Non-Repudiation Analysis with LYSA [*]

Mayla Brusò and Agostino Cortesi

Abstract This work introduces a formal analysis of the non-repudiation property for security protocols. Protocols are modelled in the process calculus LYSA, using an extended syntax with annotations. Non-repudiation is verified using a Control Flow Analysis, following the same approach of M. Buchholtz and H. Gao for authentication and freshness analyses.
The result is an analysis that can statically check the protocols to predict if they are secure during their execution and which can be fully automated.

1 Introduction

With the growth of Internet applications like e-shopping or e-voting, non-repudiation is becoming increasingly important, as a protocol property. Our aim is to provide a protocol analysis which checks this property to avoid that a protocol is used in malicious way. Among the existing techniques that perform the analysis of non-repudiation protocols, we may cite:

- The CSP (Communicating Sequential Processes) approach [12], [13]: it is an abstract language designed specifically for the description of communication patterns of concurrent system components that interact through message passing.
- The game approach [10]: it considers the execution of the protocol as a game, where each entity is a player; the protocols are designed finding a strategy, which has to defend an honest entity against all the possible strategies of malicious parties.

[*] Work partially supported by PRIN 07 MUR Project 200793N42R SOFT.

Mayla Brusò
Computer Science Department, Ca' Foscari University, Italy, e-mail: mabruso@dsi.unive.it

Agostino Cortesi
Computer Science Department, Ca' Foscari University, Italy, e-mail: acortesi@dsi.unive.it

- The Zhou-Gollmann approach [16]: it uses *SVO Logic*, a modal logic that is composed by inference rules and axioms which are used to express beliefs that can be analysed by a judge to decide if the service provided the property.
- The inductive approach [1]: it uses an inductive model, a set of all the possible histories of the network that the protocol execution may produce; a history, called *trace*, is a list of network events, that can indicate the communication of a message or the annotation of information for future use.

We follow a different approach, the same as M. Buchholtz [5] and H. Gao [8], who show how some security properties can be analysed using the LYSA [2] process calculus with annotations and a Control Flow Analysis (CFA) to detect flaws in the protocols. The main idea is to extend LYSA with specific annotations, i.e. tags that identify part of the message for which the property has to hold and that uniquely assign principal and session identifiers to encryptions and decryptions.

It is interesting to notice that the non-repudiation analysis that we propose easily fits into the CFA framework [11], yielding a suite of analyses that can be combined in various ways, with no major implementation overload.

The main differences between our proposal and the previously cited alternative approaches are the following: our analysis can check many protocols and can model scenarios with infinitely many principals while other approaches often are developed to analyse only a particular protocol and can model scenarios with finite principals.

The structure of the paper is the following: Section 2 is a quick overview of LYSA; Section 3 presents the CFA framework; Section 4 shows the new non-repudiation analysis, and its application to the protocols; Section 5 concludes.

2 LYSA

LYSA [2] is a process calculus in the π-calculus tradition that models security protocols on a global network. It incorporates pattern matching into the language constructs where values can become bound to variables. In LYSA all the communications take place directly on a global network and this corresponds to the scenario in which security protocols often operate, where channels are not considered.

2.1 Syntax and Semantics

An expression $E \in Expr$ may represent a name, a variable or an encryption. The set $Expr$ contains two disjointed subsets, $Name$ ranged over by n, which contains identifiers, nonces, keys, etc., and Var ranged over by x, which contains variables. The remaining expressions are symmetric and asymmetric encryptions of k-tuples of other expressions, defined as $\{E_1, \ldots, E_k\}_{E_0}$ and $\{| E_1, \ldots, E_k |\}_{E_0}$ respectively, where E_0 represents a symmetric or asymmetric key.

LySa also allows to construct processes $P \in Proc$, which use the expressions explained above. Processes can have the following form:

- $\langle E_1, \dots, E_k \rangle.P$: the process sends a k-tuple of values onto the global network; if the message reaches its destination, the process continues as P.
- $(E_1, \dots, E_j; x_{j+1}, \dots, x_k).P$: the process read a message and, if $E_1, \dots, E_j$ are identical to the values expected, the remaining $k-j$ values are bound to the variables $x_{j+1}, \dots, x_k$, and the process continues as P.
- decrypt E as $\{E_1, \dots, E_j; x_{j+1}, \dots, x_k\}_{E_0}$ in P: the process denotes the symmetric decryption and, if the encryption key is identical to E_0, then the process decrypts the k-tuple; if $E_1, \dots, E_j$ are identical to the values expected, the remaining $k-j$ values are bound to the variables $x_{j+1}, \dots, x_k$, and the process continues as P.
- decrypt E as $\{|E_1, \dots, E_j; x_{j+1}, \dots, x_k|\}_{E_0}$ in P: the process denotes the asymmetric decryption and it works like symmetric decryption except that E_0 and the key used to encrypt have to be a key pair m^+ and m^-.
- $(\nu\, n)P$: the process generates a new name n and it continues in P.
- $(\nu \pm m)P$: the process generates a new key pair, m^+ / m^-, and it continues in P.
- $P_1 \mid P_2$: the process denotes two processes running in parallel.
- $!P$: the process acts as an arbitrary number of processes P composed in parallel.
- 0: the process is the inactive or nil process that does nothing.

A binder introduces new names or variables which have scope in the rest of the process. Restriction, input and decryption constructs are binders of names, key pairs, and variables, which have scope in the subprocess P. Names and variables are called free whenever they are not bound by any binder; the functions $fn(P)$ and $fv(P)$ collect all the free names and variables in the process P, respectively. The bound variables are defined by the function $bv(P) \overset{def}{=} var(P) \setminus fv(P)$. All these functions are also defined on the terms.

