Compositional Analysis of Entity Authentication Protocols *

Michele Bugliesi, Riccardo Focardi **, and Matteo Maffei

Dipartimento di Informatica, Università Ca' Foscari di Venezia,
Via Torino 155, I-30172 Mestre (Ve), Italy
{michele,focardi,maffei}@dsi.unive.it

Technical Report CS-2003-12

Abstract. We propose a new method for the static analysis of entity authentication protocols. Our approach draws on our earlier work [6] on a role-based characterization of the formal properties of this class of protocols. We develop our approach based on a dialect of the spi-calculus as the underlying formalism for expressing protocol narrations. Our analysis validates the honest protocol participants against static (hence decidable) and easily checked conditions that provide formal guarantees of entity authentication. The main result is that the validation of each component is provably sound and fully compositional: if all the protocol participants are successfully validated, then the protocol as a whole guarantees entity authentication in the presence of Dolev-Yao intruders.

Keywords: Authentication, Formal Methods in Security, Cryptographic Protocols, Process Calculi.

1 Introduction

Security protocols are designed to provide diverse security guarantees in possibly hostile environments: typical guarantees include the secrecy of a message exchange between two trusted entities, the freshness and authenticity of a message, the authenticity of a claimed identity, ... and more.

The presence of hostile entities makes protocol design complex and often error prone, as shown by many attacks to long standing protocols reported in the literature (see, e.g., [9, 10, 15, 20, 24, 26]). In most cases, such attacks dwell on flaws in the protocols' logic, rather than on breaches in the underlying cryptosystem. Indeed, even when cryptography is assumed as a fully reliable building-block, an intruder can engage a number of potentially dangerous actions, notably, intercepting/replaying/forging messages, to break the intended protocol invariants.

Formal methods have proved very successful as tools for protocol design and validations. On the one hand, failures to model-check protocols against formal specifications has lead to the discovery of several attacks (see, e.g., [20, 22, 25, 26, 19]). On the other hand, static techniques, based on type systems and control-flow analyses have proved effective in providing static guarantees of correctness [14, 13, 4]. Static analysis is also at the basis of our present technique: we target the analysis of (shared-key) *entity authentication* protocols, i.e. protocols that enable one entity to prove its presence and claimed identity to a remote party [17, 21, 11].

The technique we introduce here draws on earlier work on entity authentication which we developed in [6], and briefly review next. There are three key ingredients to that approach. First, we identify a minimal set of roles played by the participants in an entity authentication session. Secondly, we envision a tagging mechanism to decorate certain messages, enciphered under long-term keys, with explicit information about the entities involved in their (the messages') exchange, and the entities' rôle in the protocol. Based on the additional information conveyed by rôles, we then isolate a

* Work partially supported by MURST project 'Modelli formali per la sicurezza' and EU Contract IST-2001-32617 'Models and Types for Security in Mobile Distributed Systems' (MyThS).

** Contact Author.

core set of invariants on the (dynamic) behavior of the honest participants which imply the desired authentication guarantees. More precisely, we prove that formal authentication guarantees can be made for any protocol whose honest participants behave according to our invariants, irrespective of the presence of (Dolev-Yao) intruders [1]. Technically, this corresponds to a safety theorem showing that all the traces of protocol runs generated by compliant honest principals, and Dolev-Yao intruders, satisfy a *correspondence assertion* property [30] that implies the desired authentication guarantees.

Overview and main results. The purpose of our present endeavour is to turn the safety theorem we proved in [6] into an effective method for analysis. We accomplish that by providing decidable sufficient conditions for the behavioral invariants that imply the safety theorem of [6]. Such conditions arise here as the result of a static, and fully compositional, analysis of the honest protocol participants. By relying on the safety theorem of [6], we then prove that our static analysis provides formal guarantees of correctness for entity authentication protocols.

There are two building blocks in our present approach. First we introduce a new process calculus, named ρ-spi calculus, that we use as the underlying formalism for specifying the authentication protocols of interest. ρ-spi is a dialect of the spi-calculus [2], with roles and other authentication-specific constructs that we introduce to support and ease our analysis. Some of the new constructs are directly inspired by the design of the process calculus for protocol narrations *Lysa* [4], In particular, like *Lysa*, ρ-spi provides for the explicit representation of all the protocol assumptions that are left implicit in traditional protocol narrations. We formalize the dynamics of the calculus in terms of a trace semantics defined in the style proposed in [5].

