A Coordination-based Methodology for Security Protocol Verification

Giacomo Baldi, Andrea Bracciali, Gianluigi Ferrari, and Emilio Tuosto

Dipartimento di Informatica
Università di Pisa, Italy
{baldig,braccia,giangi,etuosto}@di.unipi.it

Abstract. The quest for the formal certification of properties of systems is one of the most challenging research issues in the field of formal methods. It requires the development of formal models together with effective verification techniques. In this paper, we describe a formal methodology for verifying security protocols based on ideas borrowed from the analysis of *open* systems, where applications interact one with another by dynamically sharing common resources and services in a not fully trusted environment. The methodology is supported by $\mathcal{A}$SPASyA, a tool based on symbolic model checking techniques.

1 Introduction

Software applications are evolving towards *open* architectures. The key aspect in this shift is the relevance that interaction and dynamics have assumed in the life of applications. An open system is characterised by being composed of autonomous interacting components, whose configuration may dynamically vary in time. Moreover, the whole architecture of the system may be only partially accessible, both because its dynamics and because it may be distributed over different, non-controllable domains. In other words, interacting components have to coordinate their behaviour according to the dynamic supply and demand of resources. This scenario fosters the creation of an environment of *services*, offered on the net and accessed on demand by a phase of dynamical binding (preceded by complementary publishing and discovering phases). The standard example is provided by the *Web Services* technology, [14].

The advantages of an *architectural design of components,* and the way they offer and use services, have been investigated from several different perspectives. The dynamic coordination of active components, understood as *"the process of building programs by gluing together active pieces"*, [10] or the theory about *"the structure of the components of a program/system, their interrelationships, and principles and guidelines governing their design and evolution over time"* [17], has motivated a lot of research interest. However, despite the many advances obtained, lifting traditional techniques to the case of services integration in open systems present a set of recognised difficulties, which impose an upgrade of the goals and techniques that must be developed for a mature architectural design, [28, 3]. For instance, extra information must be formalised by the component abstractions, like the assumptions on which they are based on and the conditions under which they can be integrated into a system, possibly by means of dynamic binding mechanisms. Moreover, affordable verification techniques and means to express the properties of interest, which able to deal with the dynamics of the systems, are needed.

In this paper we present a specification and verification methodology tailored for security protocols, which has been inspired by the more general context outlined above. It basically considers a protocol as being an open system where participants, called principals, may dynamically join and coordinate themselves in multiple running session of the protocol.

The methodology consists of four steps: 1) Specification of the behaviour of the principals and the property expected to be enforced by the protocol, 2) specification of the conditions about how secrets can be beforehand shared among principals (and hence about how they can be dynamically connected and communicate together), 3) specification of the power of the intruder, in terms of its initial knowledge, 4) automatic verification of whether the protocol executions, as they have been formalised, do or do not satisfy the property. We have observed that the results of step 4) can be fruitfully exploited to iterate steps 2), 3), and 4), according to the insights gained in the previous iterations about the actual, and often unexpected, behaviour of the protocol.

The methodology relies on a formal framework introduced in [8, 33], and consists of a calculus for the formal description of principals enhanced with linguistic mechanisms for dynamic bindings, and an ad-hoc logic to express the security properties to be checked. The logic predicates over data exchanged in the protocol and observed by a possibly malicious component in the execution environment, and also over relations about the presumed identities of the principals that exchange the data.

The methodology is supported by an automated tool, called $\mathcal{A}$SPASyA (after Automatic Security Protocol Analysis via a SYmbolic model checking Approach), based on model checking techniques. Roughly, model checking assumes the system behaviour described in terms of transition state systems , while properties that must be verified are represented as logic formulas. The possible models for a formula are represented by a significant subset of states, for instance those corresponding to terminating computations. If the formula "exhaustively" holds in all the significant states, verification is successful, otherwise a counter example is typically produced. Among the advantages of model checking we remind efficiency (obtained by exploiting very refined data structures, like BDDs), and the possibility of being completely automated, [13]. In our case, symbolic techniques have been adopted to overcome some intrinsic sources of incompleteness. Namely, during state space generation, constraints on the possible values of principals' variables are collected, then $\mathcal{A}$SPASyA checks whether an assignment for the variables, such that the formula is satisfied, does or does not exist. The overall methodology and $\mathcal{A}$SPASyA are quite human-interactive, both because the high degree of expertise required by the problem and its formalisation, and to allow the verification process to be guided towards the properties of interest. Once the protocol and its properties are tuned by the user, the automatic verification phase starts. The user can vary the intruder's knowledge, the portion of the state space to be explored, and the specification of implicit assumptions that are very frequent in security. The user can opportunely mix those three ingredients for testing the correctness of the protocol without modifying neither the protocol specification nor its related security properties.

Model checking has been applied by other approaches to the verification of security protocols [21, 26, 12]. Recently, [5, 6] have introduced STA, a model checker for verifying causality relation on cryptographic protocols; one of the first approaches based on a symbolic technique for compacting the state space. A similar approach is adopted in TRUST [2].

The following Section 2 and Section 3 introduce security protocols, and the formal framework, respectively, to put the reader in context. The verification methodology is illustrated in Section 4 together with a comparison with similar approaches, while Section 6 contains some concluding remarks.

2 Security Protocols: An Overview

Security protocols are intended to guarantee the security of "relevant" information in a scenario where some partners, hereafter called *principals*, communicate through a "public channel". Relevant information has a very broad meaning, encompassing not only information that should be kept secret or non-modifiable (e.g. a credit card number, an encryption key, etc.), but also the very identity of the principals participating in a protocol. On the other hand, the channel is public: It does not provide means to avoid that the exchanged messages are accessed, manipulated or destroyed by the potentially malicious entities, the *intruder* components, living in the same environment of the channel.

In general, the formal certification of security protocols requires a careful definition of the underlying assumptions upon which the protocol relies, of the security property it is supposed to enforce, and also of the hypothesis on the capabilities of the intruder. In the following, we briefly review security protocols, referring to [30, 23] for a more comprehensive introduction.

Cryptography. A message m is a *plaintext* when the information it contains can be obtained directly from m, while it is a *cryptogram* when the information is "hidden" in m, and can be extracted only under specific hypothesis.

Given a *key*, the message m is *encrypted* in a new message $\{m\}_k$, a cryptogram that depends on k and such that the original information can be retrieved only by means of the knowledge of the key. We adopt the standard working assumption of *perfect encryption*: A cryptogram can be *decrypted* only using its decryption key, i.e. secrets cannot be guessed, no matter how much information is possessed. In symmetric cryptography, the same key k is used for encryption and decryption, while in asymmetric cryptography one key, called *public key* since it is publicly available, is used for encryption, and a complementary *private key*, typically known only be a principal, is used for decryption. Any principal can encrypt a message for another principal by means of its public key. The latter is the only one that can decrypted the message with its private key.