Example 1. Let us see how to encode in LySa the protocol defined by Cederquist, Corin and Dashti in [6].

$$\begin{array}{lll}
\mathrm{A} \rightarrow \mathrm{B}: & \{M\}_K, EOO_M & \text{for } EOO_M = sig_A(B, TTP, H, \{|\, K, A\,|\}_{TTP}) \\
\mathrm{B} \rightarrow \mathrm{A}: & EOR_M & \text{for } EOR_M = sig_B(EOO_M) \\
\mathrm{A} \rightarrow \mathrm{B}: & K & \\
\mathrm{B} \rightarrow \mathrm{A}: & EOR_K & \text{for } EOR_K = sig_B(A, H, K)
\end{array}$$

where $H = h(\{M\}_K)$ and h is a hash function. The encoding is the following:

$$\begin{array}{ll}
\multicolumn{2}{l}{\text{let } X \subseteq S \text{ in } (\nu_{\pm i \in X}\, AK_i)(\nu \pm\, TTP)(} \\
\quad |_{i \in X} |_{j \in X} & !(\nu\, SK_{ij})(\nu\, H_{ij})(\nu\, M_{ij}) \\
 & \langle \{M_{ij}\}_{SK_{ij}}, \{|I_j, TTP, H_{ij}, \{|SK_{ij}, I_i|\}|\} \rangle.(; xEORM_{ij}). \\
 & \text{decrypt } xEORM_{ij} \text{ as } \{|\{|I_j, TTP, H_{ij}, \{|SK_{ij}, I_i|\}|\}|\} \text{ in} \\
 & \langle SK_{ij} \rangle.(; xEORK_{ij}).\text{decrypt } xEORK_{ij} \text{ as } \{|I_i, H_{ij}, SK_{ij};\} \text{ in } 0 \\
\quad |_{i \in X} |_{j \in X} & !(; xEnMsg_{ij}, xEOOM_{ij}). \\
 & \text{decrypt } xEOOM_{ij} \text{ as } \{|I_j, TTP; xH_{ij}, xTTP|\} \text{ in} \\
 & \langle \{|xEOOM_{ij}|\} \rangle.(; xSK_{ij}). \\
 & \text{decrypt } xEnMsg_{ij} \text{ as } \{xMsg_{ij}\}_{xSK_{ij}} \text{ in } \langle \{|I_i, xH_{ij}, xSK_{ij}|\} \rangle.0)
\end{array}$$

LySa provides a reduction semantics that describes the evolution of a process step-by-step, using a *reduction relation* between two processes, written $P \rightarrow P'$. If the reduction relation holds then P can evolve in P' using the rules depicted in Table 1.

The structural congruence between two processes, written $P \equiv P'$, means that P is equal to P' except for syntactic aspects, but this does not interfere with the way they evolve. We refer to [2] [4] for a detailed description of the semantics. Notice that a substitution $P[n_1 \mapsto n_2]$ substitutes all the free occurrences of n_1 in P for n_2. Finally, we define values $V \in Val$, which are used in the reduction as expressions without variables $x \in Var$:

$$V ::= n \mid m^+ \mid m^- \mid \{V_1, \ldots, V_k\}_{V_0} \mid \{|V_1, \ldots, V_k|\}_{V_0} \tag{1}$$

A *reference monitor* is used to force additional requirements at each step before allowing it to be executed. A *substitution function* is used in the reduction rules, written $P[V/x]$, to substitute a variable x for a value V in the process P.

Table 1 Semantics of LySa calculus

(Com)	$\dfrac{\bigwedge_{i=1}^{j} V_i = V_i'}{\langle V_1,\ldots,V_k\rangle.P \mid (V_1',\ldots,V_j';x_{j+1},\ldots,x_k).P' \rightarrow_{\mathscr{R}} P \mid P'[V_{j+1}/x_{j+1},\ldots,V_k/x_k]}$		
(Dec)	$\dfrac{\bigwedge_{i=0}^{j} V_i = V_i'}{\text{decrypt } \{V_1,\ldots,V_k\}_{V_0} \text{ as } \{V_1',\ldots,V_j';x_{j+1},\ldots,x_k\}_{V_0'} \text{ in } P \rightarrow_{\mathscr{R}} P[V_{j+1}/x_{j+1},\ldots,V_k/x_k]}$		
(ADec)	$\dfrac{\bigwedge_{i=1}^{j} V_i = V_i'}{\text{decrypt } \{|V_1,\ldots,V_k|\}_{m^+} \text{ as } \{|V_1',\ldots,V_j';x_{j+1},\ldots,x_k|\}_{m^-} \text{ in } P \rightarrow_{\mathscr{R}} P[V_{j+1}/x_{j+1},\ldots,V_k/x_k]}$		
(ASig)	$\dfrac{\bigwedge_{i=1}^{j} V_i = V_i'}{\text{decrypt } \{|V_1,\ldots,V_k|\}_{m^-} \text{ as } \{|V_1',\ldots,V_j';x_{j+1},\ldots,x_k|\}_{m^+} \text{ in } P \rightarrow_{\mathscr{R}} P[V_{j+1}/x_{j+1},\ldots,V_k/x_k]}$		
(New)	$\dfrac{P \rightarrow_{\mathscr{R}} P'}{(\nu\, n)P \rightarrow_{\mathscr{R}} (\nu\, n)P'}$	(ANew)	$\dfrac{P \rightarrow_{\mathscr{R}} P'}{(\nu \pm m)P \rightarrow_{\mathscr{R}} (\nu \pm m)P'}$
(Par)	$\dfrac{P_1 \rightarrow_{\mathscr{R}} P_1'}{P_1 \mid P_2 \rightarrow_{\mathscr{R}} P_1' \mid P_2}$	(Congr)	$\dfrac{P \equiv P' \wedge P' \rightarrow_{\mathscr{R}} P'' \wedge P'' \equiv P'''}{P \rightarrow_{\mathscr{R}} P'''}$

2.2 Meta Level Calculus

The meta level describes different scenarios in which many principals execute a protocol at the same time, simply running several copies of the processes. The syntax of the meta level is identical to the syntax seen so far, except that each name and each variable are renamed using indexes. Four new processes are introduced to model these scenarios, which use a countable indexing set S to include a set of variables X and $\bar{i}$, as shorthand for $i_1, \ldots, i_k$ (a sequence of indexes); the processes are the following:

- $|_{i \in S}\, MP$: the process describes the parallel composition of instances of the process MP where the index i is an element in the set S.
- let $X \subseteq S$ in MP: the process declares a set identifier X which has some values of the index set S in the process MP; the set X can be infinite.