Secondly, we develop a static technique for the analysis of entity authentication protocols encoded as terms of ρ-spi. The analysis is carried out by means of proof rules that validate the logical structure of each principals. Intuitively, the constraints enforced by the proof rules constitute the static counterparts of the dynamic conditions underlying the behavioral invariants identified in [6].

Notably, the analysis is carried out in isolation on each participant: each validated principal is decreed *locally correct*. Our main result, then, is that protocols consisting of the composition of *locally correct* principals are safe. This is proved in two steps. First we show that the trace semantics of any such protocol only contains traces that respect the behavioral invariants of [6]. Then, based on the safety theorem in *loc. cit.*, we have a proof that the protocol conveys the intended authentication guarantees. In addition, by compositionality, we may replicate each principal unboundedly many times while still preserving the safety of the system. Hence, a safety proof for one protocol session is also a safety proof for an unbounded number of sessions.

Our approach is interesting in several respects. Verifying the conditions of local correctness is intuitively simple, efficient —in fact, linear in the size of the process being examined— and fully automatic. In addition, the compositional nature of the analysis makes it applicable for validating unboundedly many protocol sessions. The only human effort needed for the analysis is left in the encoding of the protocol narration in ρ-spi calculus, which in turn requires one to identify the roles played by the participants in the entity authentication session. As we will show, this is a relatively simple task.

Related Work. In [13, 14] Gordon and Jeffrey define a type system for the analysis of authentication protocols (with symmetric and asymmetric encryption) expressed in a dialect of the spi calculus. Their approach is related to our work in several aspects. As ours, their authentication analysis is a safety analysis based on correspondence assertions. Also, as in our approach, the reasoning is guided

[1] We implicitly appeal to the standard perfect-cryptography assumptions throughout. Dolev-Yao intruders may intercept, reply and forge new messages, but never decrypt messages without knowing the corresponding keys.

by a tagging mechanism for messages that provides enough redundancy for the analysis to identify and distinguish different messages. There are, however, important differences. Their tags are applied uniformly, so as to distinguish any pair of encrypted messages, and are used by a dependent type system to form traces of dependent effects which are then inspected to verify protocols by checking a correspondence assertion property similar to ours. Their type analysis is inherently compositional, as ours, but the safety result requires a further inspection of the effects associated with a complete protocol trace. Our tagging mechanism is less demanding, and does not impose a unique identity on each message: indeed, we only need to tag the different roles played by the entities in the authentication session (each role has always the same tag). In addition, our tags seem to be more informative, as they make it possible to formulate safety judgments based on the inspection of separate participants.

The recent work by Bodei *et al.* on a control-flow analysis for message authentication in *Lysa* [4] is also strongly related to our present approach. The motivations and goals, however, are different, since message authentication concerns the origin of a message, while entity authentication also provides guarantees about the presence of a given entity during an authentication session [17, 21, 11]. Indeed, our analysis provides a strong form of entity authentication guarantee [29, 18]: not only the presence of the claimant is analyzed, but also its intention to authenticate itself with the verifier.

Further related work is based on employing belief logics [28], like, e.g., BAN [7], GNY [12], as formal tools for reasoning about authentication guarantees provided by messages. The main difference with these papers is that we do not have any kind of protocol idealization. Indeed, we reason about 'concrete' execution traces, directly connected to the structure of ciphertexts. An interesting paper that goes in the direction of eliminating the idealization step in analyses based on (belief) logics is [23] by Durgin, Mitchell and Pavlovic. They propose a logic for security protocols which is designed around a process calculus derived from Strand Spaces, and built on top of an asymmetric encryption system. As in our framework, the semantics of the calculus is formalized in terms of a trace semantics. The formalization of authentication is, however, largely different, and not easily compared to ours.

Structure of the paper. Section 2 gives a more detailed review of our earlier work in [6]. Section 3 introduces the process calculus, ρ-spi that we employ for the analysis and presents the operational semantics of this calculus. Section 4, gives a formal description of our static analysis, and its main properties. Section 5, presents a case study and illustrates the analysis on the nonce-based Wide Mouthed Frog Protocol. Section 6 concludes the presentation with final remarks. A separate appendix collects some additional technical definitions.

