

# Formal intruder identikit for open security protocols

A. Bracciali, A. Brogi, G. Ferrari, and E. Tuosto

Dipartimento di Informatica, University of Pisa, Italy  
{braccia, brogi, giangi, etuosto}@di.unipi.it

**Abstract.** A formal methodology for the verification of security protocols is presented. The proposed methodology includes a formal language for specifying protocols, a logic for expressing protocol properties, and an algorithm to determine whether a protocol satisfies certain properties. The verification process constructs the identikit of an actual malicious intruder when the protocol fails to satisfy the desired security property. A distinguishing aspect of the methodology is that protocol verification is related to successful termination.

## 1 Introduction

The constantly increasing number of networked applications, such as e-commerce applications, are giving security issues a prominent role in the (software) economy.

Considerable attention is being paid to *security protocols*, protocols that (should) ensure properties such as secrecy, authentication or integrity. Following [DY83a], we assume that a malicious *intruder* may intercept (virtually all) the communications exchanged by a set of protocol principals (viz., principals behaving according to the protocol). As shown for instance in [Low95], intruders with such an intercepting capability may break many security protocols, even under the perfect encryption hypothesis. Typically, a malicious intruder tries to increase its knowledge by intercepting communications in order to successfully attack the protocol. Moreover, more than one protocol session may be necessary for the intruder to break a protocol, as in the case for instance of the Needham-Schroeder protocol [NS78].

This paper aims at contributing to setting the foundations for the formal verification of security protocols. The proposed methodology can be summarised in three main steps:

- (1) Specification of a protocol  $\mathcal{P}$ ,
- (2) specification of a property  $\phi$  that  $\mathcal{P}$  should satisfy, and
- (3) verification that  $\mathcal{P}$  does (or does not) satisfy  $\phi$ .

A formal language for expressing security protocols is introduced first. The language is a subset of  $\pi$ -calculus, extended with cryptographic primitives in the style of spi-calculus [AG99]. Protocol principals are specified by means of (finite) behaviour expressions containing synchronous communication operations as well

as symmetric and asymmetric encryptions. A set of (instances of) protocol principals forms a context, which may evolve both because of internal interactions and because new (instances of) principals join the context. Contexts hence provide a natural representation of multiple sessions of a protocol, as we will discuss further later on.

A logic for expressing protocol properties is then introduced. In spite of its simplicity, the logic permits to express non trivial properties on the data that are exchanged according to a protocol. The idea of specifying security properties in terms of expected values of exchanged data was recently proposed in [AG99] via the so-called notion of *magic instance* of a protocol. Our logic generalises the notion of magic instance by permitting the specification of arbitrary relations among data and principals. To illustrate the expressiveness of the logic, we will show its usage to specify the authentication property in the well-known Needham-Schroeder protocol [NS78].

We finally present a formal methodology for verifying whether a property  $\phi$  holds for a protocol specification  $\mathcal{P}$ . The verification process inputs both  $\mathcal{P}$  and  $\phi$ , and it may be additionally fed by specifying the initial (possibly empty) knowledge of the intruder and the maximum number of protocol sessions to be considered. The verification consists of determining whether there may exist a malicious intruder capable of joining a context of (instances of) protocol principals, invalidating the expected property  $\phi$ , and letting the entire context normally terminate its execution.

The main contributions of the paper may be summarised as follows:

- (1) *Reducing protocol correctness to termination.* Our verification technique relies on a state space exploration to determine whether there exists a point in the space corresponding to a successful attack. However, as we analyse the flow of information among the principals and the attacker, only a subset of the whole state space has semantical significance. Namely the relevant points of the state space are those where the information flow is stable (e.g., all keys have been successfully exchanged or a principal accepts a key as good for communicating with another principal, etc). In this paper we characterise stable states in terms of protocol termination.
- (2) *Construction of the intruder.* If the protocol does not satisfy the desired security protocol, the verification algorithm returns the “identikit” of the actual behaviour of a possible malicious intruder rather than a simple boolean answer (e.g., “secure/unsecure”). The availability of a description of the behaviour of an intruder is extremely valuable for protocol designers facing the need of refining parts of their protocol in order to fix the detected security leak.
- (3) *A unified approach to security and open composition.* The language used to specify security protocols was introduced in [BBT01] to describe and reason on open systems consisting of autonomous, interacting software components. On the one hand, this paper contributes to demonstrating the expressive power of the composition-oriented approach of [BBT01]. On the other hand,

the proposed methodology for verifying security protocols paves the way for the development of a more general methodology to enforce security in open component-based systems. The typical target application is the verification of the *acceptability* of an (untrusted) component  $C$  willing to join an open system consisting of a number of interacting components. The acceptance of  $C$  may be conditioned to the verification that its declared behaviour will not invalidate the specified security (or correctness) invariant. (In case of acceptance, the interaction of  $C$  with the rest of the system will be mediated by a wrapper in charge of dynamically checking that  $C$  is behaving according to its initial declaration.)