- $(\nu_{\bar{i}\in\bar{S}}\ n_{\overline{ai}})MP$, $(\nu_{\pm\bar{i}\in\bar{S}}\ m_{\overline{ai}})MP$: the processes describe the restriction of all the names $n_{\overline{ai}}$ and all the key pairs $m^{+}_{\overline{ai}}$ and $m^{-}_{\overline{ai}}$ respectively; $\overline{a}$ is a prefix of the index that can be empty.

In this syntax, the process let $X \subseteq S$ in MP is a binder of X, while the process $|_{i\in S}$ MP is a binder of i and the indexed restrictions are binders of names and key pairs.

An instantiation relation, written $MP \rightarrow_{\mathscr{I}} P$, is introduced to describe that a process P is an instance of a meta level process MP, as depicted in Table 2.

Table 2 Instantiation relation $MP \rightarrow_{\mathscr{I}} P$

(ILet)	$\dfrac{MP[X\mapsto S']\rightarrow_{\mathscr{I}} P}{\text{let } X\subseteq S \text{ in } MP\rightarrow_{\mathscr{I}} P}$ if $S' \subseteq_{fin} S$		
(IIPar)	$\dfrac{MP[i\mapsto a_1]\rightarrow_{\mathscr{I}} P_1 \ldots MP[i\mapsto a_k]\rightarrow_{\mathscr{I}} P_k}{\|_{i\in\{a_1,\ldots,a_k\}} MP\rightarrow_{\mathscr{I}} P_1\|\ldots\|P_k}$		
(IINew)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{(\nu_{\bar{i}\in\{\overline{a_1},\ldots,\overline{a_k}\}}\ n_{\overline{ai}})MP\rightarrow_{\mathscr{I}} (\nu\ n_{\overline{aa_1}})\ldots(\nu\ n_{\overline{aa_k}})P}$		
(IIANew)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{(\nu_{\pm\bar{i}\in\{\overline{a_1},\ldots,\overline{a_k}\}}\ m_{\overline{ai}})MP\rightarrow_{\mathscr{I}} (\nu\pm\ m_{\overline{aa_1}})\ldots(\nu\ m_{\overline{aa_k}})P}$		
(IOut)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{\langle ME_1,\ldots,ME_k\rangle.MP\rightarrow_{\mathscr{I}} \langle ME_1,\ldots,ME_k\rangle.P}$		
(IInp)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{(ME_1,\ldots,ME_j;mx_{j+1},\ldots,mx_k).MP\rightarrow_{\mathscr{I}} (ME_1,\ldots,ME_j;mx_{j+1},\ldots,mx_k).P}$		
(IDec)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{\text{decrypt } ME \text{ as } \{ME_1,\ldots,ME_j;mx_{j+1},\ldots,mx_k\}_{ME_0} \text{ in } MP\rightarrow_{\mathscr{I}}}$ decrypt ME as $\{ME_1,\ldots,ME_j;mx_{j+1},\ldots,mx_k\}_{ME_0}$ in P		
(IADec)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{\text{decrypt } ME \text{ as } \{\|ME_1,\ldots,ME_j;mx_{j+1},\ldots,mx_k\|\}_{ME_0} \text{ in } MP\rightarrow_{\mathscr{I}}}$ decrypt ME as $\{\|\ ME_1,\ldots,ME_j;mx_{j+1},\ldots,mx_k\ \|\}_{ME_0}$ in P		
(INew)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{(\nu\ n_{\overline{a}})MP\rightarrow_{\mathscr{I}} (\nu\ n_{\overline{a}})P}$	(IANew)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{(\nu\pm m_{\overline{a}})MP\rightarrow_{\mathscr{I}} (\nu\pm m_{\overline{a}})P}$
(IRep)	$\dfrac{MP\rightarrow_{\mathscr{I}} P}{!MP\rightarrow_{\mathscr{I}} !P}$	(IPar)	$\dfrac{MP_1\rightarrow_{\mathscr{I}} P_1 \quad MP_2\rightarrow_{\mathscr{I}} P_2}{MP_1\|MP_2\rightarrow_{\mathscr{I}} P_1\|P_2}$
(INil)	$0 \rightarrow_{\mathscr{I}} 0$		

3 Control Flow Analysis

In this section we introduce our Control Flow Analysis (CFA) as an extension of [11]. The aim of the CFA is to collect information about the behavior of a process and to store them in some data structures $\mathscr{A}$, called analysis components. To be finite, static analysis is forced to compute approximations rather than exact answers. Therefore the analysis can give false positives but it has to preserve soundness.

We will use Flow Logic settings [11][3] for the specifications and the proofs. It is a formalism for specifying static analysis and it focuses on the relationship between an analysis estimate and the process to be analysed, formally $\mathscr{A} \models P$.

CFA abstracts the executions and represents only some aspects of the behavior of a process which can also be infinite. We will prove the correctness of the analysis by showing that the analysis components $\mathscr{A}$ are such that the property they represent also holds when the process evolves.

Formally:

$$\mathscr{A} \models P \wedge P \rightarrow P' \Rightarrow \mathscr{A} \models P' \tag{2}$$

The Flow Logic specifications use the verbose format "$\mathscr{A} \models P$ iff a logic formula $\mathscr{F}$ holds" or the succinct format "$\mathscr{A} \models P : A'$ iff a logic formula $\mathscr{F}$ holds", i.e. they record information about a process globally or locally, respectively.

The analysis components record canonical values from the set $\lfloor Val \rfloor$ ranged over by U to represent values generated by the same restriction. The component $\kappa \in \mathscr{P}(\lfloor Val \rfloor^*)$ collects the tuples of canonical values corresponding to the values communicated in the global network while $\rho : \lfloor Var \rfloor \rightarrow \mathscr{P}(\lfloor Val \rfloor)$ records the canonical values corresponding to the values that variables may become bound. A predicate $\rho, \kappa \models P$ says that ρ and κ are valid analysis results describing the behavior of P. To analyse the expressions it is used the form $\rho \models E : \vartheta$ to describe a set of canonical values $\vartheta \in \mathscr{P}(\lfloor Val \rfloor)$ that the expression E may evaluate.