2 Authentication Roles

A typical source of flaws in security protocols, and specifically in authentication protocols, is a poor interpretation of messages, whereby certain messages are believed to convey more guarantees than they actually do. As observed in [3] "every message should say what it means, i.e., its interpretation should depend only on its content". As a simple instance of a message exchange that fails to comply with this principle consider $A \to B : \{\text{“Here I am!”}\}_{K_{AB}}$. Here *Alice* is sending to *Bob* the message "Here I am!" encrypted with a long-term key shared between them. When Bob receives the message, he could be misled into believing that the message has been generated by Alice since it is encrypted with a key that only Alice (besides Bob himself) knows. However, this is not true if Bob previously ran the same protocol with Alice (with exchanged roles). In that case, the following, well-known, *reflection attack* could be exploited by an intruder E:

$$\begin{array}{ll} a) & B \to E(A) : \{\text{“Here I am!”}\}_{K_{AB}} \\ b) & E(A) \to B : \{\text{“Here I am!”}\}_{K_{AB}} \end{array}$$

In the first run a, E pretends to be A (denoted with $E(A)$) and intercepts the message sent by Bob; in the second run b, $E(A)$ replays the same message back to Bob. As a consequence, B erroneously interprets his own message as sent by Alice. The problem is that Bob is assuming that the message has been generated by Alice without that being explicitly indicated in the message. A simple solution is to provide this additional information within the message, as in $A \rightarrow B : \{A, \text{"Here I am!"}\}_{K_{AB}}$, where the first component now signals that A is the 'claimant' of the protocol.

Our approach in [6] is best understood as a generalization of this idea: we envision a tagging mechanism for messages that helps identify their 'function' in the authentication task, and makes the interpretation of their components unambiguous. The technique also enforces positional constraints for the components of certain messages —called *authentication blocks* in [6]— that constitute the core of the authentication sessions. Specifically, such messages should always be encrypted with long-terms keys, and should contain an entity identifier together with the entity's role in the authentication task. We identify three fundamental roles: (i) *Claimant*, the role of a principal willing to authenticate itself to some other principal; (ii) *Intended verifier*: the role of a principal that a claimant wants to act as a verifier for its response; (iii) *Key owner*: the role of a principal that owns a session key. As we show in [6], these roles apply well, and uniformly, for a wide class of entity authentication protocols.

There are other message components needed to make the role-based reasoning effective. In particular, each role needs a time-dependent element, like e.g., nonces, timestamps, sequence numbers, that provides *freshness*. Here we only consider *nonces*, i.e., numbers that are used only once: they are typically generated as challenges, and sent back inside encrypted messages, to be *checked* and thus prove that a certain message is not a replay of an old one. Together with nonces, we associate entities with the session key they possess (this is always true of entities playing the *Key owner* role). This is useful for distribution of session keys, where a trusted server wants to communicate that a certain entity owns a certain session key.

Collectively, the tagging technique presupposes the following structure for the authentication blocks of the protocol: $\{\mathsf{R}, I, N, K_s, \dots\}_K$, where K is a long-term key, R is one of the three roles $(i) - (iii)$ above, I an entity identifier, N a nonce, and K_s, if present, a session key. There are no constraints on the remaining components of the ciphertext, indicated by $\dots$, as they serve no purpose for the authentication analysis. Based on this structural information, we then isolate certain invariants on the behavior of the honest protocol participants (the only entities with knowledge of long-term keys) that enforce the following, provably reliable, interpretation for authentication blocks. Assume a principal decrypts $\{\mathsf{R}, I, N, K_s, \dots\}_K$, and checks the nonce N. Then, that principal is guaranteed that "Entity I is recently playing the role R with respect to the entities that share the long-term key K, and is possessing the fresh key K_s (if K_s is present)". As we mentioned in the Introduction, the analysis we develop in the present paper provides similar guarantees, but now implied by static conditions on the structure of the honest protocol participants. The rest of the paper shows how that is accomplished.

