer Science*, 32 (2000)
5. R. Focardi, R. Gorrieri and F.Martinelli. Non-interference for the analysis of cryptographic protocols. In Procs. *Int.l Colloquium on Automata, Languages and Programming* (ICALP'00), LNCS 1853, 354-372, Springer, 2000.
6. R. Focardi, R. Gorrieri and F.Martinelli. Real-time Information Flow Analysis *Journal of Selected Areas of Communications*, IEEE Press, 2002, to appear.
7. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proceedings of World Congress on Formal Methods (FM'99)*, pages 794–813. Springer, LNCS 1708, 1999.
8. M. Hennessy and T. Regan. A temporal process algebra. *Information and Computation*, 117:221–239, 1995.
9. J.F.Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118:263–299, 1993.
10. G. Lowe. A Hiearchy of Autentication Specifications 10^{th} Computer Society Foundation Workshop Proceedings, pp. 31-43, IEEE Computer Society Press.
11. R. Milner. Communication and Concurrency. Prentice Hall, 1989.
12. S. Schneider. Analysing Time-Dependent Security Properties in CSP using PVS. In Procs. ESORICS, LNCS 1895, 2000.
13. I. Ulidowski and S. Yuen. Extending process languages with time. In *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology*, volume 1349 of *Lecture Notes in Computer Science*, pages 524–538, 1997.