In the following, we will focus on the description of the overall methodology by abstracting from some technical details.

## 2 A calculus of interacting principals

Security protocols are usually expressed in an informal language which describes the communication happening among the principals of the protocol. For example, a sentence like

$$A \rightarrow B : \{n\}_k$$

intuitively means that in the protocol, principal  $A$  sends to principal  $B$  the datum  $n$  encrypted by the key  $k$ . More precisely, the sentence is actually intended to mean “in *any possible session* of the protocol, *an instance* of principal  $A$  (i.e., a principal behaving accordingly to the specification of  $A$ ) *intends to send* the datum  $n$  encrypted with  $k$  to *an instance* of principal  $B$ ”.

In this section we present a calculus to describe multiple runs of symmetric and asymmetric cryptographic protocols. Moreover, we will focus on finite and deterministic protocols. (While this constraint helps in developing an efficient verification process, it is satisfied by a wide class of protocols.) Following the Dolev-Yao model [DY83b], principals communication can be intercepted by any other principal in the environment and in particular by malicious ones, that we call intruders. We assume the perfect encryption hypothesis: Keys cannot be guessed or deduced (even if a large number of messages is collected).

### 2.1 Syntax

We start by introducing the syntax for defining protocol principals.

**Definition 1 (Principals).** *The set of principals  $P$  participating in a protocol is defined as:*

$$P ::= PN \hat{=} (\bar{X})[E]$$

where  $PN$  is a principal name (infinitely many constants ranged over by capital letters such as  $A, B, \dots$  or  $I$ , the name of the intruder),  $\bar{X}$  is the set of open

variables, and  $E$  is the closed behavioural expression of the principal defined according to the following syntax:

$$\begin{aligned} E &::= 0 \mid \alpha.E \mid E||E \mid E + E \\ \alpha &::= in(D) \mid out(D) \\ D &::= PN \mid X \mid ?X \mid K \mid PN^+ \mid PN^- \mid NO \mid \{D\}_D \mid (D, D). \end{aligned}$$

where  $X$  is the set of variables,  $K$  is the set of symmetric keys,  $PN^+$  and  $PN^-$  are the sets of asymmetric keys ( $A^+$  and  $A^-$  denote respectively the public and the private key of a principal  $A$ ), and  $NO$  is the set of nonces (“name-once”).

The set  $\bar{X}$  of open variables of  $A \triangleq (\bar{X})[E]$  explicitly declares the keys needed by  $A$  to communicate with other principals. Principals are connected one another by assigning values (viz., symmetric keys or principal names to be used as public keys) to their open variables.

A behavioural expression may consist of the inaction  $0$  or of a communication action  $\alpha$  prefixed (“.”) to a behavioural expression, or it may be made out of the parallel ( $||$ ) and non-deterministic choice composition ( $+$ ) operators.

Communication actions synchronously send or receive data  $D$ . The notation “?” is used to denote a *binding occurrence* of a variable inside a  $in()$  operation. A variable  $x$  occurs *bound* in a behavioural expression  $E$ ,  $x \in bv(E)$ , if it is under the scope of an input action binding  $x$  or it belongs to the open variables of  $E$ . A variable  $x$  is *free* in  $E$ ,  $x \in fv(E)$ , if there is an occurrence of  $x$  in  $E$  that is not bound. Notice that in the expression  $in(?y).in(y).0$  the second occurrence of  $y$  is bound by the first occurrence of  $y$ . This means that the same datum must be received twice, while the expression  $in(?y).in(?y).0$  may receive two different data, one for each binding occurrence of the variable  $y$ . Differently, variables occurring as encryption keys in input actions can not be binding occurrences. An expression  $E$  is closed if it does not contain free variables. Substitution, sometime written  $[v/x]$ , works as expected. Communication actions encapsulate encryption and decryption mechanisms: The attempt to receive an encrypted data by means of a given key  $k$  (e.g.,  $in(\{?x\}_k)$ ) will succeed only if the key  $k$  is the correct one. For this reason, expressions such as  $in(\{d\}_{?y})$  are not allowed.<sup>1</sup> On the other hand, it is always possible to receive an encrypted data “as is” (e.g., by  $in(?y)$ ), without attempting to decrypt it.