The analysis of terms and processes is described in Table 3. The rules (AN), (ANp) and (ANm) say that names may evaluate to themselves iff the canonical names are in ϑ. The rule (AVar) says that variables may evaluate to the values described by ρ for the corresponding canonical variable. The rules (AEnc) and (AAEnc) use the analysis predicate recursively to evaluate all the subexpressions in the encryption and they require ϑ to contain all the encrypted values that can be formed combining the values that subexpressions may evaluate to. The rule (AOut) says that the expressions are evaluated and it is required that all the combinations of the values found by this evaluation are recorded in κ. The rule (AInp) says that the first j expressions in the input construct are evaluated to be the sets ϑ_i for $i = 1, \ldots, j$; if the pattern match with the values in κ is successful, the remaining values of the k-tuple is recorded in ρ as possible binding of the variables and the continuation process is analysed. The rule (ASDec), (AADec) and (AASig) evaluate the expression E into the set ϑ and the first j expressions in the decryption constructs are evaluated to be the sets ϑ_i for $i = 1, \ldots, j$; if the pattern match with the values in κ is successful, the remaining values of the k-tuple is recorded in ρ as possible binding of the variables and the continuation process is analysed. Notice that the original syntax [5] [8] uses only the rule (AADec) to define both asymmetric decryption and signature while we introduce here two rules imposing an order in the choice of the keys to make our analysis more efficient. The rule (ANew), (AANew), (APar) and (ARep) require that the subprocesses are analysed. The rule (ANil) deals with the trivial case.

Whenever the requirements hold, the continuation process is analysed.

The analysis is also defined for the meta level as an extension of the analysis seen so far and it takes the form

$$\rho, \kappa \models_{\Gamma} M \tag{3}$$

where $\Gamma : SetID \cup \mathscr{P}(Index_{fin}) \rightarrow \mathscr{P}(Index_{fin})$ is a mapping from set identifiers to finite sets of indexes.

The meta level analysis is defined in Table 4 for the new constructs. The rule (MLet) updates Γ with the mapping $X \mapsto S'$, where S' is required to be finite and it has the same canonical names as the set S. The rule (MIPar) expresses that the analysis holds for all the processes where the index i is substituted by all the elements in $\Gamma(S)$. The rules (MINew) and (MIANew) ignore the restriction operators.

Table 3 Analysis of terms and processes

(AN)	$\rho \models n : \vartheta$ iff $\lfloor n \rfloor \in \vartheta$	(ANp)	$\rho \models m^+ : \vartheta$ iff $\lfloor m^+ \rfloor \in \vartheta$
(ANm)	$\rho \models m^- : \vartheta$ iff $\lfloor m^- \rfloor \in \vartheta$	(AVar)	$\rho \models x : \vartheta$ iff $\rho(\lfloor x \rfloor) \subseteq \vartheta$
(AEnc)	$\rho \models \{E_1,\dots,E_k\}_{E_0} : \vartheta$ iff $\bigwedge_{i=0}^{k} \rho \models E_i : \vartheta_i \wedge \forall U_0,\dots,U_k : \bigwedge_{i=0}^{k} U_i \in \vartheta_i \Rightarrow \{U_1,\dots,U_k\}_{U_0} \in \vartheta$		
(AAEnc)	$\rho \models \{\lvert E_1,\dots,E_k \rvert\}_{E_0} : \vartheta$ iff $\bigwedge_{i=0}^{k} \rho \models E_i : \vartheta_i \wedge \forall U_0,\dots,U_k : \bigwedge_{i=0}^{k} U_i \in \vartheta_i \Rightarrow \{\lvert U_1,\dots,U_k \rvert\}_{U_0} \in \vartheta$		
(AOut)	$\rho, \kappa \models \langle E_1,\dots,E_k \rangle.P$ iff $\bigwedge_{i=1}^{k} \rho \models E_i : \vartheta_i \wedge \forall U_1,\dots,U_k : \bigwedge_{i=1}^{k} U_i \in \vartheta_i \Rightarrow (\langle U_1,\dots,U_k \rangle \in \kappa \wedge \rho, \kappa \models P)$		
(AInp)	$\rho, \kappa \models (E_1,\dots,E_j; x_{j+1},\dots,x_k).P$ iff $\bigwedge_{i=1}^{j} \rho \models E_i : \vartheta_i \wedge \forall \langle U_1,\dots,U_k \rangle \in \kappa : \bigwedge_{i=1}^{j} U_i \in \vartheta_i \Rightarrow \left(\bigwedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa \models P\right)$		
(ASDec)	$\rho, \kappa \models$ decrypt E as $\{E_1,\dots,E_j; x_{j+1},\dots,x_k\}_{E_0}$ in P iff $\rho \models E : \vartheta \wedge \bigwedge_{i=0}^{j} \rho \models E_i : \vartheta_i \wedge \forall \{U_1,\dots,U_k\}_{U_0} \in \vartheta \wedge \bigwedge_{i=0}^{j} U_i \in \vartheta_i \Rightarrow (\bigwedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa \models P)$		
(AADec)	$\rho, \kappa \models$ decrypt E as $\{\lvert E_1,\dots,E_j; x_{j+1},\dots,x_k \rvert\}_{E_0}$ in P iff $\rho \models E : \vartheta \wedge \bigwedge_{i=0}^{j} \rho \models E_i : \vartheta_i \wedge \forall \{\lvert U_1,\dots,U_k \rvert\}_{U_0} \in \vartheta : \forall U_0' \in \vartheta_0 : \forall (m^+, m^-) : (U_0, U_0') = (\lfloor m^- \rfloor, \lfloor m^+ \rfloor) \wedge \bigwedge_{i=1}^{j} U_i \in \vartheta_i \Rightarrow \left(\bigwedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa \models P\right)$		
(AASig)	$\rho, \kappa \models$ decrypt E as $\{\lvert E_1,\dots,E_j; x_{j+1},\dots,x_k \rvert\}_{E_0}$ in P iff $\rho \models E : \vartheta \wedge \bigwedge_{i=0}^{j} \rho \models E_i : \vartheta_i \wedge \forall \{\lvert U_1,\dots,U_k \rvert\}_{U_0} \in \vartheta : \forall U_0' \in \vartheta_0 : \forall (m^+, m^-) : (U_0, U_0') = (\lfloor m^+ \rfloor, \lfloor m^- \rfloor) \wedge \bigwedge_{i=1}^{j} U_i \in \vartheta_i \Rightarrow \left(\bigwedge_{i=j+1}^{k} U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa \models P\right)$		
(ANew)	$\rho, \kappa \models (\nu\, n)P$ iff $\rho, \kappa \models P$	(AANew)	$\rho, \kappa \models (\nu \pm m)P$ iff $\rho, \kappa \models P$
(APar)	$\rho, \kappa \models P_1 \vert P_2$ iff $\rho, \kappa \models P_1 \wedge \rho, \kappa \models P_2$	(ARep)	$\rho, \kappa \models !P$ iff $\rho, \kappa \models P$
(ANil)	$\rho, \kappa \models 0$ iff $true$		