3 The ρ-spi calculus

We introduce the process calculus upon which we develop our static technique, The ρ-spi calculus derives from the spi calculus [2], and inherits many of the features of *Lysa*, a version of the spi calculus proposed in [4] for the analysis of authentication protocols. ρ-spi differs from both calculi in several respects: (i) it incorporates the notion of role-based message exchange from [6], (ii) it provides primitives for declaring process identities and (shared) long-term keys, (iii) it guarantees the correct use of long-term keys by checking that only the authorized entities may use them, (iv) it provides new useful authentication-specific constructs.

Table 1 The syntax of ρ-spi calculus.

n ::= *Names* $\mathcal{N} = \mathcal{LN} \cup \mathcal{EN}$
- a,b,k local names $\mathcal{LN}$
- $\underline{a},\underline{b},\underline{k}$ environment names $\mathcal{EN}$

I ::= *Identities* $\mathcal{ID} = I_{\mathcal{P}} \cup I_{\mathcal{T}} \subseteq \mathcal{EN}$
- A,B principal identities $I_{\mathcal{P}}$
- T TTP identities $I_{\mathcal{T}}$
- R roles $\{\mathsf{Claim},\mathsf{Owner},\mathsf{Verif}\} \subseteq \mathcal{EN}$

N ::= *Variables or names* $\mathcal{V} \cup \mathcal{N}$
- x,y,z variables $\mathcal{V}$
- n names $\mathcal{N}$

M ::= *Messages* $\mathcal{M}$
- N variables or names
- $\{M_1,\ldots,M_m\}_E$ encrypted message

E ::= *Expressions* $\mathcal{E}$
- k_l long-term keys $\mathcal{K_L}$
- M messages

S ::= *Sequential processes*

$\mathbf{0}$	(nil)
in($N_1,\ldots,N_j;x_{j+1},\ldots,x_m$).$S$	(input)
out($N_1,\ldots,N_m$).S	(output)
new(a) S	(restriction)
decrypt x as $\{N_1,\ldots,N_j;x_{j+1},\ldots,x_m\}_K.S$	(decryption)
encrypt $\{N_1,\ldots,N_m\}_K$ as $x.S$	(encryption)
run(I_1,I_2).S	(run)
commit(I_1,I_2)	(commit)

P,Q ::= *Processes*

$I : S$	(sequential process with identity)
$P \vert Q$	(parallel composition)
key(k_l,I_1,I_2).P	(key assignment)

Syntax. The syntax of the calculus is presented in Table 1. We presuppose three countable sets: $\mathcal{N}$, $\mathcal{K_L}$ and $\mathcal{V}$, representing *names*, *long-term keys* and *variables*, respectively. As done in [5], the set $\mathcal{N}$ of names is partitioned into two countable sets, a set $\mathcal{LN}$ of *local names*, ranged over by a,b and k, and a set $\mathcal{EN}$ of *environment names*, ranged over by $\underline{a},\underline{b}$ and $\underline{k}$. The two sets $\mathcal{LN}$ and $\mathcal{EN}$ contain the names initially known only by principals (i.e., secret), and the ones known by the environment (i.e.,public), respectively. There are two special categories of public names: *roles*, composed of the three special names $\mathsf{Claim},\mathsf{Owner},\mathsf{Verif}$, and *identities* $\mathcal{ID}$, further partitioned in *principals* $I_{\mathcal{P}}$ and *trusted third parties* (TTPs) $I_{\mathcal{T}}$. The set $\mathcal{K_L}$, ranged over by k_l, represents long-term keys. *Messages* are variables, names, and ciphertexts or the form $\{M_1,\ldots,M_m\}_E$, where the M_i's are messages and E is an *expression*, that is a long-term key or a message. Note that long-term keys are not messages, hence they cannot be exchanged. Instead, the syntactic structure of messages allows arbitrary nesting of encryptions, and the use of encrypted messages as keys. However, as we shall see, the proof technique of Section 4 imposes a more disciplined use of keys in the authentication blocks of a protocol. We often use K, with subscripts, to represent 'atomic' keys drawn on the set $\mathcal{N} \cup \mathcal{V} \cup \mathcal{K_L}$.