We assume that principals do not initially know the private key of any other principal. A principal  $A$  can learn the private key  $B^-$  of another principal  $B$  only if  $B$  will send such key to  $A$ . An open variable can not be used as a private key.

*Example 1.* Let us consider a simple protocol and its formalisation:

$$\begin{array}{l} (1) A \rightarrow B : \{m\}_k \quad A \triangleq (x)[out(\{m\}_x).in(\{?y\}_{A-})] \\ (2) B \rightarrow A : \{n\}_{A^+} \quad B \triangleq (z, w)[in(\{?v\}_z).out(\{n\}_{w+})] \end{array}$$

<sup>1</sup> Notice that the syntax of the language permits to write “wrong” terms, like  $in(\{k\}_{(m,n)})$  which represents the attempt to receive a key encrypted by means of a pair of names. These cases of wrong coding will resolve in run-time errors (deadlock) since no type discipline is provided.

$A$  sends  $B$  a nonce  $m$  encrypted with the symmetric key  $k$ . Then  $B$  sends  $A$  the nonce  $n$  encrypted with the public key of  $A$ . In the formalisation, principals  $A$  and  $B$  have the open variables  $x$  and  $z$ , respectively, to share the symmetric key  $k$ . The open variable  $w$  of  $B$  will let  $B$  know the name (and hence the public key) of  $A$ .

## 2.2 Multiple-sessions

The last step needed to model multiple runs of a protocol is a suitable mechanism to compose *instances* of the principals in the same environment, that we call *context*.

Principals interact by joining a context populated by possible partners. More instances of the same principal can access a context, so that more sessions of the protocol can be running in the context. This is modeled by means of a *join* operation, that takes care of connecting open variables of (renamed apart) instances of principals. Renaming is performed by indexing data of each instance with a different natural number, so that local names (of different instances of the same principal) are distinguished. Multiple sessions are modelled by suitably connected instances of protocols.

**Definition 2 (Instance and Context).** *Given a principal  $A \triangleq (\bar{X})[E]$  and a natural number  $i$ , an instance  $A_i$  of  $A$  is  $(\bar{X}_i)[E_i]$ , where all the data  $D$  occurring in  $\bar{X}$  or  $E$  are indexed with  $i$ . A context is a (possibly empty) set  $\{(\bar{X}_1)[E_1], \dots, (\bar{X}_n)[E_n]\}$  of instances.*

The *join* operation defines how a principal instance can enter a (running) context by connecting “asymmetric” open variables to principal names and “symmetric” open variables to key names so that they are appropriately shared. Connected variables are not open anymore.

**Definition 3 (Join).** *Let  $\mathcal{C} = \{(\bar{X}_1)[E_1], \dots, (\bar{X}_n)[E_n]\}$  be a context, and let  $(\bar{Y}_{n+1})[F_{n+1}]$  be a principal instance. Let  $\gamma: \bigcup_{i=1}^n \bar{X}_i \cup \bar{Y}_{n+1} \rightarrow (PN \cup K)$  be a partial mapping. Then:*

$$join((\bar{Y}_{n+1})[F_{n+1}], \gamma, \mathcal{C}) = \bigcup_{i=1}^n (\bar{X}_i - dom(\gamma))[E_i\gamma] \cup \{(\bar{Y}_{n+1} - dom(\gamma))[F_{n+1}\gamma]\}.$$

*Example 2.* Referring to Example 1, we show how to join an instance into the (running) context  $\mathcal{C} = \{A_1, B_2, A_3\}$ , where the instance  $A_1$  shares with the instance  $B_2$  the key  $ka$ , by means of which it sends the nonce  $m_1$ .  $B_2$ , in turn, intends to send the nonce  $n_2$  to  $A_3$ . The only open variable in the context is  $x_3$  of  $A_3$ :

$$\{ ()[out(\{m_1\}_{ka}).in(\{?y_1\}_{A_1^-})], ()[in(\{?v_2\}_{ka}).out(\{n_2\}_{A_3^+})], \\ (x_3)[out(\{m_3\}_x).in(\{?y_3\}_{A_3^-})] \}$$