Table 4 The meta level analysis $\rho, \kappa \models_\Gamma M$: meta level constructs

(MLet)	$\rho, \kappa \models_\Gamma$ let $X \subseteq S$ in M	iff $\rho, \kappa \models_{\Gamma[X \mapsto S']} M$ where $S' \subseteq_{fin} \Gamma(S)$ and $\lfloor S' \rfloor = \lfloor \Gamma(S) \rfloor$
(MIPar)	$\rho, \kappa \models_\Gamma \vert_{i \in S}\, M$	iff $\bigwedge_{a \in \Gamma(S)} \rho, \kappa \models_\Gamma M[i \mapsto a]$
(MINew)	$\rho, \kappa \models_\Gamma (\nu_{\bar{i} \in \overline{S}} n_{\overline{ai}})M$	iff $\rho, \kappa \models_\Gamma M$
(MIANew)	$\rho, \kappa \models_\Gamma (\nu_{\pm \bar{i} \in \overline{S}} m_{\overline{ai}})M$	iff $\rho, \kappa \models_\Gamma M$

3.1 The attacker

The attacker is unique and runs its protocol $P_\bullet$ following the Dolev-Yao formula $\mathscr{F}_{RM}^{DY}$ [7] shown in Table 5, which explains its powers. We write $P_{sys} \mid P_\bullet$ to show that an arbitrary attacker controls the whole network while principals exchange messages using the protocol. A protocol process P_{sys} has type whenever it is close, all its free names are in $\mathscr{N}_f$, all the arities of the sent or received messages are in $\mathscr{A}_\kappa$ and all the arities of the encrypted or decrypted messages are in $\mathscr{A}_{Enc}$. These three sets are finite, like $\mathscr{N}_c$ and $\mathscr{X}_c$, used to collect all the names and all the variables respectively in the process P_{sys}. The attacker uses a new name, $n_\bullet \notin \mathscr{N}_c$, and a new variable, $z_\bullet \notin \mathscr{X}_c$, which do not overlap the names and the variables used by the legitimate principals. It is again considered a process with finitely many canonical names and variables. A formula $\mathscr{F}_{RM}^{DY}$ of the type $(\mathscr{N}_f, \mathscr{A}_\kappa, \mathscr{A}_{Enc})$, which is capable of characterizing the potential effect of all the attackers $P_\bullet$ of the type $(\mathscr{N}_f, \mathscr{A}_\kappa, \mathscr{A}_{Enc})$, is defined as the conjunction of the components in Table 5.

Table 5 The attacker's capabilities

(1)	The attacker may learn by eavesdropping
	$\bigwedge_{k\in\mathscr{A}_\kappa} \forall\langle V_1,\dots,V_k\rangle \in \kappa : \bigwedge_{i=1}^{k} V_i \in \rho(z_\bullet)$
(2)	The attacker may learn by decrypting messages with keys already known
	$\bigwedge_{k\in\mathscr{A}_{Enc}} \forall\{V_1,\dots,V_k\}_{V_0} \in \rho(z_\bullet) : V_0 \in \rho(z_\bullet) \Rightarrow \bigwedge_{i=1}^{k} V_i \in \rho(z_\bullet)$
	$\bigwedge_{k\in\mathscr{A}_{Enc}} \forall\{\mid V_1,\dots,V_k \mid\}_{m^+} \in \rho(z_\bullet) : m^- \in \rho(z_\bullet) \Rightarrow \bigwedge_{i=1}^{k} V_i \in \rho(z_\bullet)$
	$\bigwedge_{k\in\mathscr{A}_{Enc}} \forall\{\mid V_1,\dots,V_k \mid\}_{m^-} \in \rho(z_\bullet) : m^+ \in \rho(z_\bullet) \Rightarrow \bigwedge_{i=1}^{k} V_i \in \rho(z_\bullet)$
(3)	The attacker may construct new encryptions using the keys known
	$\bigwedge_{k\in\mathscr{A}_{Enc}} \forall V_0,\dots,V_k : \bigwedge_{i=0}^{k} V_i \in \rho(z_\bullet) \Rightarrow \{V_1,\dots,V_k\}_{V_0} \in \rho(z_\bullet)$
	$\bigwedge_{k\in\mathscr{A}_{Enc}} \forall m^+, V_1,\dots,V_k : m^+ \in \rho(z_\bullet) \wedge \bigwedge_{i=1}^{k} V_i \in \rho(z_\bullet) \Rightarrow \{\mid V_1,\dots,V_k \mid\}_{m^+} \in \rho(z_\bullet)$
	$\bigwedge_{k\in\mathscr{A}_{Enc}} \forall m^-, V_1,\dots,V_k : m^- \in \rho(z_\bullet) \wedge \bigwedge_{i=1}^{k} V_i \in \rho(z_\bullet) \Rightarrow \{\mid V_1,\dots,V_k \mid\}_{m^-} \in \rho(z_\bullet)$
(4)	The attacker may actively forge new communications
	$\bigwedge_{k\in\mathscr{A}_\kappa} \forall V_1,\dots,V_k : \bigwedge_{i=1}^{k} V_i \in \rho(z_\bullet) \Rightarrow \langle V_1,\dots,V_k\rangle \in \kappa$
(5)	The attacker initially has some knowledge
	$\{n_\bullet, m_\bullet^\pm\} \cup \mathscr{N}_f \subseteq \rho(z_\bullet)$

4 Non-Repudiation Analysis

Non-repudiation guarantees that the principals exchanging messages cannot falsely deny having sent or received the messages. This is done using evidences [9] that allow to decide unquestionably in favor of the fair principal whenever there is a dispute. In particular, non-repudiation of origin provides the recipient with proof of origin while non-repudiation of receipt provides the originator with proof of receipt. Evidences [15] should have verifiable origin, integrity and validity.

