A Formal Analysis for Capturing Replay Attacks in Cryptographic Protocols *

Han Gao[1], Chiara Bodei[2], Pierpaolo Degano[2], and Hanne Riis Nielson[1]

[1] Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads bldg 322, DK-2800 Kongens Lyngby, Denmark. {hg,riis}@imm.dtu.dk

[2] Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo, 3, I-56127, Pisa, Italy. {chiara,degano}@di.unipi.it

Abstract. We present a reduction semantics for the LySA calculus extended with session information, for modelling cryptographic protocols, and a static analysis for it. If a protocol passes the analysis then it is free of replay attacks and thus preserves freshness. The analysis has been implemented and applied to a number of protocols, including both original and corrected version of Needham-Schroeder protocol. The experiment results show that the analysis is able to capture potential replay attacks.

1 Introduction

Since the 80's, formal analyses of cryptographic protocols have been widely studied. Many formal methods have been put forward. Particular significant is the one built by Dolev and Yao. Indeed, most of the formal analysis tools were built upon it, e.g. Meadows and Syverson NRL [18], Millen Interrogator [19], Paulson inductive method [23], based on Isabelle [24], Blanchet's Prolog protocol verifier[2] and BAN logic [7], a logic of authentication used to analyse protocols, etc. Each tool is equipped to detect a certain amount of attacks, including replay attacks.

Replay attacks are classified by Syverson in [25] at the highest level as run-external and run-internal attacks, depending on the origin of messages. In this paper, we restrict our attention to *run-external attacks*. This type of attacks allows the attacker to achieve messages from one run of a protocol, often referred to as a *session*, and to send them to a principal participating in another run of the protocol. A *fresh* message means that it is not replayed from another session (old session or parallel session). In BAN logic, reasoning about the freshness of an entire message amounts to reasoning about the freshness of its fields, i.e. "if one part of a formula is known to be fresh, then the entire formula must also be fresh". We take advantage of the fact that the attacker can manipulate any message in clear, but it has no direct control on the encrypted messages. Indeed, in out framework, after each successful decryption, we check whether the

* This work has been partially supported by the project SENSORIA.

decrypted message is a replayed one from another session, which is a violation of freshness property.

Here we extend the LySA calculus [3, 4] with annotations about sessions and we extend the control flow analysis in [3, 4] as well. As expected, the new control flow analysis soundly over-approximates the behavior of protocols, by tracking the set of messages that are communicated over the network, and recording the potential values of variables. Since our analysis is sound, we capture malicious activities, if any, expressed in terms of annotation violations. Our static analysis is fully automatic and termination is always guaranteed. The proposed analysis has been implemented. The resulting tool was applied to some cryptographic protocols, such as Otway-Rees [22] and Needham-Schroeder [21].

As far as the security properties are concerned, replay attacks on security protocols can cause authentication and/or confidentiality violations. Besides the other security properties, e.g. authentication and confidentiality, checked with the CFA in [3, 4] we here are able to address an orthogonal property like freshness. We analyse the Wide Mouthed Frog protocol and the Needham-Schroeder protocol, both of which do not achieve freshness property in the presence of a replay attacker.

The paper is organized as follows. In Section 2, we present the LySA calculus annotated with session information. We introduce the control flow analysis in Section 3. In Section 4 we describe a Dolev-Yao attacker extended to fit into our particular setting. In section 5, we make some experiments in analyzing two versions of the Needham-Schoreder symmetric key protocol. Section 6 concludes the paper.

2 A Reduction Semantics for the LySA Calculus

LySA [3, 4] is a process algebra, in the tradition of the π- [20] and Spi- [1] calculi. Among its peculiar features, there are: (1) the absence of channels: in LySA all processes have only access to a single global communication channel, the ether and (2) tests associated with input and decryption are expressed using pattern matching.

2.1 Syntax

LySA consists of terms and processes. The syntax of terms E and processes P is given below. Here $\mathcal{N}$ and $\mathcal{X}$ denote sets of names and variables, respectively. For the sake of simplicity, we only consider here some basic terms and encryptions. The name n is used to represent keys, challenges and names of principals. Encryptions are tuples of terms $E_1, \ldots, E_k$ encrypted under a shared key represented by the term E_0. We assume perfect cryptography in this paper.

$$
\begin{array}{rl}
E ::= & n \mid x \mid \{E_1, \ldots, E_k\}_{E_0} \\
P ::= & \langle E_1, \ldots, E_k \rangle.P \mid (E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P \mid \\
& \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}^{l}_{E_0}\ \mathsf{in}\ P \mid \\
& (\nu\ n)P \mid P_1 | P_2 \mid\ !P \mid 0
\end{array}
$$

In addition to the classical constructs for composing processes, LySA also contains an input construct with matching and a decryption operation with matching. The idea behind the matching is as follows: we allow a prefix of the received tuple to match a selection of values. If the test is passed, the remaining values are bound to the relevant variables. The label l in the decryption construct uniquely identifies each decryption point, which is from a numerable set Lab ($l \in Lab$), and is mechanically attached to processes.

Extended LySA We change the syntax of standard LySA so that each term and process now carries an identifier of the session it belongs to. In what follows, we assume that SID is a fixed enumerable set of session identifiers s, and we denote $\mathcal{E}_1, \mathcal{E}_2, \ldots$ the extended terms and $\mathcal{P}, \mathcal{Q}, \ldots$ the extended processes defined below. Note that variables carry no annotation and therefore we shall consider $[x]_s$ and x to be the same (see below). Furthermore, there is no need for the nil process (0) to carry session information and hence $[0]_s$ and 0 are identical.

$$\begin{array}{l}\mathcal{E} ::= [n]_s \mid x \mid [\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}]_s \\ \mathcal{P} ::= \langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle.\mathcal{P} \mid (\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k).\mathcal{P} \mid \\ \qquad \mathsf{decrypt}\ \mathcal{E}\ \mathsf{as}\ \{\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k\}^l_{\mathcal{E}_0}\ \mathsf{in}\ \mathcal{P} \mid \\ \qquad (\nu\ [n]_s)\mathcal{P} \mid \mathcal{P}_1 | \mathcal{P}_2 \mid [!P]_s \mid 0\end{array}$$

We define a function $\mathcal{F}$ and a function $\mathcal{T}$, in the style of [9], that map standard terms and processes into the extended ones, by attaching the session identifiers inductively. Note that $\mathcal{F}$ unwinds the syntactic structure of an extended term until reaching a basic term (a name or a variable), while $\mathcal{T}$ unwinds the structure of an extended process until reaching a nil (which is untagged) or a replication.