Instance  $B_4$  is joined to the context by instantiating the open variables  $z_3$  and  $z_4$  with the key  $kb$  and the open variable  $w_4$  with the public key of  $A_1$ :

$$\begin{aligned} & \text{join}(B_4, [A_1/w_4, kb/x_3, kb/z_4], C) \\ &= \\ & \{ ()[\text{out}(\{m_1\}_{ka}).\text{in}(\{?y_1\}_{A_1^-})], ()[\text{in}(\{?v_2\}_{ka}).\text{out}(\{n_2\}_{A_3^+})], \\ & \quad ()[\text{out}(\{m_3\}_{kb}).\text{in}(\{?y_3\}_{A_3^-})], ()[\text{in}(\{?v_4\}_{kb}).\text{out}(\{n_4\}_{A_1^+})] \} \end{aligned}$$

### 2.3 Semantics

A brief presentation of the semantics ruling the evolution of a context concludes this section. Semantics of the interactions happening in a context is given by means of two transition systems. The first one,  $(\rightarrow)$ , defined up to structural congruence<sup>2</sup>, models the “stand alone” behaviour of a principal.

**Definition 4.**  $(\rightarrow)$

$$\frac{}{\alpha.E \xrightarrow{\alpha} E} \text{ (in)} \quad \frac{E \xrightarrow{\alpha} E'}{E + F \xrightarrow{\alpha} E'} \text{ (+)} \quad \frac{E \xrightarrow{\alpha} E'}{E \parallel F \xrightarrow{\alpha} E' \parallel F} \quad \text{bv}(\alpha) \cap \text{fv}(F) = \emptyset \text{ (||)}$$

The second transition system,  $(\mapsto)$ , models both communication inside a context and the possible evolution of a context due to the joining of a new instance. Communication behaves in the expected way, but parallel components of the same instance can not communicate with each other.

**Definition 5.**  $(\mapsto)$

$$\frac{E_i \xrightarrow{\text{in}(d')} E'_i \quad E_j \xrightarrow{\text{out}(d)} E'_j \quad d'\sigma = d}{\{(\bar{X}_i)[E_i], (\bar{X}_j)[E_j]\} \cup C \mapsto \{(\bar{X}_i)[E'_i\sigma], (\bar{X}_j)[E'_j]\} \cup C} \sigma \text{ ground (comm)}$$

$$\frac{C' = \text{join}((\bar{X}_i)[E_i], \gamma, C) \quad A \stackrel{\Delta}{=} (\bar{X})[E]}{C \mapsto C'} \text{ (join)}$$

Rule  $(comm)$  says that two instances can communicate if they can perform complementary input/output actions. The input data must equal – via a matching, i.e. a set of mapping  $x \rightarrow d$ , with  $d$  ground term and  $x$  variable – the output data. For the case of asymmetric keys, a private key  $A^-$  matches the corresponding public key  $A^+$  via the empty matching, and this is the only possible matching among asymmetric keys. Notice that a “bad”  $\gamma$  in rule  $(join)$  may yield a “wrong” context where keys are not shared in the intended way.

A context is successful if all its instances are terminated, while it is (may-) correct if it is either successful or if there exists at least one possible evolution leading to a successful context.

**Definition 6 (Successful context).** A context  $C = \cup_i (\bar{X}_i)[E_i]$  is successful if and only if  $\forall i : E_i \equiv 0$ .

**Definition 7 (May-correct context).** A context  $C$  is may-correct if and only if there exists a successful context  $C'$  such that  $C \mapsto^* C'$ .

<sup>2</sup> Structural congruence,  $\equiv$ , is obtained by extending  $\alpha$ -conversion with  $\parallel$  and  $+$  as monoidal operators, whose neutral element is 0.

### 3 Property specification

We now present a logic to express properties about data communicated in a protocol and their relationships. The logic also allows one to predicate over the knowledge  $\kappa$  that may be accumulated by an intruder intercepting data exchanged among the protocol principals. With  $\partial(\kappa)$  we indicate the set of data that can be deduced from  $\kappa$ . Let  $\lambda^-$  be a decrypting (private or symmetric) key and  $\lambda^+$  an encrypting key:

$$\begin{aligned} \lambda^-, \{d\}_{\lambda^-} \in \partial(\kappa) &\Rightarrow d \in \partial(\kappa) & d, d' \in \partial(\kappa) &\Rightarrow (d, d') \in \partial(\kappa) \\ d, \lambda^+ \in \partial(\kappa) &\Rightarrow \{d\}_{\lambda^+} \in \partial(\kappa) & (d, d') \in \partial(\kappa) &\Rightarrow d, d' \in \partial(\kappa) \end{aligned}$$

The logic  $\mathcal{L}$  is based on the sort  $D_i$  that represents the data  $D$  of the calculus, indexed by natural numbers.