The syntax of the process calculus LYSA has to be extended to guarantee, given a protocol, the non-repudiation property, i.e. authentication (only the sender of the message can create it), integrity and freshness. This is done using electronic signatures and unique identifiers for users and sessions. To this aim, we introduce two sets, used in the body of the messages to collect information that will be useful to perform the analysis: ID, where $id \in ID$ is a unique identifier for a principal, and NR, where $nr \in NR$ says that non-repudiation property is required for that part of the message *nr*. To include these sets in our analysis, a redefinition of the syntax is required, and this is done by applying a function called $\mathscr{G}$ to the processes of the protocol analysed, that acts recursively on the subprocesses and redefines subterms using another function, called $\mathscr{F}$ (see Table 6). In the new syntax *id*s are attached whenever an asymmetric key appears and a session identifier u is attached to each encryption and decryption; parallel composition and replication are modified to assign different *id*s to different processes. The rule $!P \equiv P \mid !P$ has to be removed because the structural equivalence does not hold in this case. The replication process evolves in $\mathscr{G}(P,id) \mid \mathscr{G}(!P,id')$, where id' is a unique user identifier by the replication rule. Finally, we have to add the following annotations to the signatures:

Table 6 Functions $\mathscr{F}$ and $\mathscr{G}$

$\mathscr{F} : E \times ID \to \varepsilon$	
- $\mathscr{F}(n,id) = n$	- $\mathscr{F}(x,id) = x$
- $\mathscr{F}(m^+,id) = [m^+]_{id}$	- $\mathscr{F}(m^-,id) = [m^-]_{id}$
- $\mathscr{F}(\{E_1,\dots,E_k\}_{E_0},id) = \{\mathscr{F}(E_1,id),\dots,\mathscr{F}(E_k,id)\}_{\mathscr{F}(E_0,id)}$	
- $\mathscr{F}(\{\!\mid E_1,\dots,E_k \mid\!\}_{E_0},id) = \{\!\mid \mathscr{F}(E_1,id),\dots,\mathscr{F}(E_k,id) \mid\!\}^u_{\mathscr{F}(E_0,id)}$	
$\mathscr{G} : P \times ID \to \mathscr{P}$	
- $\mathscr{G}(\langle E_1,\dots,E_k\rangle.\mathscr{P},id) = \langle \mathscr{F}(E_1,id),\dots,\mathscr{F}(E_k,id)\rangle.\mathscr{G}(P,id)$	
- $\mathscr{G}((E_1,\dots,E_j;x_{j+1},\dots,x_k).P,id) = (\mathscr{F}(E_1,id),\dots,\mathscr{F}(E_j,id);x_{j+1},\dots,x_k).\mathscr{G}(P,id)$	
- $\mathscr{G}(\text{decrypt } E \text{ as } \{E_1,\dots,E_j;x_{j+1},\dots,x_k\}_{E_0} \text{ in } P,id) =$	
decrypt $\mathscr{F}(E,id)$ as $\{\mathscr{F}(E_1,id),\dots,\mathscr{F}(E_j,id);x_{j+1},\dots,x_k\}_{\mathscr{F}(E_0,id)}$ in $\mathscr{G}(P,id)$	
- $\mathscr{G}(\text{decrypt } E \text{ as } \{\!\mid E_1,\dots,E_j;x_{j+1},\dots,x_k \mid\!\}^u_{E_0} \text{ in } P,id) =$	
decrypt $\mathscr{F}(E,id)$ as $\{\!\mid \mathscr{F}(E_1,id),\dots,\mathscr{F}(E_j,id);x_{j+1},\dots,x_k \mid\!\}^u_{\mathscr{F}(E_0,id)}$ in $\mathscr{G}(P,id)$	
- $\mathscr{G}((\nu\, n)P,id) = (\nu\, n)\mathscr{G}(P,id)$	- $\mathscr{G}((\nu \pm m)P,id) = (\nu \pm [m]_{id})\mathscr{G}(P,id)$
- $\mathscr{G}(P \mid Q,id) = \mathscr{G}(P,id) \mid \mathscr{G}(Q,id')$	- $\mathscr{G}(!P,id) = [!P]_{id}$
- $\mathscr{G}(0,id) = 0$	

- [from *id*] is associated to encryption and it means that the recipient expects a message from *id*.
- [check *NR*] is associated to decryption and it means that for all the elements of the set *NR*, non-repudiation property must be guaranteed. It is interesting to notice that the elements in the set *NR* can specify a part of the message, not necessarily the whole message, according to the definition of non-repudiation.

The syntax of asymmetric encryption becomes $\{|\varepsilon_1,\dots,\varepsilon_k|\}^u_{\varepsilon_0}$[from *id*] while the syntax of asymmetric decryption becomes decrypt ε as $\{|\varepsilon_1,\dots,\varepsilon_j;x_{j+1},\dots,x_k|\}^u_{\varepsilon_0}$ [check *NR*] in $\mathscr{P}$.

Notice that the annotation [from *id*] and the label u have a different role in the analysis. The first says that the principal who encrypted the message must be the same specified in the label associated to the private key used, while the second expresses that the message has to belong to a precise session.

To guarantee the dynamic property, the values have to be redefined into *NVal*:

$$NV ::= n \mid [m^+]_{id} \mid [m^-]_{id} \mid \{NV_1,\dots,NV_k\}_{NV_0} \mid \{|NV_1,\dots,NV_k|\}^u_{NV_0}[\text{from } id] \qquad (4)$$