Definition 1. *Distributing Session Identifiers*

$$\mathcal{F} : E \times SID \to \mathcal{E}$$

$$-\mathcal{F}(n, s) = [n]_s \qquad -\mathcal{F}(x, s) = x$$
$$-\mathcal{F}(\{E_1, \ldots, E_k\}_{E_0}, s) = [\{\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_k, s)\}_{\mathcal{F}(E_0, s)}]_s$$

$$\mathcal{T} : P \times SID \to \mathcal{P}$$

$$-\mathcal{T}(\langle E_1, \ldots, E_k \rangle.P, s) = \langle \mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_k, s) \rangle.\mathcal{T}(P, s)$$
$$-\mathcal{T}((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k).P, s) = (\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_j, s); x_{j+1}, \ldots, x_k).\mathcal{T}(P, s)$$

$$-\mathcal{T}(\mathit{decrypt}\ E\ \mathit{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}^l_{E_0}\ \mathit{in}\ P, s) = \mathit{decrypt}\ \mathcal{F}(E, s)\ \mathit{as}\ \{\mathcal{F}(E_1, s), \ldots, \mathcal{F}(E_j, s); x_{j+1}, \ldots, x_k\}^l_{\mathcal{F}(E_0, s)}\ \mathit{in}\ \mathcal{T}(P, s)$$

$$-\mathcal{T}(P \mid Q, s) = \mathcal{T}(P, s) \mid \mathcal{T}(Q, s) \qquad -\mathcal{T}((\nu\ n)P, s) = (\nu\ [n]_s)\mathcal{T}(P, s)$$
$$-\mathcal{T}(!P, s) = [!P]_s \qquad -\mathcal{T}(0, s) = 0$$

2.2 Operational Semantics

Below we assume the standard *structural congruence* $\equiv$ on LySA processes, as the least congruence satisfying the following clauses (as usual $fn(P)$ is the set of the free names of P):

$$\begin{array}{ll} P \mid 0 \equiv P & (\nu x)0 \equiv 0 \\ P \mid Q \equiv Q \mid P & (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \\ (P \mid Q) \mid R \equiv P \mid (Q \mid R) & (\nu x)(P \mid Q) \equiv P \mid (\nu x)Q \text{ if } x \notin fn(P) \\ P \equiv Q \text{ if } P \text{ and } Q \text{ are } \alpha\text{-equivalent} & \end{array}$$

Technically, the addition of session identifiers to the syntax of LySA means that it is necessary to carry on the session identifiers to the semantics of values, i.e. terms without variables. The extended value domain will be referred to as Val, ranged over by V built from the grammar $V ::= [n]_s \mid [\{V_1, \ldots, V_k\}_{V_0}]_s$ The equivalence relation $V_1 \stackrel{f}{=} V_2$ is defined to be the least equivalence over Val that (inductively) ignores the session identifers. For example, $[n]_s \stackrel{f}{=} [n]_{s'}$ for any s and s' and $[\{[n_1]_{s_1}, [n_2]_{s_2}\}_{[n_0]_{s_0}}]_s \stackrel{f}{=} [\{[n_1]_{s'_1}, [n_2]_{s'_2}\}_{[n_0]_{s'_0}}]_{s'}$ for any s, s', s_1, s_2, s'_1 and s_2. For the subsequent treatment, it is convenient introducing an auxiliary operator, $\mathcal{I}$, which extracts the (outermost) session identifier of an extended value V.

Definition 2. *Extracting Session Identifers* $\mathcal{I}$*:* $Val \rightarrow SID$

$- \ \mathcal{I}([n]_s) = s \qquad - \mathcal{I}([\{v_1, \ldots, v_k\}_{v_0}]_s) = s$

In BAN logic [7], the freshness property is described as "if one part of a formula is known to be fresh, then the entire formula must also be fresh", formally

$$\frac{P|\equiv \sharp(X)}{P|\equiv \sharp(X, Y)}$$

Because of the presence of the network attacker, who can manipulate any message in clear, we shall here only focus on the encrypted messages, which is not directly under the control of the attacker. Namely, after each successful decryption, we check whether there is any field of the encrypted tuple such that its session identifier is the same as expected. This point is made clearer in the semantics shown below.

Following the tradition of the π-calculus, we shall give the extended LySA a reduction semantics. The *reduction relation* $\rightarrow_{\mathcal{R}}$ is the least relation on closed processes that satisfies the rules in Table below and uses the standard notion of substitution, $\mathcal{P}[V/x]$ and structural congruence, as defined above.

As far as the semantics is concerned, we consider two variants of *reduction relation* $\rightarrow_{\mathcal{R}}$, identified by a different instantiation of the relation $\mathcal{R}$, which decorates the transition relation. One variant ($\rightarrow_{\mathsf{RM}}$) takes advantage of annotations, the other one ($\rightarrow$) discards them: essentially, the first semantics checks freshness of messages, while the other one does not (see below):

– the *reference monitor semantics* $\mathcal{P} \rightarrow_{\mathsf{RM}} \mathcal{Q}$ takes $\mathsf{RM}(s, s') = (s = s')$
– the *standard semantics* $\mathcal{P} \rightarrow \mathcal{Q}$ takes, by construction, $\mathcal{R}$ to be universally true.

More specifically, after each successful decryption the reference monitor checks whether *at least one* field of the encrypted message is coming from the expected session, i.e. it is *fresh*, which makes the entire encryption such.

$$\text{(Com)}\quad \frac{\wedge_{i=1}^{j} V_i \stackrel{f}{=} V_i'}{\langle V_1, \ldots, V_k\rangle.\mathcal{P} \mid (V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k).\mathcal{P}' \rightarrow_{\mathcal{R}} \mathcal{P} \mid \mathcal{P}'[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k]}$$

$$\text{(Dec)}\quad \frac{\wedge_{i=0}^{j} V_i \stackrel{f}{=} V_i' \ \wedge\ \vee_{i=1}^{j} \mathcal{R}(\mathcal{I}(V_i), \mathcal{I}(V_i'))}{\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k\}_{V_0'}^{l}\ \mathsf{in}\ \mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}[V_{j+1}'/x_{j+1}, \ldots, V_k'/x_k]}$$

$$\text{(Res)}\quad \frac{\mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{P}'}{(\nu\ [n]_s)\mathcal{P} \rightarrow_{\mathcal{R}} (\nu\ [n]_s)\mathcal{P}'} \qquad \text{(Repl)}\quad [!P]_s \rightarrow_{\mathcal{R}} \mathcal{T}(P, s) \mid [!P]_{s'}\quad (s' \text{ is fresh})$$

$$\text{(Par)}\quad \frac{\mathcal{P}_1 \rightarrow_{\mathcal{R}} \mathcal{P}_1'}{\mathcal{P}_1 \mid \mathcal{P}_2 \rightarrow_{\mathcal{R}} \mathcal{P}_1' \mid \mathcal{P}_2} \qquad \text{(Congr)}\quad \frac{P \equiv P' \ \wedge\ \mathcal{T}(P', s) \rightarrow_{\mathcal{R}} \mathcal{T}(P'', s)}{\mathcal{T}(P, s) \rightarrow_{\mathcal{R}} \mathcal{T}(P'', s)}$$