In the following we denote the intruder by I, principals by $A, B, \ldots$, A^- and A^+ are the private and public keys of a principal A, λ^- is the complementary key of λ (i.e. A^+ if $\lambda = A^-$, A^- if $\lambda = A^+$, and k if $\lambda = k$ is a symmetric key), and m, n is the pair of messages m and n. For the ease of presentation, we assume that keys have no structure, i.e. they are simply names. Structured keys add a source of complexity to the analysis that can be dealt with by the same symbolic techniques illustrated in Section 3 for handling structured, and hence infinite in number, messages.

Notice that an appropriate sharing of keys, that may also be obtained dynamically, is an essential feature for the correct assignment of the roles played by the principals in the execution of protocols. For instance, keys and their ownership may be used to attribute identity to principals.

Protocol specification. A *security protocol* may be naively thought of as a finite sequence of messages between two or more participants. There is a great variety of specification mechanisms for protocols and properties, but traditionally an informal mix of natural language and ad hoc notation is used (for instance, SSL [16], SSH [35], IKE [18] are specified in this style). The next example illustrates the informal specification of a protocol as a list of communication steps. This notation will be used in the rest of the paper to introduce protocols.

Example 1. The Wide Mouthed Frog (WMF) protocol [9] aims at letting A send a fresh session key kab, i.e. a key to be used within a limited temporal interval, to B through a trusted server S. Both A and B share two private keys with S (kas and kbs, respectively). The WMF protocol is:

$$\begin{array}{l} (1)\ A \rightarrow S : A, \{ta, B, kab\}_{kas} \\ (2)\ S \rightarrow B : \{ts, A, kab\}_{kbs} \end{array}$$

A sends S its own identity and a cryptogram with the identity of B, the session key kab and a new *nonce* (after name-once) ta, that is used to mark the current session of the protocol. S sends B a cryptogram with kab, the identity of A, and a new nonce ts, guaranteeing the timeliness of the session key kab. ◇

Notice that the above description is not a complete specification, since it does not specify, for instance, whether or not only B and S must know kab and ts, and who can access the exchanged messages. Moreover, many of the known attacks consist of an intruder exploiting the information acquired by playing different roles in concurrent sessions of the protocol. Hence, a clear specification of how sessions can be interleaved, and of the security properties expected after such concurrent runs, is also necessary, as recognised for instance in [9].

Security properties. Among the many security properties for enforcing which a protocol can be designed, we restrict here to *integrity* i.e., informally speaking, the intruder cannot corrupt exchanged cryptograms, *secrecy*, it cannot know exchanged cryptograms, and *authentication*, the intruder cannot let a principal misunderstand the identity of its partners in the communications. Even if there is not agreement about a general classification of security properties and their formalisation, these can be considered the "elementary" properties that protocols aim at guaranteeing, are quite related to each other, and can be the basis for more complex kind of properties, like *fairness* and *non-repudiation*.

Example 2. An obvious properties that the WMF protocol of Example 1 is expected to enforce is the secrecy of kab: In every session its value must be known only by the principals playing the roles of A, B and S. Moreover, also the authentication of A to B is desirable: In every session where B receives the message from S, it must be ensured that, *in the same session*, A has created kab and asked S to forward it to B. $\diamond$

Intruder model. We adopt the widely accepted Dolev-Yao intruder model [15]. In this model the intruder is a principal that can interfere with all the communications, e.g. by hiding, reading, modifying a message, with the only limit of the perfect encryption hypothesis. The intruder can have some private data, and can store data exchanged in previous runs of a protocol. It is characterised in terms of its acquired knowledge, and it can be formalised as the execution environment which collects all the sent messages and can manipulate and send them to the other principals waiting for some input.

Within the scope of this paper, the set $M = \{m, n, \dots\}$ of messages is defined as

$$M \ ::= \ N \ \mid \ K \ \mid \ M, M \ \mid \ \{M\}_M$$

where N is a countable set of names, containing nonces and principal names, while K contains both symmetric and asymmetric keys.

The Dolev-Yao intruder is characterised by a set of messages κ, the intruder knowledge, and the messages it can derive from it, written as $\kappa \bowtie m$, where $\bowtie$ consists of splitting known pairs, pairing and encrypting known messages, and decrypting known cryptograms whose key is known. In [12] decidability of $\bowtie$ has been proved for private key cryptography; decidability of $\bowtie$ for public key cryptography has been proved in [8, 33].

Example 3. Given $\kappa = \{\{A^{-1}\}_k,\ \{m\}_{A^+},\ k\}$, it holds $\kappa \bowtie \{m\}_k$. Indeed, from $\kappa \bowtie \{A^{-1}\}_k$ and $\kappa \bowtie k$, it follows $\kappa \bowtie A^{-1}$, which allows the decryption of $\{m\}_{A^+}$ obtaining m. Finally, from $\kappa \bowtie k$ it follows $\kappa \bowtie \{m\}_k$.

3 Formalising Security Protocols

The methodology we present rely upon a framework consisting of two ingredients: The *cryptographic Interaction Pattern* calculus (cIP), and the *Protocol Logic* ($\mathcal{PL}$), introduced in [8, 33]. The calculus allows us to formally specify the principals of a protocol, indicating how they can be connected together, i.e. how keys can be shared by them, and the behaviour they can exercise. It consists of an instance of the IP-calculus, introduced in [8] for describing the behavioural composition of components in open systems. The logic is used to formalise the properties that the protocol is expected to enforce. It predicates over the intruder knowledge, the way principal share keys and the data they exchange. Moreover it allows for quantification over principal instances in order to uniformly express properties about multiple-runs of protocols.

The cIP calculus The cIP calculus is a name-passing process calculus of the π-calculus family [25]. It extends the IP calculus with cryptographic primitives, in the style of [1], and with explicit constructs for the dynamic sharing of secrets.

A cIP processes $A \triangleq (\tilde{X})[E]$ stands for the principal A that is ready to share keys represented by its *open variables* $\tilde{X}$ and behave according to its *behavioural expression* E. A behavioural expression consists of a finite sequence of input and output actions $in(d)$ or $out(d)$ (over a generic public channel, whose name is omitted). The datum d is a message where variables can appear. Open variables are binders for the free occurrences of $\tilde{X}$ in E; usual scoping rules hold: Occurrences of variables in input actions are binder, and are denoted as $?x$ (we assume that output actions do not contain binders and input actions contain at most one binder of each variable). A variable can hence be instantiated either because of a communication action, or of a dynamic connection done when the principal is connected together with its partners in a protocol execution. Notice that open variables provide a general mechanism to deal with dynamic binding of components sharing resources, communication channel for instance. In this paper, this mechanism has been specialised to the sharing of cryptographic keys that allows principal interactions within a protocol run.

Names and variables are syntactically distinguished entities, the former are constant terms, the latter placeholders that can be substituted with terms or opportunely renamed. A principal must be closed: Variables must be bound by either input actions or open variables.