Processes (or *protocols*), ranged over by P,Q are the parallel composition of *sequential processes*, ranged over by S. Every sequential process is tagged by an identity I. A sequential process may never fork into two parallel processes. This assumption helps assigning unique identities in $\mathcal{ID}$ to (sequential) processes and it is justified by observing that principals, such as initiators, responders and TTPs, can be specified as sequential processes, possibly sharing some long-term keys. To allow the sharing of long-term keys among sequential processes, we provide ρ-spi with the primitive key(k_l,I_1,I_2) to declare a long-term key k_l shared between I_1 and I_2.

We use a number of notation conventions. The restriction operator is a binder for names, while the input and decryption prefixes are binders for the variables that occur to the right of the semicolon; in all case the scope of the binders is the continuation process. The notions of free/bound names and variables arise as expected. We write $P[M/x]$ for the result of substituting M for all the free occurrences of x in P. Moreover, we identify processes up to renaming of bound variables and names, i.e., up to α-equivalence, and only consider processes $\mathcal{P}$ that are closed, i.e., that have no free variables.

The constructs for input, output, new and decryption are essentially the same as in the calculus *Lysa*. In particular, we presuppose a unique (anonymous) public channel from/to which messages are read/sent (this differs from the spi calculus where different (even private) channels may be used to communicate). As observed in [4], the presence of just one public channel models well the worst-case situation of an intruder with a complete control of the network. Another difference with respect to the spi calculus , is that our input and decryption prefixes may test part of the read and decrypted message through a simple pattern-matching mechanism. This is useful, and convenient, to express nonce-checks that typically arise in *nonce-based* challenges. A semi-colon is used to separate the components to be pattern matched from those to be substituted. The initialization construct declares the identifier of a sequential process whereas the two special actions 'run' and 'commit' are used to check the *correspondence assertions* as in [13].

Unlike the spi calculus and Lysa, in ρ-spi we introduce an explicit construct for encryption. In particular, the syntactic structure of ρ-spi requires that all the encrypted messages be explicitly formed in an encryption prefix, and forbids ciphertexts to be sent directly on the output channel. Thus, for instance, the process encrypt $\{M\}_K$ as x.out(x).$\mathbf{0}$ is well-formed, while out($\{M\}_K$).S is not. Notice that these conditions are purely static, as encrypt $\{M\}_K$ as x.out(x).$\mathbf{0}$ reduces to out($\{M\}_K$).$\mathbf{0}$ in one computation step. Similar observations apply to the process forms for input and decryption, where we require the terms used for pattern matching to be atomic (i.e. variables or names). These syntactic restrictions are useful for our static analysis, as they ease the reasoning based on the structural inspection of the encrypted messages of a protocol. We remark that, as a result of these restrictions the set of syntactically legal terms is a (strict) subset of the set of terms that arise at run time (cf. Table 7).

The intuitive reading of the process forms is as follows. Process $\mathbf{0}$ is the null process that does nothing; process *in*($N_1,\dots,N_j;x_{j+1},\dots,x_m$).$S$ reads a message composed of m parts, requiring that the first j match values $N_1,\dots,N_j$; in that case it continues as S with $x_{j+1},\dots,x_m$ substituted by the remaining $m-j$ input messages. Process *out*($N_1,\dots,N_m$).S outputs a messages composed of m parts and then continues as S. Process *new(a)* S generates a fresh local name a to be used by S. Process *decrypt x as* $\{N_1,\dots,N_j;x_{j+1},\dots,x_m\}_K$.$S$ tries to decrypt x with key K; if successful, it matches the first j components of the deciphered text against $N_1,\dots,N_j$, and if these matches also are successful continues as S with $x_{j+1},\dots,x_m$ substituted by the remaining $m-j$ components of the deciphered text. Process *encrypt* $\{N_1,\dots,N_m\}_K$ *as x.S* binds variable x to the encrypted message $\{N_1,\dots,N_m\}_K$. in the continuation S. The process forms *run*(I_1,I_2).S and *commit*(I_1,I_2) declare that the sequential process I_1 is starting, respectively committing, a protocol session with I_2.