The reference monitor semantics $P \rightarrow_{RM} P'$ defines RM as

$$RM(id, id', u, u', \{NV_1, \dots, NV_n\}, NR) \\ = (id = id' \wedge u = u' \wedge \forall nr \in NR : nr \in \{NV_1, \dots, NV_n\})$$

where $\{NV_1, \dots, NV_n\}$ is a set of redefined values for non-repudiation analysis. The main difference between the standard and the redefined semantics is expressed by the rule used to verify a signature, which ensures that the non-repudiation property holds for the elements specified by the annotations:

$$\frac{\bigwedge_{i=1}^{j} NV_i = NV_i' \wedge RM(id, id', u, u', \{NV_{j+1}, \dots, NV_k\}, NR)}{\begin{array}{c}\text{decrypt } \{|NV_1, \dots, NV_k|\}^{u}_{[m^-]_{id}} [\text{from } id'] \text{ as } \{|NV_1', \dots, NV_j'; x_{j+1}, \dots, x_k|\}^{u'}_{[m^+]_{id}} \\ [\text{check } NR] \text{ in } \mathscr{P} \rightarrow_{RM} \mathscr{P}[NV_{j+1}/x_{j+1}, \dots, NV_k/x_k]\end{array}}$$

Definition 1 (Dynamic Non-Repudiation). A process $\mathscr{P}$ ensures *dynamic non-repudiation* property if for all the executions $\mathscr{P} \rightarrow^* \mathscr{P}' \rightarrow_{RM} \mathscr{P}''$ then $id = id'$ and $u = u'$ and $\forall nr \in NR : nr \in \{NV_1, \dots, NV_k\}$ when $\mathscr{P}' \rightarrow_{RM} \mathscr{P}''$ is derived using (ASig) on the asymmetric decryption construct.

Definition 1 says that an extended process $\mathscr{P}$ ensures non-repudiation property if there is no violation in any of its execution.

4.1 Static Property

A component $\psi \subseteq \mathscr{P}(NR)$ will collect all the labels nr such that the non-repudiation property for the element nr is possibly violated. The $\propto$ operator is introduced to ignore the extension of the syntax. The non-repudiation property has to be checked whenever a signature is verified, therefore the rule (ASig) becomes the following:

$$\begin{array}{l}\rho, \kappa, \psi \models \text{decrypt } \varepsilon \text{ as } \{|\, \varepsilon_1, \dots, \varepsilon_j; x_{j+1}, \dots, x_k \,|\}^{u'}_{\varepsilon_0} [\text{check } NR] \text{ in } \mathscr{P} \\ \text{iff } \rho \models \varepsilon : \vartheta \wedge \bigwedge_{i=0}^{j} \rho \models \varepsilon_i : \vartheta_i \wedge \forall \{|\, NV_1, \dots, NV_k \,|\}^{u}_{NV_0} [\text{from } \lfloor id \rfloor] \propto \vartheta : \\ \forall NV_0' \propto \vartheta_0 : \forall m^+, m^-, id, id' : (NV_0, NV_0') = ([\lfloor m^+ \rfloor]_{id'}, [\lfloor m^- \rfloor]_{id'}) \\ \wedge \bigwedge_{i=1}^{j} NV_i \propto \vartheta_i \Rightarrow (\bigwedge_{i=j+1}^{k} NV_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models \mathscr{P} \wedge \forall nr \in NR : \\ (id \neq id' \vee u \neq u' \vee nr \notin \{NV_{j+1}, \dots, NV_k\}) \Rightarrow \lfloor nr \rfloor \in \psi).\end{array}$$

To prove the correctness of our analysis we must prove that it respects the extended operational semantics of LYSA, i.e. if $\rho, \kappa, \psi \models \mathscr{P}$ then the triple (ρ, κ, ψ) is a valid estimate for all the states passed through in a computation of $\mathscr{P}$. Furthermore, we prove that when ψ is empty, then the reference monitor is useless.

Theorem 1 (Correctness of the non-repudiation analysis). *If* $\rho, \kappa, \psi \models \mathscr{P}$ *and* $\psi = \emptyset$ *then* $\mathscr{P}$ *ensures* static non-repudiation.

The proof of this theorem, as well as the proof of the next ones, can be found in [4].

4.2 The attacker

In the setup of $\mathscr{P} \mid \mathscr{P}_\bullet$, the attacker process $\mathscr{P}_\bullet$ has to be annotated with the extended syntax. We will use a unique label $u_\bullet$ to indicate the session and a unique label $id_\bullet$ to indicate the encryption place used by the attacker. The Dolev-Yao condition has to be redefined to be used for the non-repudiation analysis.

The main enhancement with the usual LySa attacker can be seen in rule (3.): whenever the attacker is able to generate an encrypted message with a known key, the receiver checks the *id* of the sender, and, in case the latter does not correspond to the intended one, the component ψ becomes non empty, as a signal of a non-repudiation violation:

$$\begin{array}{l}
\bigwedge_{k \in \mathscr{A}_{Enc}} \forall [m^-]_{id}, NV_1, \ldots, NV_k : [m^-]_{id} \in \rho(z_\bullet) \wedge \bigwedge_{i=1}^{k} NV_i \in \rho(z_\bullet) \\
\Rightarrow \{|NV_1, \ldots, NV_k|\}^{u_\bullet}_{[m^-]_{id_\bullet}} \in \rho(z_\bullet) \wedge \forall \text{ decrypt } \{|NV'_1, \ldots, NV'_k|\}^{u_\bullet}_{[m^-]_{id_\bullet}} \text{[from } id'] \text{ as} \\
\{|NV''_1, \ldots, NV''_j; x_{j+1}, \ldots, x_k|\}^{u''}_{[m^+]_{id''}} \text{[check } NR] \text{ in } \mathscr{P} : \\
\forall nr \in NR\, ((id' \neq id_\bullet \vee u'' \neq u_\bullet \vee nr \notin \{NV'_{j+1}, \ldots, NV'_k\}) \Rightarrow \lfloor nr \rfloor \in \psi)
\end{array}$$

Theorem 2 (Correctness of Dolev-Yao Condition). *If (ρ, κ, ψ) satisfies $\mathscr{F}^{DY}_{RM}$ of type $(\mathscr{N}_\mathrm{f}, \mathscr{A}_\kappa, \mathscr{A}_{Enc})$ then $\rho, \kappa, \psi \models \overline{Q}$ for all attackers Q of extended type $(\{z_\bullet\}, \mathscr{N}_\mathrm{f} \cup \{n_\bullet\}, \mathscr{A}_\kappa, \mathscr{A}_{Enc})$.*

The theorem says that the redefined Dolev-Yao condition holds.