The rule (Com) expresses that an output $\langle V_1, \ldots, V_j, V_{j+1}, \ldots, V_k\rangle.\mathcal{P}$ matches an input $(V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k)$ in case the first j values are pairwise equal (under the equivalence $\stackrel{f}{=}$) when all the annotations are recursively removed. When the matching is successful each V_i is bound to the corresponding x_i. Note that the equivalence relation $\stackrel{f}{=}$ is defined over the extended value domain Val.

Similarly, the rule (Dec) expresses the result of matching an encryption $[\{V_1, \ldots, V_k\}_{V_0}]_s$ with $\mathsf{decrypt}\ V\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}, \ldots, x_k\}_{V_0'}\ \mathsf{in}\ \mathcal{P}$. As it was the case for communication, the first j values V_i and V_i' must be equal, and additionally the keys must be equal, i.e. $V_0 \stackrel{f}{=} V_0'$. When the matching is successful, each V_i is bound to the corresponding x_i. In the *reference monitor semantics* we ensure that the decrypted message comes from the current session by checking whether any of the first j values V_i and V_i' have the same session identifiers. In the *standard semantics* the disjunction $\vee_{i=j+1}^{k} \mathcal{R}(\mathcal{I}(V_i), \mathcal{I}(V_i'))$ is universally true and thus can be ignored.

In case of (Repl), the process is unfolded once. Note that the new session identifier, s', in this case, has to be unique, i.e. not occurring anywhere else along the evolution of the process $\mathcal{P}$. This makes sure that each copy of a protocol process has a unique session identifer such that different copies will not be mixed up.

The rule (Congr) makes use of the function $\mathcal{T}$, which bridges the gap between the semantics defined on the extended processes $\mathcal{P}$ and the structural congruence defined on the standard processes P.

The rules (Res) and (Par) are standard.

Following the line of BAN logic, the freshness of a LySA process is defined as follows:

Definition 3 (Freshness). *A process $\mathcal{P}$ ensures freshness property if for all the possible executions $\mathcal{P} \rightarrow^*_{\mathcal{R}} \mathcal{P}' \rightarrow \mathcal{P}''$ when $\mathcal{P}' \rightarrow \mathcal{P}''$ is derived using (Dec) on*

$$\mathit{decrypt}\ [\{V_1, \ldots, V_k\}_{V_0}]_s\ \mathit{as}\ \{V'_1, \ldots, V'_j; x_{j+1}, \ldots, x_k\}^l_{V_0}\ \mathit{in}\ \mathcal{P}$$

there exists at least one i $(1 \leq i \leq j)$ such that $\mathcal{I}(V_i) = \mathcal{I}(V'_i)$

It says that an extended process $\mathcal{P}$ ensures freshness property if there is no violation of the annotations in any of its executions.

2.3 Example

We shall use the simplified version (without timestamps) of the Wide Mouthed Frog protocol [7] (WMF) for illustrating how to encode protocols in our calculus. WMF is a symmetric key management protocol aiming at establishing a secret session key K_{ab} between the two principals A and B sharing secret master keys K_A and K_B, respectively, with a trusted server S. The protocol is specified by the following informal narration:

$$\begin{array}{l} 1.\ A \rightarrow S : \{B, K_{ab}\}_{K_A} \\ 2.\ S \rightarrow B : \{A, K_{ab}\}_{K_B} \\ 3.\ B \rightarrow A : \{Msg\}_{K_{ab}} \end{array}$$

The extended LySA specification of the WMF protocol is $[!P]_0$ where $P = (\nu\ K_A)(\nu\ K_B)(A|B|S)$ contains three processes A, B and S, running in parallel, each of them models one principal's activity, and is as follows:

$$\begin{array}{lll} 1.\ A & & (\nu\ K_{ab}) \\ \quad A \rightarrow & & \quad \langle A, S, \{B, K_{ab}\}_{K_A}\rangle. \\ 3'.\ \quad \rightarrow A & & \quad\ (B, A; z). \\ 3''.\ \quad\ A & & \quad\ \mathsf{decrypt}\ z\ \mathsf{as}\ \{; z_m\}^{l1}_{K_{ab}}\ \mathsf{in}\ 0 \\ 2'.\ \quad \rightarrow B & \mid & (S, B; y). \\ 2''.\ \quad\ B & & \quad\ \mathsf{decrypt}\ y\ \mathsf{as}\ \{A; k\}^{l2}_{K_B}\ \mathsf{in} \\ 3.\ B & & \quad\ (\nu\ Msg) \\ \quad B \rightarrow & & \quad\ \langle B, A, \{Msg\}_k\rangle.0 \\ 1'.\ \quad \rightarrow S & \mid & (A, S; p). \\ 1''.\ \quad\ S & & \quad\ \mathsf{decrypt}\ p\ \mathsf{as}\ \{B; k'\}^{l3}_{K_A}\ \mathsf{in} \\ 2.\ S \rightarrow & & \quad\ \langle S, B, \{A, k'\}_{K_B}\rangle.0 \end{array}$$

3 Static Analysis

The LySA calculus is especially designed to model security protocols involving a number of principals, where each of them execute a sequence of actions, synchronised by communications. Because of interactions, in most of the cases, it is

impossible to predict the exact behaviour of each principal. In this section, we present a control flow analysis aiming at collecting the central aspect of the information of a protocol of interest. This is done by over-approximating at static time the protocol behaviour along all the execution paths.

3.1 Domain of the Analysis

The control flow analysis describes a protocol behaviour by collecting all the communications that a process may participate in. This information, i.e. the tuples of values that maybe communicated over the network, is recorded in an analysis component κ, i.e. $\kappa \subseteq \wp(Val^*)$ is the *abstract network environment* that includes all the tuples forming a message that may flow on the network. As said before, successful communications involve pattern matching and variable binding, i.e. binding values to variables. To collect this information, we introduce another analysis component $\rho : \mathcal{X} \to \wp(Val)$ that maps the variables to the sets of values that they may be bound to.