Example 4. The principals of the WMF protocol can be formalised in cIP as:

$$\begin{aligned} A &\triangleq (x, xas)[out(A, \{ta, x, kab\}_{xas})], \\ S &\triangleq (u, ya, v, yb)[in(u, \{?t, v, ?r\}_{ya}).out(\{ts, u, r\}_{yb})], \\ B &\triangleq (zbs)[in(\{?s, ?x, ?w\}_{zbs})]. \end{aligned}$$

Principal A intends to agree on a session key with a partner whose identity will be assigned to the open variable x when A will join a protocol run (together with a given S and B). The open variables xas of A and ya of S, by being instantiated with the same symmetric key, allow the two principals to share a key. Similarly, yb and zbs play the same role for S and B. Finally, the server S gets the identity of A and B in u and v. $\diamond$

In order to model multiple-runs, principals may be replicated in *principal instances*, obtained by indexing all variables (open or not) and all names in E with a distinguished natural number, e.g. $A_1 \triangleq (x_1, xas_1)[out(A_1, \{ta_1, x_1, kab_1\}_{xas_1})]$. Terms with different indexes are distinguished, e.g. $A_1 \neq A_3$ are different principal instances.

Instances run in a *context*, i.e. a set of running instances which may be dynamically joined by other instances, via a new sharing of keys, as explained below. Notice that the actual sharing is determined off-line as a mapping from open variables to keys. Indeed, the verification task is not oriented to determine the "right" sharing of secrets, but in testing whether a given sharing does or does not allow for an attack.

Let $\mathcal{C}$ be a (running) context, with $n-1$ instances of principals, and $\mathrm{ov}(\mathcal{C})$ the set of the open variables of its principals. Given $A_n \triangleq (\tilde{X}_n)[E_n]$ a principal instance, and γ a partial mapping from $\mathrm{ov}(\mathcal{C}) \cup \tilde{X}_n$ to the set of keys, then the *join* operation is defined as:

$$join(A_n, \gamma, \mathcal{C}) = (\tilde{X}_n - dom(\gamma))[E_n\gamma] \;\cup \bigcup_{(\tilde{Y})[E'] \in \mathcal{C}} (\tilde{Y} - dom(\gamma))[E'\gamma].$$

$$\frac{\kappa \bowtie m : \; \exists \gamma \text{ ground s.t. } d\gamma \sim m}{\langle (\tilde{X}_i)[in(d).E_i] \cup \mathcal{C}, \chi, \kappa \rangle \;\mapsto\; \langle (\tilde{X}_i)[E_i\gamma] \cup \mathcal{C}, \chi\gamma, \kappa \rangle}\;(in)$$

$$\frac{}{\langle (\tilde{X}_i)[out(m).E_i] \cup \mathcal{C}, \chi, \kappa \rangle \;\mapsto\; \langle (\tilde{X}_i)[E_i'] \cup \mathcal{C}, \chi, \kappa \cup m \rangle}\;(out)$$

$$\frac{\mathcal{C}' = join(A_i, \gamma, \mathcal{C}) \qquad A \stackrel{\triangle}{=} (\tilde{X})[E] \qquad i \text{ new}}{\langle \mathcal{C}, \chi, \kappa \rangle \;\mapsto\; \langle \mathcal{C}', \chi\gamma, \kappa \cup \{A_i, A_i^+\} \rangle}\;(join)$$

Fig. 1. Context reduction semantics

It returns the new context joined by the principal instance, according to the mapping. Open variables must be assigned to symmetric or asymmetric keys according to their usage (that is explicit in the syntax of the principal). Once assigned, they are no longer open.

Semantics of contexts The operational semantics of contexts is given in terms of the Labelled Transition System (LTS) $\mapsto$, illustrated in Figure 1. The semantic model of cIP formalises the Dolev-Yao intruder model. All the communications occur throughout the intruder: It records all the exchanged messages and the names of the principal instances that join the context, and it sends messages derived from its knowledge to principals. In the simplest case it only forwards messages, allowing for the intended execution of the protocol to take place. States of the LTS $\mapsto$ are triple $\langle \mathcal{C}, \chi, \kappa \rangle$, where $\mathcal{C}$ is a context, χ are the bindings due to communications and join executions, and κ is the intruder knowledge. A principal can input a datum, if an appropriately matching $\sim$ message m can be derived from κ, rule (in). All the outputs of the principals are recorded in κ, rule (out), as well as the name and the public key of every principal joining the context, rule $(join)$.

Join is not an operator of the calculus. The way principals are connected together, namely how keys are initially shared, is typically given beforehand. Since this is determinant for the aims of protocol verification, it has been described at the semantic level, which hence formally describes the consequences of a new principal joining the running protocol. As it will be discussed in Section 4, this also provides an efficient mean to control the conditions of the verification experiment, and to focus on those initial sharing of secrets which are more significant for the properties one wants to certify.

Cryptography. Differently from others proposals, like [1, 6], encryption and decryption are embedded into communications via the notion of matching (rule (in)). Input actions must declare the key with which they intend to receive and decrypt a cryptogram, and the synchronization can take place only if the input and output messages match, namely $\{m\}_\lambda \sim \{n\}_{\lambda^-}$, with $m \sim n$. This guarantees a correct use of keys. For instance, $\{m\}_k$ matches the input message $in(\{?x\}_k)$, since the symmetric key k decrypts the cryptogram, while $\{m\}_{k1}$ and $\{?x\}_{k2}$ would not match, with $k1 \neq k2$. After the communication, variable x is instantiated with the content m of the cryptogram $\{m\}_k$. This approach supports *i) modularity,* since any data type system can be embedded in the notion of matching, *ii) tight correspondence of traces,* since only "correct" messages can be communicated, and *iii) symbolic analysis*, since the structure of the input message can be used to "finitely" derive from κ a matching message, as discussed below.

Symbolic analysis The semantics of cIP contains a source of incompleteness in the choice of the message m, rule (in), among the infinitely many that can be derived from κ as input message for a principal. This leads to an infinite state space (we refer to [12] about sources of incompleteness in protocol analysis). In order to address this problem, we have adopted a symbolic approach, consisting in delaying the choice of the message and annotating the inputs variables with a finite representation of the current κ, and, hence, of all the messages that can be derived from it. We represent this, for an input variable x, by means of the *symbolic variable* $x(\kappa)$, for "x can assume any message m such that $\kappa \bowtie m$". Symbolically matching messages can be structurally composed in terms of symbolic variables and standard messages. Future evolutions of the context, may further specify the values of $x(\kappa)$, for instance when it is used in place of a key. Symbolic variables allow the existential quantifier of rule (*in*) to be replaced by a constructive generation of a finite set of possible symbolic messages. A symbolic trace represents all the concrete traces obtained by instantiating its symbolic variables with the messages derivable from the associated knowledges. Proving the correspondence of symbolic and concrete traces [8, 33] guarantees the correctness of the verification and dramatically impacts, together with other optimisations, on efficiency.