Theorem 3. *If $\mathscr{P}$ guarantees static non-repudiation then $\mathscr{P}$ guarantees dynamic non-repudiation.*

Example 2. **Protocol 1.** The encoding with annotations of the protocol by Cederquist, Corin, and Dashti introduced in Example 1 becomes:

$$\begin{array}{ll}
\multicolumn{2}{l}{\text{let } X \subseteq S \text{ in } (\nu_{\pm i \in X}\ [AK_i]_{I_i})(\nu \pm\ TTP)(} \\
\mid_{i \in X} \mid_{j \in X} & !(\nu\ SK_{ij})(\nu\ H_{ij})(\nu\ M_{ij})\langle \{M_{ij}\}_{SK_{ij}}, \{|EOO_M|\}^{u_{ij}}_{[AK_i^-]_{I_i}} \text{[from } I_i]\rangle. \\
& (;xEORM_{ij}).\text{decrypt } xEORM_{ij} \text{ as } \{|\{|EOO_M|\}^{u_{ij}}_{[AK_i^-]_{I_i}} \text{[from } I_i]|\}^{u_{ij}}_{[AK_j^+]_{I_j}} \\
& \text{[check } \{|EOO_M|\}] \text{ in } \langle SK_{ij}\rangle. \\
& (;xEORK_{ij}).\text{decrypt } xEORK_{ij} \text{ as } \{|I_i, H_{ij}, SK_{ij};\}^{u_{ij}}_{[AK_j^+]_{I_j}} \text{[check } H_{ij}, SK_{ij}] \text{ in } 0 \\
\mid_{i \in X} \mid_{j \in X} & !(;xEnMsg_{ij}, xEOOM_{ij}). \\
& \text{decrypt } xEOOM_{ij} \text{ as } \{|I_j, TTP; xH_{ij}, xTTP|\}^{u_{ij}}_{[AK_i^+]_{I_i}} \text{[check } xH_{ij}] \text{ in} \\
& \langle \{|xEOOM_{ij}|\}^{u_{ij}}_{[AK_j^-]_{I_j}} \text{[from } I_j]\rangle.(;xSK_{ij}). \\
& \text{decrypt } xEnMsg_{ij} \text{ as } \{xMsg_{ij}\}_{xSK_{ij}} \text{ in } \langle \{|I_i, xH_{ij}, xSK_{ij}|\}\rangle.0)
\end{array}$$

where $EOO_M = I_j, TTP, H_{ij}, \{|SK_{ij}, I_i|\}^{u_{ij}}_{[TTP^+]_{TTP}}$ [from $\emptyset$]
Protocol 2 (Zhou-Gollmann [14]):

A → B :	f_{NRO}, B, L, C, NRO
B → A :	f_{NRR}, A, L, NRR
A → TTP :	f_{SUB}, B, L, K, sub_K
B ↔ TTP :	$f_{CON}, A, B, L, K, con\_K$
A ↔ TTP :	$f_{CON}, A, B, L, K, con\_K$

The result of the analysis of Protocol 1 shows that a possible flaw may arise. In fact, it does not use labels to identify the session, and this is why our analysis says that this protocol does not guarantee non-repudiation property. However the protocol is correct, because of an implicit additional assumption on the uniqueness of the keys, which prevents from replay attacks. On the other side, Protocol 2 passes the analysis and this guarantees that it is secure with respect to non-repudiation.

5 Conclusions and Future Works

This paper extends the work by M. Buchholtz and H. Gao who defined a suite of analyses for security protocols, namely authentication, confidentiality, freshness, simple and complex type flaws. The annotations we introduce allow to express non-repudiation also for part of the message: this allow to tune the analysis focussing on relevant components. It results that the CFA framework developed for the process calculus LYSA can be extended to security properties by identifying suitable annotations, thus re-using most of the theoretical work.

References

1. Bella, G., Paulson, L.C.: Mechanical proofs about a non-repudiation protocol. In: TPHOL01, volume 2152 of LNCS, pp. 91–104. Springer (2001)
2. Bodei, C., Buchholtz, M., Degano, P., Nielson, F., Nielson, H.R.: Static validation of security protocols. In: Journal of Computer Security, pp. 347–390 (2005)
3. Braghin, C., Cortesi, A., Focardi, R.: Information flow security in boundary ambients. Inf. Comput. **206**(2-4), 460–489 (2008)
4. Brusò, M., Cortesi, A.: Non-repudiation analysis using LYSA with annotations, *CS Tech. Report, Univ. Ca' Foscari*. Tech. rep. (2008). URL http://www.unive.it/nqcontent.cfm?a_id=5144#rapporti08
5. Buchholtz, M., Lyngby, K.: Automated analysis of security in networking systems. ph. d. thesis proposal. available from http://www.imm.dtu.dk/ mib/thesis. Tech. rep. (2004)
6. Cederquist, Corin, Dashti: On the quest for impartiality: Design and analysis of a fair non-repudiation protocol. In: ICIS, LNCS (2005)
7. Dolev, D., Yao, A.C.: On the security of public key protocols. Tech. rep., Stanford, CA (1981)
8. Gao, H.: Analysis Of Protocols By Annotations. Ph. D. Thesis, Informatics and Mathematical Modelling, Technical University of Denmark (2008)
9. Gollmann, D.: Computer security. John Wiley & Sons, Inc., New York, NY, USA (1999)
10. Kremer, S., Raskin, J.F.: A game-based verification of non-repudiation and fair exchange protocols. In: Journal of Computer Security, pp. 551–565. Springer-Verlag (2001)
11. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag New York, LLC (1999)
12. Schneider, S.: Formal analysis of a non-repudiation protocol. In: 11th IEEE Computer Security Foundations Workshop, p. 54 (1998)
13. Schneider, S., Holloway, R.: Security properties and csp. In: IEEE Symp. Security and Privacy, pp. 174–187. IEEE Computer Society Press (1996)
14. Zhou, J., Gollmann, D.: A fair non-repudiation protocol. IEEE Computer Society Press (1996)
15. Zhou, J., Gollmann, D.: Evidence and non-repudiation. J. Netw. Comput. Appl. **20**(3), 267–281 (1997). DOI http://dx.doi.org/10.1006/jnca.1997.0056
16. Zhou, J., Gollmann, D.: Towards verification of non-repudiation protocols. In: Proceedings of International Refinement Workshop and Formal Methods Pacific. Springer-Verlag (1998)