Name Space Both the analysis components κ and ρ have to do with recording values $V \in Val$ in some format. However, a LySA process may generate infinitely many values during an execution because of the restriction and replication constructs, e.g. $!(\nu\ n)\langle n\rangle$, which means that the analysis components have to be able to record infinitely many names.

For keeping the analysis component finite, we partition all the names used by a process into finitely many equivalence classes and we use the names of the equivalence classes instead of the actual names. This partition works in a way that names from the same equivalence class are assigned a common *canonical name* and consequently there are only finitely many canonical names in any execution of a given process. This is enforced by assigning the same canonical name to every name generated by the same restriction. The canonical name $\lfloor n \rfloor$ is for a name n; similarly $\lfloor x \rfloor$ is for a variable x. For example, a process, that may generate infinitely many names, is $!(\nu\ n)P$, as shown in the following chain of equivalences: $!(\nu\ n)P \equiv (\nu\ n')P' \mid !(\nu\ n)P \equiv (\nu\ n')P' \mid (\nu\ n'')P'' \mid !(\nu\ n)P \equiv \ldots$ Furthermore, the names n, n' and n'' are generated by the same restriction and hence have the same canonical name, i.e. $\lfloor n \rfloor = \lfloor n' \rfloor = \lfloor n'' \rfloor$. Hereafter, when unambiguous, we shall simply write n (resp. x) for $\lfloor n \rfloor$ (resp. $\lfloor x \rfloor$).

3.2 Analysis of Terms and Processes

For each term $\mathcal{E}$, the analysis will determine a superset of the possible values it may evaluate to. The judgement for terms takes the form $\rho \models \mathcal{E} : \vartheta$ where $\vartheta \subseteq Val$ is an acceptable *estimate* (i.e. a sound over-approximation) of the set of values that $\mathcal{E}$ may evaluate to in the environment ρ. The judgement is defined by the axioms and rules in the upper part of Table below. Basically, the rules demand that ϑ contains all the values associated with the components of a term. In the sequel we shall use two kinds of membership tests: the usual $V \in \vartheta$ that

simply tests whether V is in the set ϑ and the *faithful* test $V \propto \vartheta$ that holds if there is a value V' in ϑ that equals V, when the annotations are inductively ignored.

The judgement for processes has the form: $\rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \psi$ expressing that ρ, κ and ψ are valid analysis estimates of process $\mathcal{P}$. The additional component $\psi \subseteq \wp(Lab)$ is the possibly empty set of error-component which collects an over-approximation of the freshness violations: a label $l \in \psi$ means that the value binding after a successful decryption, marked with label l, violates the freshness annotations and therefore is not allowed. We prove in Theorem 2 (in Section 3.1) that when $\psi = \emptyset$ we may do without the reference monitor. The judgement is defined by the axioms and rules in the lower part of Table below (where $A \Rightarrow B$ means that B is analyzed only when A is *true*) and are explained below. Note that we only check whether a proposed triple, (ρ, κ, ψ), is indeed valid; the algorithm to build solutions is sketched in Section 5.1.

$$\text{(Name)}\ \frac{[n]_s \in \vartheta}{\rho \models [n]_s : \vartheta} \qquad \text{(Var)}\ \frac{\rho(x) \subseteq \vartheta}{\rho \models x : \vartheta}$$

$$\text{(Enc)}\ \frac{\begin{array}{c}\wedge_{i=0}^{k} \rho \models \mathcal{E}_i : \vartheta_i\ \wedge \\ \forall V_0, \ldots, V_k : \wedge_{i=0}^{k} V_i \in \vartheta_i \Rightarrow [\{V_1, \ldots, V_k\}_{V_0}]_s \in \vartheta\end{array}}{\rho \models [\{\mathcal{E}_1, \ldots, \mathcal{E}_k\}_{\mathcal{E}_0}]_s : \vartheta}$$

$$\text{(Out)}\ \frac{\begin{array}{c}\wedge_{i=1}^{k} \rho \models \mathcal{E}_i : \vartheta_i\ \wedge \\ \forall V_1, \ldots, V_k\ \wedge_{i=1}^{k} V_i \in \vartheta_i \Rightarrow \\ \langle V_1, \ldots, V_k\rangle \in \kappa\ \wedge \rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \psi\end{array}}{\rho, \kappa \models_{\mathsf{RM}} \langle \mathcal{E}_1, \ldots, \mathcal{E}_k\rangle.\mathcal{P} : \psi}$$

$$\text{(Inp)}\ \frac{\begin{array}{c}\wedge_{i=1}^{j} \rho \models \mathcal{E}_i : \vartheta_i\ \wedge \\ \forall \langle V_1, \ldots, V_k\rangle \in \kappa : \wedge_{i=1}^{j} V_i \propto \vartheta_i \Rightarrow \\ \wedge_{i=j+1}^{k} V_i \in \rho(x_i)\ \wedge \rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \psi\end{array}}{\rho, \kappa \models_{\mathsf{RM}} (\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k).\mathcal{P} : \psi}$$

$$\text{(Dec)}\ \frac{\begin{array}{c}\rho \models \mathcal{E} : \vartheta\ \wedge\ \wedge_{i=0}^{j} \rho \models \mathcal{E}_i : \vartheta_i \wedge \\ \forall [\{V_1, \ldots, V_k\}_{V_0}]_s \in \vartheta : \wedge_{i=0}^{j} V_i \propto \vartheta_i \Rightarrow \\ (\wedge_{i=j+1}^{k} V_i \in \rho(x_i)\ \wedge \rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \psi\ \wedge \\ (\nexists i : 1 \leq i \leq k : (\mathcal{I}(V_i) = \mathcal{I}(\mathcal{E}_i)) \Rightarrow l \in \psi))\end{array}}{\rho, \kappa \models_{\mathsf{RM}} \mathsf{decrypt}\ \mathcal{E}\ \mathsf{as}\ \{\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k\}^{l}_{\mathcal{E}_0}\ \mathsf{in}\ \mathcal{P} : \psi}$$

$$\text{(Rep)}\ \frac{\rho, \kappa \models_{\mathsf{RM}} \mathcal{T}([P]_s) : \psi\ \wedge\ \rho, \kappa \models_{\mathsf{RM}} \mathcal{T}([P]_{s'}) : \psi}{\rho, \kappa \models_{\mathsf{RM}} [!P]_s : \psi} \qquad \text{(Nil)}\ \rho, \kappa \models_{\mathsf{RM}} 0 : \psi$$

$$\text{(Par)}\ \frac{\rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \psi\ \wedge\ \rho, \kappa \models_{\mathsf{RM}} \mathcal{Q} : \psi}{\rho, \kappa \models_{\mathsf{RM}} \mathcal{P} \mid \mathcal{Q} : \psi} \qquad \text{(Res)}\ \frac{\rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \psi}{\rho, \kappa \models_{\mathsf{RM}} (\nu [n]_s)\mathcal{P} : \psi}$$

The rule for *output* does two things: first, all the expressions are abstractly evaluated and then it is required that all the combinations of the values found

by this evaluation are recorded in κ. Finally, the continuation process must be analysed, which is also the case for *input* and *decryption* rules.