**Definition 8 (Logic  $\mathcal{L}$ ).**

$$\begin{aligned} \phi ::= X_i = D_i \mid D_i \in \partial(\kappa) \mid \text{rel}(D_i, D_i) \mid \forall i. \phi \mid \exists i. \phi \mid \neg \phi \mid \phi \wedge \phi \mid \phi \vee \phi \\ D_i ::= PN_i \mid X_i \mid ?X_i \mid K_i \mid PN_i^+ \mid PN_i^- \mid NO_i \mid \{D_i\}_{D_i} \mid (D_i, D_i) \end{aligned}$$

A formula  $\phi$  may be an assertion on the value of a variable of an instance ( $X_i = D_i$ ), an assertion on the knowledge of the intruder ( $D_i \in \partial(\kappa)$ ), or it may express a relation  $\text{rel}$  between data ( $\text{rel}(D_i, D_i)$ ). The formula  $r(a, b)$  describes the relationship between  $a$  and  $b$  intended by the protocol designer. We will require that the relation is *functional* (formally,  $r(a, b) \wedge r(a, c) \Rightarrow b = c$ ). For example, to state that  $A_i$  is the (only possible) author of nonce  $na_i$  one may write  $\text{author}(na_i, A_i)$ .

Index variables (e.g.,  $i, j$ ) quantified by  $\forall$  and  $\exists$  range over a finite segment of natural numbers (corresponding to the instance indexes of a context). For instance, the formula  $\forall i. x_i = na_i$  reads as “for all instances  $i$  (participating in the protocol) in which the variable  $x_i$  occurs,  $x_i$  must assume the value  $na_i$ ”. The operators  $\neg, \wedge$  and  $\vee$  are the usual boolean operators. The derived formulas  $\phi \rightarrow \psi, x_i \neq D_i$  and  $D_i \notin \partial(\kappa)$  read as usual as  $\neg \phi \vee \psi, \neg(x_i = D_i)$  and  $\neg(D_i \in \partial(\kappa))$ , respectively.

frozen

*Example 3.* Referring to Examples 1 and 2, we illustrate two simple properties. The first one says that the symmetric key used by any instance of  $A$  must be kept secret, i.e. must not be known by the intruder:

$$\forall i. x_i \notin \partial(\kappa).$$

The second one expresses a relation among data. If an instance of  $B$  sends  $n$  to the same instance of  $A$  from which it has received the first communication, then the two instances share the same symmetric key:

$$\forall i. \exists j. \text{communicated}(A_j, A_j) \wedge \text{communicated}(v_i, w_i) \rightarrow x_j = z_i$$

With  $\text{communicated}(t, u)$  we mean that  $u$  takes part in the communication of  $t$ . After  $B_i$  has received  $A_j$  in  $v_i$ , the antecedent can be rethought as  $\text{communicated}(A_j, A_j) \wedge \text{communicated}(A_j, w_i)$ . Since relations must be functional,  $w_i = A_j$ . Since  $B_i$

sends  $n$  to  $w_i$ , the antecedent holds if  $B_i$  communicates always with the same instance  $A_j$ . Quantifiers state that implication must hold for all the instance of  $B$  which communicate with an instance of  $A$ .

We now define the satisfiability relation  $\models$  for ground formulas  $\phi$  in normal form, where  $\forall$  ( $\exists$ ) quantifiers are transformed into finite and (or) summations and variables  $x_i$  have been instantiated to ground data values.

**Definition 9 (Model).** *Let  $\mathcal{R}$  be a finite set of functional relations, and let  $\kappa$  be a set of data  $D_i$  (the knowledge of  $I$ ). The pair  $\langle \mathcal{R}, \kappa \rangle$  is a model for a normal form formula  $\phi$  iff  $\langle \mathcal{R}, \kappa \rangle \models \phi$ , where:*

$$\frac{}{\langle \mathcal{R}, \kappa \rangle \models t = t} (=) \frac{t \in \partial(\kappa)}{\langle \mathcal{R}, \kappa \rangle \models t \in \partial(\kappa)} (\in) \frac{(rel, t, u) \in \mathcal{R}}{\langle \mathcal{R}, \kappa \rangle \models rel(t, u)} (rel)$$