Example 5. A configuration of a context of the WMF protocol, joined by the principal instances A_3, S_2 and B_1 is $\langle \{A_3, S_2, B_1\},\ \emptyset, \gamma_0\rangle$, where

$$\begin{array}{lcl} A_3 & \triangleq & ()[out(A_3, \{ta_3, B_1, kab_3\}_k)] \\ S_2 & \triangleq & ()[in(A_3, \{?t_2, B_1, ?r_2\}_k).out(\{ts_2, A_3, r_2\}_k)] \\ B_1 & \triangleq & ()[in(\{?s_1, ?x_1, ?w_1\}_k)] \end{array} \quad \text{and} \quad \gamma_0 = \begin{cases} zbs_1, xas_3, ya_2, yb_2 & \mapsto k \\ x_3, v_2 & \mapsto B_1 \\ u_2 & \mapsto A_3 \end{cases}$$

Notice that, even if not intended, this context models the case in which A_3, S_2 and B_1 share the same key k. A possible terminating trace leads to the final intruder knowledge $\kappa = \{A_3, \{ta_3, B_1, kab_3\}_k, \{ts_2, A_3, kab_3\}_k\}$, and mapping $\gamma_1 = \gamma_0\{^{ta_3, kab_3, ta_3, A_3, kab_3}/_{t_2, r_2, s_1, x_1, w_1}\}$, which is generated when the environment sends the cryptogram $\{ta_3, B_1, kab_3\}_k$ to B_1 (observe that the match holds). $\diamond$

The cIP calculus offers the possibility of *uniformly* extending a context with new instances of principals of the protocol. What it is meant by "uniformly" is the fact that variables and names occurring in principal expressions are labelled with a unique index when instances join the context. This linguistic mechanism allows us to determine which are the instances that originated the names used through the execution of the protocol as well as to distinguish between different participants playing the same rôle.

$\mathcal{PL}$ logic Security properties are expressed by means of the $\mathcal{PL}$ logic (Protocol Logic), which predicates over the messages derivable from κ, the values assumed by variables and their relationship with the principals that generate and communicate them, reflecting the features of cIP calculus. Integrity is read as the possibility of fixing some values, generalizing the approach introduced in [1], secrecy as the values contained or not contained in κ, and authentication through relations among principals' variables. More precisely, let δ, σ be messages, variables, or the distinguished name of the intruder I, the formulae ψ, ϕ of $\mathcal{PL}$ are

$$\psi, \phi = \delta \in \kappa \quad | \quad \delta = \sigma \quad | \quad \forall A.i : \phi \quad | \quad \neg\phi \quad | \quad \phi \wedge \psi,$$

where $\in$ is read as *derivability*, $=\neg, \wedge$ are equality, negation and conjunction, and $\forall A.i : \phi$ allows for quantification over instances: i is a variable ranging over instance indexes, possibly occurring as indexes in ϕ.

Example 6. A property that the WMF protocol should satisfy is the secrecy of the session key kab, unless it is really intended for I: $\forall A.i : x_i \neq I \rightarrow kab_i \notin \kappa$, i.e. "for every instance of A, which has not I as partner (x_i is the open variables of A for the name of its partner), then the session key kab_i (generated by A_i) does not belong to κ". $\diamond$

Formulae depend on both κ and assignments χ, i.e. they depend on the past computation and are hence verified against models consisting of couples κ, γ produced by terminating traces, since those are the states of interest where the formula must hold. The notation $\kappa \models_\chi \phi$ means that κ, under the variable assignment χ, is a model of the formula ϕ. Relation $\models$ (for closed formulas) is defined by deduction rules, like (and the analogous ones for $\in$, $\bowtie$, and $\wedge$):

$$\frac{x_i\chi = \delta\chi}{\kappa \models_\chi x_i = \delta}\,(=) \qquad \frac{\kappa \not\models_\chi \phi}{\kappa \models_\chi \neg\phi}\,(\neg) \qquad \frac{\kappa \models_\chi \phi\{^{A_j}/_i\} \quad \text{for all } A_j : \kappa \bowtie A_j}{\kappa \models_\chi \forall A.i : \phi}\,(\forall).$$

Index quantification ranges over the finite number of instances that participated to a session, whose name is in κ ($\forall$). Moreover, it can be proved that $\models$ is decidable and hence ($\neg$) is well-defined. Finally, the lifting of the logic to the case in which symbolic variables occur in formulas as effect of assignments is easy, considering symbolic variables as conditions of membership to the associated κ.

Example 7. Considering again the WMF protocol, one may wish to verify the secrecy property written in its naive form, as $\phi = \forall A.i : kab_i \notin \kappa$: "The session key generated by A is not know by the intruder". Let us suppose that we want to let I play the role of a normal principal, namely B. In this case, it is sufficient to add a key, say ksi to κ. Let us consider the context where S shares ksi with I, and A wants to speak with I:

$$\begin{array}{rcl} A_1 & \triangleq & ()[out(A_1, \{ta_1, I, kab_1\}_{kas})] \\ S_2 & \triangleq & ()[in(A_1, \{?t_2, I, ?r_2\}_k).out(\{ts_2, A_1, r_2\}_{ksi})] \end{array} \quad \text{where} \quad \gamma_0 = \begin{cases} xas_1, ya_2 & \mapsto kas \\ yb_2 & \mapsto ksi \\ x_1, v_2 & \mapsto I \end{cases}$$

It is easy to verify that $\kappa_1 = \{A_1, S_2, I, ksi, \{Ta, I, kab\}_{kas}, \{Ts, A, kab\}_{ksi}\}$, and $\gamma = \{xas_1, ya_2 \mapsto kas,\ yb_2 \mapsto ksi,\ x_1, v_2 \mapsto I,\ t_2 \mapsto ta_1,\ r_2 \mapsto\ kab_1\}$, which are produced by a terminating trace of the context, are not a model for ϕ. The discovery of this "spurious", but possible, attack can easily be avoided by further specifying the property in its original formulation: $\forall A.i : x_i \neq I \rightarrow kab_i \notin \kappa$. In this case the antecedent of the formula is false, and the trace is not considered as an attack. This kind of expressiveness of the logic will result helpful in focusing the verification on "intended" classes of attacks.

◇

4 The Verification Methodology

The formal framework introduced in Section 3 supports a methodology for the analysis of properties of security protocols. Our approach allows the user to directly control and manage many aspects of the verification.

Our methodology consists of four steps:

1. Initially, the informal narration of the protocol is specified in cIP on the top of which the security property is formalised by means of a $\mathcal{PL}$ formula as shown in step 3;
2. the connection $\mathcal{PL}$ formula is specified in order to explicitly represent "secrets' sharing";
3. the intruder knowledge is set according to the power that the verifier grants to the intruder, for instance public/private keys or messages acquired in previous run of the protocol;
4. the automatic phase of the verification starts. $\mathcal{A}$SPASyA is invoked and, depending on the results, steps 2 and 3 are iterated.

It is necessary to define cIP principals specifying their open variables. About step 1, even though it seems that the formalisation of the narrative informal specification into cIP processes can hardly be done in a fully automatic manner, we can give some rule of thumbs:

– Initiator usually needs an open variable which the responder should join;
– if the identity of the partner is acquired in a communication, then no open variable should be necessary about the identity of partner (unless checks on the identity are required);
– an open variable might be necessary when a principal must interact with a server, to share a secret (e.g. a key or a name);
– each step of the informal specification corresponds to an input action for the receiver and an output action for the sender. A binding occurrence of a variable is needed each time a principal receives a name it is stored in a binding variable.