The rule for *input* incorporates pattern matching, which is dealt with by first abstractly evaluating all the of first j expressions in the input to be the sets ϑ_i for $i = 1, \ldots, j$. Next, if any of the sequences of length k in κ are such that the first j values component-wise are included in ϑ_i then the match is considered to be successful. In this case, the remaining values of the k-tuple must be recorded in ρ as possible bindings of the variables.

The rule for *decryption* handles the matching similarly to the rule for *input*. The only difference is that here the matching is performed also on the key. We use the faithful test for matching because the semantics ignores the annotations. After the successful matching, values are bound to the corresponding variables and, more importantly, the session identifiers of the key and of the first j components have to be checked equivalent. In case for some i, $\mathcal{I}(v_i) \neq \mathcal{I}(\mathcal{E}_i)$, meaning that not all the values are from the current session, the label of the decryption l is recorded in the error component ψ.

The rule for *replication* attaches two different session identifiers to two copies of the process before analysing both of them. Again the newly generated session identifier has to be unique in order not to mix processes up. We prove in Theorem 2 that it is enough to only analyse two copies of the process. For an informal argument: a replay attack is about replaying messages from a sessions to a principal not participating in the session and the control flow analysis treats sequential sessions and parallel session in the same way, analysing more than two sessions are not giving more information about attacks.

The rules for the inactive process, parallel composition and restriction are straightforward.

3.3 Semantic Properties

In this section, we shall show a list of lemmas and theorems concerning the semantics correctness. The detail proofs are omitted due to space limitations.

Our analysis respects the operational semantics of extended LySA. More precisely, we prove a subject reduction result for both the standard and the reference monitor semantics: if $\rho, \kappa \models \mathcal{P} : \psi$, then the same triple (ρ, κ, ψ) is a valid estimate for all the states passed through in a computation of $\mathcal{P}$. Additionally, we show that when the ψ component is empty, then the reference monitor is useless.

It is convenient to prove the following lemmata. The first states that estimates are resistant to substitution of closed terms for variables, and it holds for both extended terms and processes. The second lemma says that an estimate for an extended processes $\mathcal{P}$ is valid for every process congruent to $\mathcal{P}$, as well.

Lemma 1. ***(Substitution)***

1. $\rho \models \mathcal{E} : \vartheta$ *and* $\mathcal{E}' \in \rho(x)$ *imply* $\rho \models \mathcal{E}[\mathcal{E}'/x] : \vartheta$
2. $\rho, \kappa \models P : \psi$ *and* $\mathcal{E} \in \rho(x)$ *imply* $\rho, \kappa \models P[\mathcal{E}/x] : \psi$

Proof. The proofs proceed by structural induction over terms.

Lemma 2. ***(Congruence)***
If $P \equiv Q$ *and* $\rho, \kappa \models \mathcal{T}([P]_s) : \psi$ *then* $\rho, \kappa \models \mathcal{T}([Q]_s) : \psi$

Proof. By a straightforward inspection of each of the clauses defining $P \equiv Q$.

Subject reduction result holds for both the standard and the reference monitor semantics: if $\rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \psi$, then the same triple (ρ, κ, ψ) is a valid estimate for all the derivatives of $\mathcal{P}$.

Theorem 1. ***(Subject reduction)***

1. *If* $\mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{Q}$ *and* $\rho, \kappa \models \mathcal{P} : \psi$ *then also* $\rho, \kappa \models \mathcal{Q} : \psi$*;*
2. *Furthermore, if* $\psi = \emptyset$ *then* $\mathcal{P} \rightarrow_{\mathsf{RM}} \mathcal{Q}$

Proof. The proof is done by induction of the inference of $\mathcal{P} \rightarrow_{\mathcal{R}} \mathcal{Q}$.

The next result shows that our analysis correctly predicts when we can safely dispense with the reference monitor. We shall say that the reference monitor RM *cannot abort* a process $\mathcal{P}$ when there exist no $\mathcal{Q}, \mathcal{Q}'$ such that $\mathcal{P} \rightarrow_{\mathcal{R}}^{*} \mathcal{Q} \rightarrow_{\mathsf{RM}} \mathcal{Q}'$ and $\mathcal{P} \rightarrow_{\mathsf{RM}}^{*} \mathcal{Q} \not\rightarrow_{\mathsf{RM}}$. As usual, * stands for the transitive and reflexive closure of the relation in question, and $\mathcal{Q} \not\rightarrow_{\mathsf{RM}}$ stands for $\nexists \mathcal{Q}' : \mathcal{Q} \rightarrow_{\mathsf{RM}} \mathcal{Q}'$.

Theorem 2. ***(Static check for reference monitor)***

If $\rho, \kappa \models \mathcal{P} : \emptyset$ *then RM cannot abort* $\mathcal{P}$*.*

Proof Suppose *per absurdum* that such $\mathcal{Q}$ and $\mathcal{Q}'$ exist. A straightforward induction extends the subject reduction result to $\mathcal{P} \rightarrow^{*} \mathcal{Q}$ giving $\rho, \kappa \models_{\mathsf{RM}} \mathcal{Q} : \emptyset$. Theorem 1 part 2 of applied to $\mathcal{Q} \rightarrow \mathcal{Q}'$ gives $\mathcal{Q} \rightarrow_{\mathsf{RM}} \mathcal{Q}'$ which is a contradiction.