$$\frac{\langle \mathcal{R}, \kappa \rangle \models \phi \quad \mathcal{R}, \kappa \models \psi}{\langle \mathcal{R}, \kappa \rangle \models \phi \wedge \psi} (\wedge) \frac{\langle \mathcal{R}, \kappa \rangle \models \phi}{\langle \mathcal{R}, \kappa \rangle \models \phi \vee \psi} (v1) \frac{\langle \mathcal{R}, \kappa \rangle \models \psi}{\langle \mathcal{R}, \kappa \rangle \models \phi \vee \psi} (v2) \frac{\langle \mathcal{R}, \kappa \rangle \not\models \phi}{\langle \mathcal{R}, \kappa \rangle \models \neg \phi} (\neg)$$

with  $u, t$  ground data  $D_i$ . The inductive definition of model is based on a *close world assumption*, in the sense that a couple  $\langle \mathcal{R}, \kappa \rangle$  is a model for a formula  $\phi$  if and only if there exists a proof for  $\langle \mathcal{R}, \kappa \rangle \models \phi$ . This justifies the symbol  $\not\models$  of rule  $(\neg)$ , which reads as “ $\langle \mathcal{R}, \kappa \rangle$  is a model for  $\neg\phi$  if  $\langle \mathcal{R}, \kappa \rangle$  is not a model for  $\phi$ ”, i.e. if does not exist a proof for  $\langle \mathcal{R}, \kappa \rangle \models \phi$ . Since  $\kappa$  and  $\mathcal{R}$  are finite sets, membership in  $\kappa$  and  $\mathcal{R}$  is decidable, and so is equality over ground formulas. It follows that  $\models$ , and hence  $\not\models$ , are decidable, too.

## 4 Protocol verification

The last step of the methodology consists of verifying whether a certain property  $\phi$  holds for a given protocol specification  $\mathcal{P}$ . The verification process inputs both  $\mathcal{P}$  and  $\phi$ , and determines whether there may exist an intruder  $I$  capable of letting a context of (instances of) protocol principals successfully terminate, while violating the expected property  $\phi$ . Verification relates in this way the “static” declarative aspects of the property to “dynamics” of a protocol multi-session run.

In a more general setting,  $\phi$  satisfiability could be checked other than in (global) termination points, provided that all the (instance) variables of the formula have been instantiated. In this paper, we have restricted our attention to the end of successful runs only. We are rather interested in intruders that are able to steal secrets without that the failure of the whole protocol warns of their malicious activity.

The verification process may be additionally feed by The choice of checking  $\phi$  only the end of successful runs where all the variables of  $\phi$  are instantiated is motivated by the idea that an intruder that can stole some secrets, but cannot use them without being detected, cannot invalidate some security properties (like authentication). On the other hand, a possible generalisation of our approach may choose different point for checking  $\phi$ , provided that all the (instance) variables

of the formula have been instantiated. specifying the initial (possibly empty) knowledge  $\kappa_{init}$  of the intruder and the maximum number  $n$  of protocol sessions to be considered. These two parameters may support a quantitative analysis of the robustness of the protocol. In principle,  $n$  needs not to be fixed *a priori*, but practically the algorithm runs up to a given  $n$ . The knowledge  $\kappa_{init}$  may contain unhazardous data, like nonces the intruder can generate, or also keys supposed to be secret.

The verification is performed by an algorithm that consists of two main steps:

1. *(Candidate) intruder construction.* The algorithm nondeterministically constructs an evolving context of (instances of) protocol principals, according to the context semantics described by rules *(comm)* and *(join)* of Definition 5. Starting from the empty context, the algorithm nondeterministically joins new principal instances and incrementally constructs an intruder that interferes with the communications among the participants.

More precisely, the intruder tries to match all output communications performed by the principals inside the context. When there is no more output operation to match, the intruder tries to output some value or to join a new principal in the context. (The nondeterminism of the algorithm originates from the choice of which output operation is matched by the intruder and by the choice of which principal is added and how it is connected to the other instances in the context.)

Intuitively speaking, the goal of the construction is to determine an intruder  $I$  and a set of principal instances  $\eta$  such that the context consisting of  $\eta$  and  $I$  is may-correct. Ultimately, the intruder construction phase produces a set of quadruples, relative to the successful traces obtained, of the form

$$\langle I, \chi, \kappa, \eta \rangle$$

where:

- $I$  is the candidate intruder identikit
- $\chi$  is the set of bindings of (instance) variables, due to communication and join operations,
- $\kappa$  is the final knowledge accumulated by  $I$ ,
- $\eta$  is the set of principal instances forming the context.