The formalisation of security properties is a complex task and requires experience. The logic $\mathcal{PL}$ can formalise:

– The impossibility for the intruder to know a particular datum d in a run of the protocol, when checking for secrecy properties,
– the relations between variables of different principals that must hold in every run, when more complex properties, like authentication, must be verified.

For instance, we equip the join primitive with *connection formulae* that are $\mathcal{PL}$ formulae constraining the possible ways in which principals can be connected. Connection formulae can be thought of as being contexts invariant that principals must guarantee when they join a context. We remark that the join operation, together with the connection formula, is a distinguished feature supporting a very expressive mechanism for driving the verification phase.

The join operation and the connection formulae can also be seen as a coordination mechanism for programming and modelling the interactions of processes in an open systems where components dynamically access running contexts by connecting to other participants. For instance, considering *web services*, the components are the services and new services are built by properly connecting existing components. In the current practice, this is statically done by the programmer; while the join operation and the connection formula would allow the programmers to specify the constraints over the dynamic connections of their components.

In step 2 a connection formula is specified; the $\mathcal{PL}$ formula that details the connection constraints aims at twofold purpose:

– First, it specifies the assumptions on the keys shared between principals, and
– second, it prunes the state space allowing reducing the set of traces to be checked.

Sometimes, the result of a verification session reveals that some assumptions on the protocol have not been correctly formalised. By refining the hypothesis on principal connections, incorrect attacks can be filtered out. Basically, connection formulae can be exploited for focussing the verification on those states that are interesting to the verifier. For instance, later on we will assume that all the communications with a trusted server in the KSL protocol are safe and thence we state a formula that properly specifies the connections among the server and the other participants.

Step 3 specifies the initial knowledge of the intruder to test a protocol under weaker conditions. It is used mainly for two purposes:

– Let the intruder know some secrets (e.g. compromised keys) enhancing its attacking power, for instance to test the robustness of the protocol;
– Let the intruder know something about past interactions between principals (cryptograms exchanged in previous sessions).

The latter is especially useful in finding *replay attacks* where the intruder exploits messages appeared in previous session.

Steps 2, 3 and 4 can be iterated in order to tune the connection conditions, and the initial knowledge, according to the results obtained in previous iterations.

In order to explain the methodology, we apply it to the analysis of the KSL protocol [19] (a simplification of Kerberos [20]). The goal of KSL is the repeated authentication between principals A and B that exploit a trusted server S. The protocol is divided into two parts: An initial message exchange to establish a session key between principals, followed by the repeated authentication part. Repeated authentication is performed by means of an expiring ticket generated by B for A. Until the ticket is valid (not expired), A can re-authenticate itself with B without requesting a new session key from S. The informal specification of KSL is as follows:

$$
\begin{array}{l}
(1)\ A \rightarrow B : na, A \\
(2)\ B \rightarrow S : na, A, nb, B \\
(3)\ S \rightarrow B : \{nb, A, k^{ab}\}_{k^{bs}}, \{na, B, k^{ab}\}_{k^{as}} \\
(4)\ B \rightarrow A : \{na, B, k^{ab}\}_{k^{as}}, \{Tb, A, k^{ab}\}_{k^{bb}}, nc, \{na\}_{k^{ab}} \\
(5)\ A \rightarrow B : \{nc\}_{k^{ab}} \\
\\
(6)\ A \rightarrow B : ma, \{Tb, A, k^{ab}\}_{k^{bb}} \\
(7)\ B \rightarrow A : mb, \{ma\}_{k^{ab}} \\
(8)\ A \rightarrow B : \{mb\}_{k^{ab}}
\end{array}
$$

Messages $(1 \div 5)$ are the key exchange part whereas messages $(6 \div 8)$ are the repeated authentication. Namely, each further interaction between A and B starts from message (6). Server S shares a symmetric key with each principal. Initiator A generates a nonce na, and sends it to B. Then B asks S for a new session key, S generates k^{ab} and, in (3), encrypts the session key in two cryptograms $\{nb, A, k^{ab}\}_{k^{bs}}$ and $\{na, B, k^{ab}\}_{k^{as}}$ sent to B. Notice that here it is implicitly assumed that k^{as} (resp. k^{bs}) is known only to S and A (resp. B). After decrypting $\{nb, A, k^{ab}\}_{k^{bs}}$, B assumes that k^{ab} is the fresh session key generated by S and meant to be shared with A (freshness of k^{ab} is enforced by nonce nb).

Message (4) is quite complex and crucial; B sends to A a message containing:

- the cryptogram $\{na, B, k^{ab}\}_{k^{as}}$, generated by S on message (3),
- the "ticket" $\{Tb, A, k^{ab}\}_{k^{bb}}$,
- a new nonce nc and
- the nonce na encrypted with k^{ab}.

The ticket is a cryptogram sealed with a key k^{bb} that only B knows and will be used in the second part of KSL for achieving repeated authentication; apart from the identity of A, it contains a *generalized time-stamp*[1] and the session key so that B can check the validity of the ticket. The nonce nc will be used to prove to B that A really asked for the session key k^{ab}, while the cryptogram $\{na\}_{k^{ab}}$ is generated to grant A that B has acquired k^{ab}. Message (5) closes the first part of KSL: A sends back nc encrypted with k^{ab} so that B is granted that A acquired the session key.

Principal A knowing k^{ab} and the ticket issued by B can re-authenticate itself performing messages $(6 \div 8)$. In (6), B receives a nonce, ma, and the ticket that B has previously generated for A. If the ticket is valid, B sends ma encrypted with k^{ab} to A together with a new nonce mb, used to ensure the identity of A in message (8).

The first step of our methodology prescribes to give the cIP formalisation of each role of the protocol and the formalisation of the property of interest. We first describe the first

[1] A generalised time-stamp reports the current time of the local clock of B, an indication of lifetime and an "epoch" identifier to protect B against replay attacks. Refer to [27] for problems related to time-stamps.

phase of the protocol, i.e messages ($1 \div 5$), in order to explain the linguistic features of cIP wrt the formalisation of cryptographic protocols. Later, we will describe the rest of the protocol according to a verification session. This allows us to concentrate on the main peculiarity of the two phases and also shows how the methodology accomplishes with the intuitions behind the verification. The principals are

$$
\begin{aligned}
S \triangleq\ & (a, ak, b, bk) \\
& [in(?cna, a, ?cnb, b). \\
& out(\{cnb, a, kab\}_{bk}, \{cna, b, kab\}_{ak}) \ldots]
\end{aligned}
$$

$$
\begin{aligned}
A \triangleq\ & (b, sk) \\
& [out(na, A). \\
& in(\{na, b, ?r\}_{sk}, ?tkb, ?bn, \{na\}_r). \\
& out(\{bn\}_r) \ldots]
\end{aligned}
\qquad
\begin{aligned}
B \triangleq\ & (sk\ ) \\
& [in(?cn, ?u). \\
& out(cn, u, nb, B). \\
& in(\{nb, u, ?r\}_{sk}, ?tka). \\
& out(tka, \{nt, u, r\}_{kbb}, nc, \{cn\}_r). \\
& in(\{nc\}_r) \ldots]
\end{aligned}
$$

It is important to emphasize the use of open variables. Principal S has variables a and b respectively for the identity of the initiator and the responder. The server needs two further variables, ak and bk that are meant to store the symmetric keys that S shares with A and B, respectively. Similarly, A and B use sk for storing the keys they share with S. Notice that variables are distinct in different principals.