3.4 Example

The least solution of the analysis of the WMF protocol and has a non-empty ψ-component, i.e.

$$\rho, \kappa \models_{\mathsf{RM}} WMF : \psi$$

where ρ, κ and ψ have the following entries

$$\begin{aligned}
\rho : \; & y \mapsto \{\{[A]_0, [K_{ab}]_0\}_{[K_B]_0}, \{[A]_1, [K_{ab}]_1\}_{[K_B]_1}\} \\
& z \mapsto \{\{[Msg]_0\}_{[K_{ab}]_0}, \{[Msg]_1\}_{[K_{ab}]_1}\} \\
& p \mapsto \{\{[B]_0, [K_{ab}]_0\}_{[K_A]_0}, \{[B]_1, [K_{ab}]_1\}_{[K_A]_1}\} \\
& k \mapsto \{[K_{ab}]_0, [K_{ab}]_1\} \\
& k' \mapsto \{[K_{ab}]_0, [K_{ab}]_1\} \\
& z_m \mapsto \{[Msg]_0, [Msg]_1\}
\end{aligned}$$

$$\begin{aligned}
\kappa : \; & \{\langle [A]_0, [S]_0, [\{[B]_0, [K_{ab}]_0\}_{[K_A]_0}]_0\rangle, \langle [A]_1, [S]_1, [\{[B]_1, [K_{ab}]_1\}_{[K_A]_1}]_1\rangle\} \cup \\
& \{\langle [B]_0, [A]_0, [\{[Msg]_0\}_{[K_{ab}]_0}]_0\rangle, \langle [B]_1, [A]_1, [\{[Msg]_1\}_{[K_{ab}]_1}]_1\rangle\} \cup \\
& \{\langle [S]_0, [B]_0, [\{[A]_0, [K_{ab}]_0\}_{[K_B]_0}]_0\rangle, \langle [S]_1, [B]_1, [\{[A]_1, [K_{ab}]_1\}_{[K_B]_1}]_1\rangle\}
\end{aligned}$$

$$\psi : \{l1, l2, l3\}$$

According the rule for $[!P]_s$ in Table shown before, the analysis makes two copies of P with different session identifiers (0 and 1 in our case), which models two sessions running together.

The messages from both sessions are sent over the network, which the attacker has the total control of. Therefore, the attacher can fool a principal to accept a message actually coming from another session. This is suggested by the non-empty ψ: the three variables in ψ indicate that messages in step $1''$, $2''$ and $3''$ may not be fresh. This is highly dangerous because the principal may be forced to use an old session to encrypt the security data and in case of an old session key is revealed, confidentiality is not preserved any longer. A possible attack derivable from the solution above is shown below, where M represents the attacker:

$$\begin{array}{rl} 1. & [A]_1 \rightarrow [S]_1 : \{[B]_1, [K_{ab}]_1\}_{[K_A]_1} \\ 2. & [S]_1 \rightarrow M : \{[A]_1, [K_{ab}]_1\}_{[K_B]_1} \\ & M \rightarrow [B]_1 : \{[A]_0, [K_{ab}]_0\}_{[K_B]_0} \\ 3. & [B]_1 \rightarrow [A]_1 : \{[Msg]_1\}_{[K_{ab}]_0} \end{array}$$

4 Modelling the Attackers

In a protocol execution, several principals exchange messages over an open network, which is accessible to the attackers and therefore vulnerable to malicious behaviour. We assume an active Dolev-Yao attacker [11]. It is active in the sense that it is not only able to eavesdrop, but also to replay, encrypt, decrypt or generate messages providing that the necessary information is within his knowledge.

This scenario can be modelled in extended LySA as an attacker process running in parallel with the protocol process. Formally, we shall have $\mathcal{P}_{sys} \mid \mathcal{Q}$, where $\mathcal{P}_{sys}$ represents the protocol process and $\mathcal{Q}$ is some arbitrary attacker. The attacker acquires its knowledge by interacting with $\mathcal{P}_{sys}$, starting from the public knowledge. Note that the secret messages and keys, e.g. K_{ab}, are restricted to their scope in $\mathcal{P}_{sys}$ and thus they are not immediately accessible to the attacker.

4.1 Constructing Attacker Process

Our aim consists in finding a general way of constructing the attacker process, which is able to characterise all the attackers. The idea here is to define a formula, inspired by the work [3, 4], and then to prove its correctness.

In order for the attacker process to interact with the protocol process, some basic information of the protocol process has to be known in advance. We shall say that a process $\mathcal{P}_{sys}$ has the type $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}_\mathsf{Enc})$ whenever: (1) it is close, (2) all the free names of $\mathcal{P}_{sys}$ are in $\mathcal{N}_\mathsf{f}$, (3) all the arities used for sending or receiving are in $\mathcal{A}_\kappa$ and (4) all the arities used for encryption or decryption are in $\mathcal{A}_\mathsf{Enc}$. Obviously, $\mathcal{N}_\mathsf{f}$, $\mathcal{A}_\kappa$ and $\mathcal{A}_\mathsf{Enc}$ are all finite and can be computed by inspecting the process $\mathcal{P}_{sys}$.

One concern regarding the attacker process is about the names and variables it uses, which have to be apart from the ones used by $\mathcal{P}_{sys}$. Let all the names

used by $\mathcal{P}_{sys}$ to be in a finite set $\mathcal{N}_{\mathsf{c}}$, all the variables in a finite set $\mathcal{X}_{\mathsf{c}}$ and all the session identifiers in a finite set $\mathcal{S}_{\mathsf{c}}$; we can then postulate a new extended name $[n_\bullet]_{s_\bullet}$, where $n_\bullet$ is not in $\mathcal{N}_{\mathsf{c}}$, a new variable $z_\bullet$ not in $\mathcal{X}_{\mathsf{c}}$, and a new session identifier $s_\bullet$ not in $\mathcal{S}_{\mathsf{c}}$.

In order to control the number of names and variables used by the attacker, we construct a semantically equivalent process $\overline{\mathcal{Q}'}$, for a process $\mathcal{Q}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$, as follows: 1) all restrictions $(\nu[n]_s)\mathcal{P}$ are α-converted into restrictions $(\nu[n']_{s_\bullet})\mathcal{P}$ where n' has the canonical representative $n_\bullet$, 2) all the occurrences of variables x_i in $(\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k).\mathcal{P}$ and of variables x_i in $\mathsf{decrypt}\ \mathcal{E}\ \mathsf{as}\ \{\mathcal{E}_1, \ldots, \mathcal{E}_j; x_{j+1}, \ldots, x_k\}\ \mathsf{in}\ \mathcal{P}$ are α-converted to use variables x'_i with canonical representative $z_\bullet$. Therefore $\overline{\mathcal{Q}'}$ only has finitely many canonical names and variables.