2. *Property satisfaction.* For each quadruple  $\langle I, \chi, \kappa, \eta \rangle$  generated at the previous step, the algorithm checks whether there exists a set of relations  $\mathcal{R}$  satisfying the given property  $\phi$ .

More precisely, the algorithm constructs, if any, a set of relations  $\mathcal{R}$  such that:

$$\langle \mathcal{R}, \kappa \rangle \models \neg \mathcal{N}_{\chi, \eta}(\phi)$$

where  $\mathcal{N}_{\chi, \eta}(\phi)$  is the normalization of  $\phi$  obtained by instantiating it by means of the bindings  $\chi$  and the set  $\eta$ , and by eliminating  $\forall$  and  $\exists$  quantifiers. Since the model is finite, the possible construction of  $\mathcal{R}$ , can be performed by means of an effective non-deterministic proof procedure.

## 5 Example: The Needham-Shroeder public key protocol

As an example of the methodology we apply it to the Needham-Shroeder public key protocol. The protocol authentication phase is informally specified as follows:

$$\begin{aligned} A &\rightarrow B : \{na, A\}_{B^+} \\ B &\rightarrow A : \{na, nb\}_{A^+} \\ A &\rightarrow B : \{na, nb\}_{B^+} \end{aligned}$$

$A$  sends  $B$  a nonce and its own name encrypted with the public key of  $B$ ;  $B$  sends  $A$  the couple of nonces  $na, nb$  (both encrypted with the public key of  $A$ ) where  $nb$  is a new nonce generated by  $B$ . Finally,  $A$  confirms to  $B$  of being its partner in the protocol by sending back the same couple of nonces encrypted with the public key of  $B$ .

The protocol has been designed to achieve the authentication of  $A$  to  $B$ . Indeed,  $A$  and  $B$  exchange their nonce ( $na$  and  $nb$ , respectively) in order to mutually “recognize” themselves. More precisely, when  $B$  receives the last message, he assumes that the identity of the partner is  $A$  because he associated  $nb$  to the nonce  $na$  received in the first message  $\{na, A\}_{B^+}$  that  $B$  interprets as “ $A$  created the nonce  $na$  to gain authentication”.

Protocol principals are formalised as:

$$\begin{aligned} A &\triangleq (y)[out(\{na, A\}_{y^+}).in(\{na, ?u\}_{A^-}).out(\{na, u\}_{y^+})] \\ B &\triangleq ()[in(\{?x, ?z\}_{B^-}).out(\{x, nb\}_{z^+}).in(\{x, nb\}_{B^-})] \end{aligned}$$

The instance  $A$  starts creating a nonce that is deemed to be used for authenticating  $A$  to  $B$ . The nonce received by  $A$  is supposed to be generated by  $B$ . The instance of  $B$ , after the reception of the first message, assumes that the nonce  $x$  has been created by  $z$ . Then  $B$  creates a new nonce  $nb$  in order to authenticate  $z$  via the couple  $\langle x, nb \rangle$ . We may formalise this statement with a  $\mathcal{L}$ -formula  $\phi$ :

$$\phi \equiv \forall i. \exists j. (u_i = nb_j \rightarrow (x_j = na_i \wedge z_j = A_j \wedge y_i = B_j))$$

For each  $A_i$ , instance of  $A$ , there exists  $B_j$ , an instance of  $B$ , such that, if the nonce generated  $B_j$  is received by  $A_j$  then  $B_j$  receives the nonce generated by  $A_i$  and, furthermore,  $A_i$  and  $B_j$  “believe” that they had communicated each other.

We then apply the verification procedure to the Needham-Shroeder protocol and the property  $\phi$ , with an empty  $\kappa_{init}$ .

The first step returns, for the case of a successful trace of a session among the instances  $A_1, B_2$  and  $I$ , the tuple  $\langle I, \chi, \kappa, \eta \rangle$ , where:

*Notazione da sistemare!!!*

$$\begin{aligned} I &\triangleq ()[in(?u_1).out(\{na_1, A_1\}_{B_2^+}).in(?u_2).out(\{na_1, nb_2\}_{A_1^+}).in(?u_3).out(\{na_1, nb_2\}_{B_2^+}).0] \\ \chi &= \begin{cases} y_1 \rightsquigarrow I, \\ x_2 \rightsquigarrow na_1, \\ z_2 \rightsquigarrow A_1, \\ u_1 \rightsquigarrow nb_2 \end{cases}, \kappa = \left\{ A_1, \{na_1, A_1\}_{I^+}, B_2, \{na_1, nb_2\}_{A_1^+}, \{na_1, nb_2\}_{I^+} \right\}, \eta = \{A_1, B_2\}. \end{aligned}$$