KSL tries to achieve repeated mutual authentication; informally this means that each time B (connected to S) terminates a run of the protocol, thinking to have interacted with a A (connected to S) then A has recently executed a session with B and A actually has been the partner in the communications, and viceversa. A $\mathcal{PL}$ formula that formalises the authentication of A to B is:

$$
\begin{aligned}
\psi_{KSL} \triangleq \forall B.l : \exists S.i : \exists A.j : b_i = B_l \ \wedge \ u_l = A_j \ \wedge \ b_j = B_l \ \wedge \ a_i = A_j \rightarrow \\
cna_i = na_j \ \wedge \ cnb_i = nb_l \ \wedge \\
r_j = kab_i \ \wedge \ r_l = kab_i \ \wedge \\
cn_l = na_j \ \wedge \ nc_l = bn_j .
\end{aligned}
$$

Formula ψ_{KSL} states that any instance B_l is attached to an instance of the server template S_i ($b_i = B_l$) and that, if B_l believes to have talked with an instance A_j ($u_l = A_j$) connected as initiator to the same server ($a_i = A_j$) and aimed at authenticating itself to B_l ($b_j = B_l$) then a "correct" data exchange should take place, namely:

- The server receives the correct nonces ($cnb_i = nb_l$ and $cna_i = na_j$);
- both B_l and A_j obtain the same session key generated by S_i ($r_j = kab_i$ and $r_l = kab_i$);
- finally, the nonces received by B_l are all generated by A_j ($nc_l = bn_j$ and $cn_l = na_j$).

The second step of our methodology requires to specify a connection formula ϕ_{KSL}, that we indicate with ϕ_{KSL}. It can be exploited to state assumptions that the protocols informally uses. For instance, KSL assumes that the server shares private keys with the initiator and the responder. In $\mathcal{PL}$

$$\phi_{KSL} \triangleq \forall S.i : \exists A.j : (a_i = A_j \rightarrow ak_i = sk_j) \ \wedge \ \exists B.l : (b_i = B_l \rightarrow bk_i = sk_l)$$

According to ϕ_{KSL}, for every S_i (instance of the server) there is a correspondence between the name of the initiator (resp. responder) and the open variable holding the shared key. Notice that ϕ_{KSL} excludes that an instance of A can act as responder. In order to allow this possibility, we can consider the following formula:

$$
\begin{aligned}
\phi'_{KSL} \triangleq \forall S.i : \exists A.j : (a_i = A_j \rightarrow ak_i = sk_j \ \wedge \ b_i = A_j \rightarrow bk_i = sk_j) \ \wedge \\
\exists B.l : (b_i = B_l \rightarrow bk_i = sk_l \ \wedge \ a_i = B_l \rightarrow ak_i = sk_l),
\end{aligned}
$$

however, we stick to the simpler ϕ_{KSL} for our verification.

Table 1 reports the results of the verification for the first phase of KSL. We checked various scenarios by varying the number of instances and the possible connection formulae. Even though no attack has been discovered in this phase, it is worth noticing how the

	3 Instances			4 Instances		
Join	Configurations	Time	Attacks	Configurations	Time	Attacks
true	10240	58	0	-	-	-
ϕ_{KSL}	550	12	0	13218	4:21	0
ϕ'_{KSL}	590	34	0	15723	5:07	0

Table 1. Attack report for the first phase of KSL

connection formulae can help in reducing the size of the state space. For instance, in the case of three instances of principals, the number of states in a completely unconstrained setting (first row) is very large compared to the case where one of ϕ_{KSL} or ϕ'_{KSL} is used. This is even more evident when considering the case of four instances; in fact, the verification with the trivial connection formula requires a unreasonable amount of time (one day on a 2.4MHz Athlon processor), whereas both ϕ_{KSL} and ϕ'_{KSL} terminate in few minutes.

Since the first phase of the protocol does not yield any attack, we perform the verification of KSL by focussing on the second phase of the protocol, i.e., on the messages $(6 \div 8)$. We consider correct the session key exchange phase, and check whether an attack can be built during the repeated authentication phase. Under this hypothesis, at the end of the 5-th message of KSL, we can safely assume that

- the intruder is aware of the initial five messages of the session; therefore, the ticket is in its knowledge (among other data);
- A (resp. B) thinks that B (resp. A) is running a session protocol as responder (resp. initiator);
- A and B share a key in the current session;
- the key is valid in virtue of a certificate issued by B.

Since, we are assuming that the interactions of A and B with S are trusted and not compromised, we can consider only the principals for A and B:

$$A \triangleq (b, sk, tk)[out(nma, \{b, A, sk\}_{tk}).in(?mb, \{nma\}_{sk}).out(\{mb\}_{sk})]$$
$$B \triangleq (a, sk, tk)[in(?ma, \{B, a, sk\}_{tk}).out(nmb, \{ma\}_{sk}).in(\{nmb\}_{sk})];$$

where, A and B use the open variable tk for modelling the sharing of the ticket acquired by A in the first phase. Indeed, notice that A never uses tk as decryption key (i.e., in an input action) since the key encrypting the ticket is known only to B; indeed, A simply uses tk in the first output for communicating the ticket to B.

The assumptions on the secrets shared by A and B can be formally stated by the following connection formula:

$$\bar{\phi}_{KSL} = \exists B.l : \exists A.j : tk_j = tk_l \rightarrow b_j = a_l \wedge sk_j = sk_l,$$

that states that if there are two instances of A and B sharing a ticket, then they aim at communicating one another ($b_j = a_l$) and share a session key ($sk_j = sk_l$).

The authentication formula to be checked can be stated similarly to what done for ψ_{KSL}, but it is simpler than ψ_{KSL} because we can ignore the communications with the server:

$$\bar{\psi}_{KSL} \triangleq \forall B.l : \exists A.j : b_j = B_l \ \wedge \ a_l = A_j \rightarrow ma_l = nma_j \ \wedge \ mb_j = nmb_l.$$

Since the verification of KSL with two instances does not yield any attack (as reported in Table 2), we focus on the case with three participants.

In the third step we must specify the intruder's knowledge. We have already pointed out that the intruder is aware of those messages exchanged in the first phase, hence, the initial knowledge κ_0 contains the following messages:

$I,$	always known
$B_1, B_2, A_3,$	the instances that joined the session
$\{B_1, A_3, sk_1\}_{tk_1},$	ticket issued by B_1 for A_3.