(1) $\wedge_{k \in \mathcal{A}_\kappa}\ \forall\, \langle v_1, \ldots, v_k \rangle \in \kappa :\ \ \wedge_{i=1}^{k}\, v_i \in \rho(z_\bullet)$ the attacker may learn by eavesdropping
(2) $\wedge_{k \in \mathcal{A}_{\mathsf{Enc}}}\ \forall [\{v_1, \ldots, v_k\}_{v_0}]_s \in \rho(z_\bullet) :$ $v_0 \propto \rho(z_\bullet) \Rightarrow \wedge_{i=1}^{k} v_i \in \rho(z_\bullet)$ the attacker may learn by decrypting messages with keys already known
(3) $\wedge_{k \in \mathcal{A}_{\mathsf{Enc}}}\ \forall\, v_0, \ldots, v_k : \wedge_{i=0}^{k} v_i \in \rho(z_\bullet) \Rightarrow [\{v_1, \ldots, v_k\}_{v_0}]_{s_\bullet} \in \rho(z_\bullet)$ the attacker may construct new encryptions using the keys known
(4) $\wedge_{k \in \mathcal{A}_\kappa}\ \forall v_1, \ldots, v_k : \wedge_{i=1}^{k} v_i \in \rho(z_\bullet) \Rightarrow \langle v_1, \ldots, v_k \rangle \in \kappa$ the attacker may actively forge new communications
(5) $\{[n_\bullet]_{s_\bullet}\} \cup \mathcal{N}_f \subseteq \rho(z_\bullet)$ the attacker initially has some knowledge

We now have sufficient control over the capabilities of the attacker. Now, we extend the standard Dolev-Yao threat model with session identifiers. We express the extended Dolev-Yao condition for our LySA calculus and define a formula $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$ as the conjunction of the five components in Table shown above, where each line describes an ability of the attacker. Furthermore, we claim that the formula $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ is capable of characterising the potential effect of all attackers $\overline{\mathcal{Q}}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$.

The soundness of our Dolev-Yao condition is established by the following Theorem.

Theorem 3. ***(Correctness of the extended Dolev-Yao condition)***

If (ρ, κ) satisfies $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$ then there exists ψ such that for all attackers $\mathcal{Q}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$ $\rho, \kappa \models_{\mathsf{RM}} \overline{\mathcal{Q}} : \psi$

Proof. The proof is done by structural induction on $\overline{\mathcal{Q}}$.

5 Main Results

The session identifiers in the extended LySA are designed to make the capture of replay attacks easier, thus ensuring that the receiving messages are fresh. For the

dynamic property, we say that $\mathcal{P}_{sys}$ guarantees *dynamic freshness* with respect to the annotations in $\mathcal{P}_{sys}$ if the reference monitor RM cannot abort $\mathcal{P}_{sys} \mid \overline{\mathcal{Q}}$ regardless of the choice of the attacker $\mathcal{Q}$.

Similarly, for static property we say that $\mathcal{P}_{sys}$ guarantees *static freshness* with respect to the annotations in $\mathcal{P}_{sys}$ if there exists ρ and κ such that $\rho, \kappa \models_{\mathsf{RM}} \mathcal{P} : \emptyset$ and (ρ, κ) satisfies $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$.

Theorem 4. *If $\mathcal{P}$ guarantees static freshness then $\mathcal{P}$ guarantees dynamic freshness.*

Proof. If $\rho, \kappa \models_{\mathsf{RM}} \mathcal{P}_{sys} : \emptyset$ and (ρ, κ) satisfies $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ then, by Theorems 2 and 3, RM does not abort $\mathcal{P}_{sys} \mid \overline{\mathcal{Q}}$ regardless of the choice of attacker $\mathcal{Q}$.

5.1 Implementation and Complexity

To obtain an implementation we transform the analysis into a logically equivalent formation written in Alternation-free Least Fixed Point logic (ALFP) [12], and use the Succinct Solver [12], which computes the least interpretation of the predicate symbols in a given ALFP formula. The time complexity of solving a formula in the Succinct Solver is polynomial in the size of the universe, over which the formula is interpreted. For our implementation the universe is linear in the size of the process and a simple worst-case estimate of the degree of the complexity polynomial is given as one plus the maximal nesting depth of quantifiers in the formula. For our current implementation the nesting depth is governed by the maximal length of the sequences used in the communication and encryption. In practice, the implementation runs in sub-cubic time and we obtain running times well in few seconds for all of our experiments.

5.2 Validation of Needham-Schroeder Symmetric Key Protocol

Needham-Schroeder Symmetric Key Protocol is a classical protocol and has been used widely as an example for protocol verification. The protocol has 6 steps: in the first steps, a fresh session key K is generated by the trusted server S and sent to both parties, A and B; in the following two steps, B sends out a challenge to make sure A is in possession of the new session key. After a protocol run, A and B share a secret session key for secure communication. The protocol narration is listed below in the left,

1. $A \rightarrow S : A, B, N_a$	1. $A \rightarrow S : \quad A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K, \{K, A\}_{K_b}\}_{K_a}$	2. $S \rightarrow A : \quad \{N_a, B, K, \{K, A\}_{K_b}\}_{K_a}$
3. $A \rightarrow B : \{A, K\}_{K_b}$	3. $M(A) \rightarrow B : \{A, K'\}_{K_b}$
4. $B \rightarrow A : \{N_b\}_K$	4. $B \rightarrow M(A) : \{N_b\}_{K'}$
5. $A \rightarrow B : \{N_b - 1\}_K$	5. $M(A) \rightarrow B : \{N_b - 1\}_{K'}$
6. $A \rightarrow B : \{Msg\}_K$	6. $M(A) \rightarrow B : \{Msg\}_{K'}$
the protocol narration	*a replay attack scenario*

The analysis result of Needhan-Schroeder Symmetric Key Protocol shows a violation, meaning that it is subject to a replay attack. This result corresponds

to the replay attack reported by Denning & Sacco in [10]: the message in step 3 can be replayed with an old compromised session key by an active attacker and consequently B is forced to use the old key K' for communication. An example trace is shown above in the right.

To fix this problem, Denning & Sacco and Needham & Schroeder proposed different solutions but both make use of new nonces. Needham & Schroeder's solution is: having A ask B for another random value N'_a to be sent to the Server for return in $\{A, N'_a, K\}_{K_b}$. After the correction, the first three steps become the followings and others keep unchanged.

$$\begin{array}{ll} 1.\ A \rightarrow S: & A, B, N_a, N'_a \\ 2.\ S \rightarrow A: & \{N_a, B, K, \{A, N'_a, K\}_{K_b}\}_{K_a} \\ 3.\ M(A) \rightarrow B: & \{A, N'_a, K\}_{K_b} \end{array}$$

After applying the analysis to the above version, the result becomes: ***no violations possible***, i.e. $\psi = \emptyset$, meaning that the attacker now cannot replay the message from step 3 and therefore no replay attack is possible to this corrected version.