The second step verifies that  $\chi, \kappa$  and  $\eta$  satisfy the formula:

$$\mathcal{N}_{\chi, \eta}(\neg\phi) \equiv nb_2 = nb_2 \wedge (na_1 \neq na_1 \vee A_1 \neq A_1 \vee I \neq B_2)$$

dire che la procedura restituisce  $R$ , e che  $R$ , vuoto soddisfa  $\phi$  e che  $I$  e' quello - guardacaso - di Lowe

that is easily verified and, therefore, the algorithm output is  $\langle I, \chi, \kappa, \eta \rangle$ . We conclude that the property  $\phi$  does not hold for the Needham-Schroeder public key protocol. We remark that  $I$  behaves like the Lowe's attacker [Low95].

## 6 Concluding Remarks

We have introduced a formal methodology to specify security protocols in multi-session runs, and verify their properties.

In the last years several formal techniques have been proposed and employed for the analysis of security protocols. Some are based on finite state model checking [CJM98,Low95,MSS98] and inductive theorem proving techniques [Mea96,Pau98]. Others rely on a process calculus representation of protocols and some form of observational equivalence to specify their properties [AG99,FG96]. Our approach has many commonalities with both approaches. In particular we adopted a variant of the SPI-calculus [AG99] to describe the behaviour of principals, and we extended the notion of *magic instance* [AG99] to handle general protocol properties. The semantic model of our calculus is basically the set of all possible traces of a security protocol. However, the reduction of protocol verification to termination allows us to keep the state space finite without making severe limitation or simplifying assumptions about the model of the attacker. A distinguishing and innovative feature of our verification procedure is the explicit handling of the knowledge of the attacker. This has the main advantage that the verification algorithm can effectively provide an intruder, if any.

Future work will be devoted to validate further the proposed methodology by experimenting its application to the work-bench examples of [CJ96] by exploiting the prototype implementation (currently under development). A promising approach to reduce the cost of the state space exploration is to take fully advantage from the syntactic structure of the logical formula (expressing the property) and the knowledge of the attacker to discard useless alternatives. Another important lines of development concerns the integration of the methodology into a full fledged calculus for component composition. We plan to extend the verification algorithm to handle secure composition of components (an overview of the approach together with some preliminary results can be found in [?]). Finally, we plan to investigate the relationships with the so called *symbolic* approaches. Recent results [AP99,ALV01,Bor01], have shown that interesting properties such as secrecy or authenticity are handled by exploiting this technique. Stressing the open-endedness of both approaches.

## References

- [AG99] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 10 January 1999.
- [ALV01] R. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. Technical report, INRIA-Sophia, 2001.

- [AP99] Roberto M. Amadio and Sanjiva Prasad. The game of the name in cryptographic tables. In P. S. Thiagarajan and R. Yap, editors, *Advances in Computing Science - ASIAN'99 December, 1999*, volume 1742 of *LNCS*, pages 15–26. Springer, December 1999.
- [BBT01] Andrea Bracciali, Antonio Brogi, and Franco Turini. Coordinating interaction patterns. In *Proceedings of the ACM Symposium on Applied Computing, Las Vegas, USA*. ACM, 2001.
- [Bor01] Michele Boreale. Symbolic trace analysis of cryptographic protocols. In *28th Colloquium on Automata, Languages and Programming (ICALP)*, LNCS. Springer, July 2001.
- [CJ96] John Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature. Unpublishe, August 1996.
- [CJM98] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *In Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.
- [DY83a] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE/TIT: IEEE Transactions on Information Theory*, 29, 1983.
- [DY83b] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
- [FG96] Riccardo Focardi and Roberto Gorrieri. Automatic compositional verification of some security properties. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 167–186. Springer-Verlag, 1996.
- [Low95] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.
- [Mea96] Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, February 1996.
- [MSS98] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In *Proceedings of the 7th USENIX Security Symposium (SECURITY-98)*, pages 201–216, Berkeley, January 26–29 1998. Usenix Association.
- [NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
- [Pau98] Lawrence C. Paulson. *The inductive approach to verifying cryptographic protocols*. Technical report; no. 443. 4006797499. University of Cambridge Computer Laboratory, Cambridge, UK, USA, February 1998.