The previous messages contain variables (e.g., sk_1, tk_1,...) that will be instantiated during the generation of the initial contexts by means of the join operation. We can think of κ_0 as a template for a knowledge that compactly specify a set of messages depending of the join operation. In this case, $\mathcal{A}$SPASyA finds (and reports) the following attack (among others):

$$
\begin{array}{l}
(1)\ A_3 \to I : nma_3, \{B_2, A_3, ks\}_{kb2} \\
(2)\ I \to B_2 : nma_3, \{B_2, A_3, ks\}_{kb2} \\
(3)\ B_2 \to I : nmb_2, \{nma_3\}_{ks} \\
(4)\ I \to B_1 : nmb_2, \{B_1, A_3, ks\}_{kb1} \\
(5)\ B_1 \to I : nmb_1, \{nmb_2\}_{ks} \\
(6)\ I \to B_2 : \{nmb_2\}_{ks} \\
(7)\ I \to A_3 : nmb_1, \{nma_3\}_{ks} \\
(8)\ A_3 \to I : \{nmb_1\}_{ks} \\
(9)\ I \to B_1 : \{nmb_1\}_{ks}
\end{array}
$$

In messages $(1 \div 3)$, A_3 and B_2 begin the authentication phase; the communications are possible because of the ticket in κ_0. In messages $(4 \div 5)$, I, playing the role of A_3, uses B_1 for encrypting nmb_2 with ks. At this point, I can match the input data requested by B_2 and can subsequently, playing the role of B_2, use A_3 as an encrypting oracle to obtain $\{nmb_1\}_{ks}$ which is needed to end the protocol run with B_1. Hence the intruder has been able to let B_1 believe he was interacting with A_3 while he was interacting with I that violates the authentication property.

Let us remark that the attack is possible because there is a trace that start from a context where the join has assigned the same session keys for the two different tickets (the one in κ_0 and the other generated by A_3 in (1)). Observe that nothing prevents this neither in ψ_{KSL} nor in $\bar{\phi}_{KSL}$; therefore, we could repeat the verification by imposing this condition (that is indeed, required by the informal specification of KSL). Nevertheless, this is the hypothesis imposed in the analysis of KSL reported in [21] where the same attack has been firstly reported; this analysis is anyway motivated by considering the robustness of a protocol in presence of weaker assumptions (two tickets that contains the same session key) that are also realistic.

As a final attempt, we check whether enriching the intruder's knowledge, KSL has new flaws. Let us iterate the verification algorithm with the initial knowledge

$$\bar{\kappa}_0 = \kappa_0 \cup \{\{B_2, A_3, sk_2\}_{tk_2}\}$$

that corresponds to the fact that the intruder has collected the ticket generated by B_2 for A_3. Table 2 collects the results of the verification. First, observe that there is no difference in using κ_0 or $\bar{\kappa}_0$; the reason for this is that the extra message added to κ_0 is generated by the instance A in any case. Another aspect to remark is that, looking at the table in the case of 3 instances, it seems that $\bar{\phi}_{KSL}$ cuts off some attacks. However, by analysing the reported attacks it is possible to recognise that the extra attacks found with the trivial connection formula *true* are special cases of the attack presented previously. Indeed, in those attacks, the two instances of B use the same session key and the same key for encrypting the tickets (that is unrealistic). Moreover, the reported attacks basically correspond to "permutations"

	2 Instances			3 Instances			4 Instances		
Join/Knowl.	Conf.	Time	Attacks	Conf.	Time	Attacks	Conf.	Time	Attacks
$true, \kappa_0$	104	0.69	0	3878	1.53	8	-	-	-
$true, \bar{\kappa}_0$	104	0.85	0	3878	1.89	8	130870	2:27	16
$\bar{\phi}_{KSL}, \kappa_0$	71	0.64	0	3220	1.50	6	-	-	-
$\bar{\phi}_{KSL}, \bar{\kappa}_0$	71	0.80	0	3220	1.85	6	52692	1:16	12

Table 2. Attack report for KSL repeated authentication part

of the attack shown above, namely, they are the same attack where a different indexing of the instances is used.

We conclude by emphasising the advantage of using join and connection formulae; indeed, Table 1 and the last two rows of Table 2, show how the generated state space is dramatically reduced by exploiting a non trivial connection formula, while discovering the same set of attacks[2].

5 Comparisons

We briefly constrast our framework with some verification approaches (and their related tools) for security protocols based on model checking. The most important ones (up to our knowledge) are in [5, 34, 29, 24, 22, 29]. Our analysis will be more focused on methodological aspects rather than on efficiency issues, because different semantics have been exploited by different frameworks.

The approach in [22] has many similarities with our framework in modeling security protocols as open systems. There the openness is represented by a non-completely specified and extendible context. Principals are expressed in CCS calculus (equipped with cryptographic primitives) and properties are also given in a suitable logic. The main differences of our framework wrt [22] are represented by the open variables and the join primitive that, together with the connection formulae, can be seen as a coordination mechanism for open systems. Moreover we exploit symbolic techniques to shrink the state space.

In [5, 34] symbolic techniques for generating and analysing traces have been described. They are based on dialects of the π-calculus for principal representation while properties are stated as *correspondence assertions*; in [34] assertions are embedded in principal definitions, violating separation of concerns (changing the property to be checked leads to a re-formalisation of principal definitions). We separated more clearly the specification of principals from that of security properties. Both [5, 34] lack the possibility of template definition, hence every principal instance has to be specified by hand, which may be long and error prone, and impact on the formalisation of protocols and properties. For instance, many protocol assumptions depend on the initial knowledge and secrets sharing and must be explicitly stated in [5, 34]. For instance, we analysed the KSL protocol using TRUST and, in order to find the attack reported in Section 4, we had to explicitly state that the tickets must have the same session key, while the join mechanism of $\mathcal{A}$SPASyA automatically generates and find the flawed context. Both [5, 34] offer the possibility to specify the initial knowledge of the intruder but without any parameterisation (as done in Section 4).

The approaches in [29, 24] are based on the *strand space* model presented in [31, 32, 11]. Properties are expressed in terms of connections between strands of different kinds. A strand can be parameterised with variables and a trace is generated by finding a substitution for which an interaction graph exists. Principals are represented with terms of a free algebra

[2] The number of attacks are doubled with respect to the case of three instances because there are the same attacks where the two instances of A are swapped in the attacks.

whereas properties are specified by a suitable logic. Both approaches provide devices very similar to the join mechanism of our framework but there is not the possibility for the user to impose constraints on principal connections. Initial knowledge specification is given by adding data to the strand space, and can be fully parameterised with variables.