6 Conclusion

In this paper we have introduced a sound way to detect replay attacks at static time. To do that, we extended the standard LySA calculus with session identifiers and gave it a reduction semantics. The semantics ensures session identifiers are properly treated along the evolution of a process. On the static side, we extended the control flow analysis [3, 4] to verify the freshness property of the extended processes. The static property ensures that, if the secret information received by a principal is in the right context, then a process is not subject to a run-external attack at execution time. As far as the attacker is concerned, we adopted the notion from Dolev-Yao threat model and extended it with session identifiers in order to fit it into our setting. The extended Dolev-Yao attacker is able to monitor the traffic over the network and actively generate messages within his knowledge. We implemented the analysis and used our tool to check some significant protocols, including classical protocols, e.g. Wide Mouthed Frog, Yahalom, Andrew Secure RPC, Otway-Rees, Needham-Schroeder, Amended Needham-Schroeder. Besides the classical protocols, at present, we are successfully applying our analysis to other kinds of protocols, like the ones in the family of IEEE 802.16 [17]. The tool confirmed that we can successfully detect potential replay attacks on the protocols.

The original LySA calculus and the control flow analysis [3, 4] are designed to validate authentication property of security protocols. In this paper, they are extended systematically such that we are able to address an orthogonal property, freshness. The way we validate freshness is inspired by BAN logic [7], which is actually a set of rules for defining and analysing security protocols, namely "if one part of a formula is known to be fresh, then the entire formula must also be fresh". We also prove that analysing two copies of a process in our framework is

sufficient for capturing run-external replay attacks. The experiments conducted also confirmed this. The literature already has similar results, e.g. Comon & Cortier [8] and Millen [19].

Several papers deal with replay attacks and freshness. Because of lack of space, we only mention the closest to ours, i.e. [14–16] and [6], where the approach is based on type (and effects) systems that statically guarantee entity authentication of protocols. Gordon and Jeffrey [14–16] defined type (and effects) systems that statically guarantee authentication of protocols specified in a Spi-calculus enriched with assertions *à la* Woo-Lam. In [6], Bugliesi, Focardi, Maffei still use a type and effect system, but use a different technique and a different calculus (the ρ-spi calculus).

The analysis presented in this paper is part of a project, analysing various security properties of communication protocols using annotations. It can be easily combined with other kinds of annotations from the same framework, e.g. the one from [13] for confidentiality, and the one from [5] for simple type flaw attacks, and hence gives a more comprehensive analysis result.

References

1. Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation, 148(1)*, pp. 1-70, 1999.
2. B Blanchet. An efficient cryptographic protocol verifier based on prolog rules. *IEEE Computer Society Press*, 2001.
3. Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson and Hanne Riis Nielson. Automatic Valication of Protocol Narration. *In Proceeding of Computer Security Foundations Workshop*, IEEE Press, pp. 126-140, 2003.
4. Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson and Hanne Riis Nielson. Static Validation of Security Protocols. *Journal of Computer Security*, 13(3), pp.347 - 390, 2005.
5. Chiara Bodei, Pierpaolo Degano, Han Gao and Linda Brodo. Detecting and Preventing Type flaws: a Contro Flow Analysis with tags. *In Proceeding of 5th International Workshop on Security Issues in Concurrency*. ENTCS series. To appear.
6. Michele Bugliesi, Riccardo Focardi, Matteo Maffei. Authenticity by Tagging and Typing. *In Proceeding of 2nd ACM Workshop on Formal Methods in Security Engineering*, 2004, ACM Press.
7. Michael Burrows and Martín Abadi and Roger Needham. A Logic of Authentication, ACM. *Transactions in Computer Systems*, 8(1), pp. 18-36, 1990.
8. Hubert Comon-Lundh and Veronique Cortier. Tree automata with one memory set constraints and cryptographic protocols. *Theoretical Computer Science*. 331(1): 143-214, Elsevier, 2005.
9. Michele Curti, Pierpaolo Degano and Cosima Tatiana Baldari. Causal π-Calculus for Biochemical Modelling. *In Proceeding of Workshop on Computational Methods in Systems Biology, LNCS 2602*, pp. 21-33, 2003.
10. Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8): 533-536, 1981.
11. Danney Dolev and Andrew C. Yao. On the Security of Public Key Protocols. IEEE TIT, IT-29(12):198-208, 1983.

12. Flemming Nielson, Helmut Seidl, and Hanne Riis Nielson. A Succinct Solver for ALFP. *Nordic Journal of Computing*, 9:335-372, 2002.
13. Han Gao and Hanne Riis Nielson. Analysis of LYSA-calculus with explicit confidentiality annotations. *In Proceeding of Advanced Information Networking and Applications*, IEEE Computer Society, 2006.
14. Andrew D. Gordon and Alan Jeffrey. Authenticity by Typing for Security Protocols. In *In Proceeding of Computer Security Foundations Symposium*. IEEE, 2001.
15. Andrew D. Gordon. Typing Correspondence Assertions for Communication Protocols. In *In Proceeding of Mathematical Foundations of Programming Semantics*, 2001.
16. Andrew D. Gordon and Alan Jeffrey. Types and Effects for Asymmetric Cryptographic Protocols. In *In Proceeding of Computer Security Foundations Symposium*. IEEE, 2002.
17. IEEE Std 802.16e-2005, 2006. Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1, IEEE, New York, USA.
18. Catherine Meadows, Paul Syverson, and Iliano Cervesato. Formal Specification and Analysis of the Group Domain of Interpretation Protocol Using NPATRL and the NRL Protocol Analyzer. *Journal of Computer Security. 12(6)*, pp. 893-931, 2004.
19. Jonathan K. Millen. Term Replacement Algebra for the Interrogator. The MITRE Corporation, MP 97B65, 1997.
20. Robin Milner. Communicating and mobile systems: the π-calculus. Cambridge University Press, 1999.
21. Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM, 21(12)*, Dec 1978.
22. Dave Otway and Owen Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review, 21(1)*, pp.8-10, 1987.
23. Lawrence C. Paulson. Inductive Analysis of the Internet Protocol TLS. *ACM Transactions on Computer and System Security, 2(3): 332-351*, 1999.
24. Lawrence C Paulson. The foundation of a generic theorem prover. Automated Reasoning 5 (1989), pp. 363-397.
25. Paul Syverson. A Taxonomy of Replay attacks. *In Proceeding of Computer Security Foundations Symposium*, IEEE Computer Society Press, 1994.