	Number of states			Times		
Protocol	$\mathcal{A}$SPASyA	TRUST	STA	$\mathcal{A}$SPASyA	TRUST	STA
NS (2 instances)	55	328	24	0.7	0.06	0.07
KSL (2 instances)	39	135	33	0.8	0.04	0.04
KSL (4 instances)	21742	69875	-	43	1.8	-

Table 3. Comparing $\mathcal{A}$SPASyA

Regarding efficiency issues, the amount of time used by $\mathcal{A}$SPASyA is comparable to those used by STA and TRUST (Table 3). $\mathcal{A}$SPASyA is a bit slower than others because it consumes time in generating initial contexts and checking connection formulae. However, STA stops as soon as the first attack trace is found whereas TRUST and $\mathcal{A}$SPASyA perform a search over the whole state space. TRUST generates the largest state space, mainly because it is based on a small step semantics. $\mathcal{A}$SPASyA and STA have more compact state spaces whose difference lies in the fact that $\mathcal{A}$SPASyA initially applies the join mechanism.

6 Conclusions

We have addressed the problem of security protocol analysis taking inspiration from an approach oriented to a more general framework. We have proposed a verification methodology, and the $\mathcal{A}$SPASyAtool which supports it, and presented the results of some practical experimentations.

The methodology tries to limit as much as possible the sources of errors in the formalisation process by keeping the different aspects of the formalisation into clearly separated steps. Importantly, and distinguishing from others approaches, the protocol and the property specification are clearly separate. Moreover, once the principal behaviours are given in the first step of the methodology, they remain unchanged. Several verification parameters can be finely tuned by the verifier, mostly in an intuitive way, like the search space by means of connection formulas in step two, and the power of the intruder in step three. Principal connections can be constrained by means of a $\mathcal{PL}$ formula and the intruder power can be augmented by adding information to its initial knowledge, allowing for discovering attacks where the intruder exploits information about previous sessions of the protocol, and for testing the robustness of protocols under unexpected conditions.

The automatic phase of verification is performed at a cost comparable with similar state of the art tools, also thanks to mechanisms for the selective pruning of the state space. Experimentally, we have applied the methodology to the verification of several protocols, some of which have been illustrated in this paper, detecting all the known flaws (like the one recently reported in [4], found with techniques not based on model checking).

In order to enhance our methodology, we are planning to extend the framework to handle non atomic keys, hashing functions and time-marked names. Moreover, along the line of connection formulas, we believe that security properties can be exploited as heuristic strategies to "guide" the state exploration towards states that more likely make the property false.

Finally, we would like to extend the open variables and the constrained join mechanism, now based on connection formulas, to the more the general case of open system verification, where open variables represent resources, and the join is constrained by formulas aimed at guarantee more general composition properties, as initially suggested in [7].

References

1. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.
2. Roberto Amadio, Denis Lugiez, and Vincent Vancackére. On the symbolic reduction of processes with cryptographic functions. Technical report, INRIA-Sophia, 2001.
3. L. Andrade, J. Fiadeiro, J. Gouveia, G. Koutsoukos, and M. Wermelinger. Coordination for orchestration. In F. Arbab and C. Talcott, editors, *Proc. 5th International Conference on Coordination Models and Languages, (COORDINATION 2002)*, volume 2315 of *LNCS*, pages 5–13, York, 2002. Springer.
4. Chiara Bodei, Michael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis can find new flaws too. In *Workshop on Issues on the Theory of Security (WITS'04), Barcelona, 2004, ENTCS, Elsevier*, 2004.
5. Michele Boreale. Symbolic trace analysis of cryptographic protocols. In *28th Colloquium on Automata, Languages and Programming (ICALP)*, LNCS. Springer, July 2001.
6. Michele Boreale and Marzia Buscemi. A framework for the analysis of security protocols. In *CONCUR: 13th International Conference on Concurrency Theory.* LNCS, Springer-Verlag, 2002.
7. A. Bracciali, A. Brogi, G. Ferrari, and E. Tuosto. Security and dynamic composition of open systems. In F. Arbab and R. Menezes, editors, *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA02)*, pages 1372–1377, Las Vegas, NV, 2002. CSREA Press.
8. Andrea Bracciali. *Behavioural Patterns and Software Composition.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 2003. available at `http://www.di.unipi.it/~phd`.
9. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
10. N. Carriero and D. Gelernter. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, 1992.
11. Cervesato, Durgin, Mitchell, Lincoln, and Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop.* IEEE Computer Society Press, 2000.
12. Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Using state space exploration and a nautural deduction style message derivation engine to verify security protocols. In *In Proc.* IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998*, 1998.
13. E.M. Clarke, J.M. Wing, et al. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, Dec. 1996.
14. F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S.Weerawarana. The next step in web services. *Communications of the ACM*, 46(10):29–34, Oct. 2003.
15. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEETIT: IEEE Transactions on Information Theory*, 29, 1983.
16. Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol — version 3.0. Available at `http://home.netscape.com/eng/ssl3/ssl-toc.html`, March 1996.
17. D. Garlan and D. E. Perry. Introduction to the special issue on software architecture. *Software Engineering*, 21(4):269–274, 1995.
18. Dan Harkins and Dave Carrel. RFC 2409: The Internet Key Exchange (IKE), November 1998. Status: PROPOSED STANDARD.
19. Axel Kehne, Jürgen Schönwälder, and Horst Langendörfer. Multiple authentications with a nonce-based protocol using generalized timestamps. In *Proc. ICCC '92*, Genua, 1992.
20. J. Kohl and B. Neuman. The kerberos network authentication service (version 5). Internet Request for Comment RFC-1510, 1993.
21. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
22. Fabio Martinelli. Analysis of security protocols as open systems. *TCS*, 209(1):1057–1106, 2003.
23. Alfred J. Menzies, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

24. Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.
25. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.
26. John C. Mitchell, Mark Mitchell, and Ulrich Ster. Automated analysis of cryptographic protocols using murϕ. In *10th IEEE Computer Security Foundations Workshop, IEEE Press*, pages 141–151, 1997.
27. Clifford Neumann and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, 1993.
28. M. Shaw. *Architectural Requirements for Computing with Coalitions of Resources*, 1999. Position paper for First Working IFIP Conference on Software Architecture. Available at `http://www.cs.cmu.edu/~Vit/paper_abstracts/ShawCoalitions_paper.html`.
29. Dawn. Song, Sergey Berezin, and Adrian Perrig. Athena, a novel approach to efficient automatic security protocol analysis. *Computer Security*, 9(1,2):47–74, 2001.
30. Douglas R. Stinson. *Cryptography: Theory and practice.* CRC Press, 1995.
31. J. Thayer, J. Herzog, and J. Guttman. Honest ideals on strand spaces. In *11th IEEE Computer Security Foundations Workshop*, 1998.
32. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct, 1999.
33. Emilio Tuosto. *Non-Functional Aspects of Wide Area Network Programming.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 2003.
34. Vincent Vanackére. The trust protocol analyser. automatic and efficient verification of cryptographic protocols. In *VERIFY02*, 2002.
35. Tatu Ylonen. SSH — secure login connections over the Internet. In USENIX Association, editor, *6th USENIX Security Symposium, July 22–25, 1996. San Jose, CA*, pages 37–42, Berkeley, CA, USA, July 1996. USENIX.