Example 1. We illustrate ρ-spi with the following simple (flawed) authentication protocol:

$$\begin{array}{l} 1)\ B \to A : N \\ 2)\ A \to B : \{N, M\}_{K_{AB}} \end{array}$$

We assume K_{AB} to be known only by A and B and N to be a fresh nonce generated by B. The intention of the protocol is to give guarantee to B that the last message has been (recently) generated by A as only A should be able to encrypt the freshly generated nonce N. This protocol can be simply formalized in our calculus as shown in Table 2. Process *Initiator*, declares its name A and generates a fresh message $M \in \mathcal{LN}$. After receiving x, it signals the start of a new authentication run with B, encrypts x and M with the long term key K_{AB}, and then sends out the encrypted message. Similarly, *Responder* declares its name B, generates a fresh nonce $n_B \in \mathcal{LN}$ and sends it out. Then, it reads y from the net and decrypts it with the long term key K_{AB}, checking the nonce n_B (through the pattern-matching

Table 2 A sample protocol narration ρ-spi

Protocol	$\triangleq$ key$(K_{AB},A,B).(A : \mathit{Initiator}(A,B) \mid B : \mathit{Responder}(B,A))$
Initiator(A,B)	$\triangleq$ new(M).in$(;x)$.run(A,B).encrypt$\{x,M\}_{K_{AB}}$ as y.out(y)
Responder(B,A)	$\triangleq$ new(n_B).out(n_B).in$(;y)$.decrypt y as $\{n_B;z\}_{K_{AB}}$.commit(B,A)

mechanism of decryption). If the match is successful, the variable z gets bound to a message (from A) and the principal commits through commit(B,A).

Notice that we are only modeling one protocol session. However, as we will see, multiple sessions can be easily achieved by just replicating the *Initiator* and *Responder* processes.

Operational Semantics. We define the operational semantics of ρ-spi in terms of *traces*, after [5]. A trace is a closed sequence of *actions*. We use ε to denote the empty trace, and symbol :: to denote concatenation. Every construct of the calculus generates a corresponding action, Specifically, the action *key*(k_l,I_1,I_2) is generated by key assignment, *in*$(N_1,\ldots,N_m)$ by input, *out*$(N_1,\ldots,N_m)$ by output, *new(a)* by restriction, *I encrypts* $\{M_1,\ldots,M_m\}_K$ by encryption and *I decrypts* $\{M_1,\ldots,M_m\}_K$ by decryption, *run(A,B)* and *commit(A,B)* by 'run' and 'commit'. The actions for encryption and decryption are the only ones to specify the entity identifier I of the process that performs the corresponding operation. The set of all possible actions is denoted with Act.

The trace semantics is given in terms of a transition relation $\longrightarrow$ between *configurations*, i.e., between pairs $\langle s,P\rangle$, where $s \in Act^*$ is a trace, P is a (closed) process. The transition relation on configurations simulates the execution of a command of P and records the corresponding action t by adding it to s. The effect is to reach a new configuration $\langle s :: t,P'\rangle$, where P' is P after executing t. As previously explained, every sequential process S is tagged by an identity I. The transition relation preserves such a tag, that is rules on sequential processes have the form $\langle s,I : c.S\rangle \longrightarrow \langle s',I : S\rangle$. This is useful, since, as showed in Appendix A, most of the semantic rules use such a tag. For instance, as mentioned above, the action generated by an encrypt construct is parameterized by the identity attached to the sequential process in the initial configuration: if c=encrypt x as $\{a\}_k$ then s'=s::I encrypts $\{a\}_k$.

The transition semantics is defined formally in Table 7. Most of the rules should be easily understood: we give detailed explanations in Appendix A. As usual, the intruder is modeled implicitly in the transition rules, by encoding it with rules that formalize the possible ways to manipulate a message at disposal to an intruder. In particular, notice that the INPUT rule requires that any messages read by a process must be computable using the output generated so far using the available keys. The rules for message manipulation formalize the power of a Dolev-Yao intruder, as expected.

Definition 1 (Traces). *The set $T(P)$ of traces of process P is the set of all the traces that may be generated by a finite sequence of transitions from the configuration $\langle \varepsilon,P\rangle$. Formally, $T(P) = \{s \mid \exists P'$ s.t. $\langle \varepsilon,P\rangle \longrightarrow^* \langle s,P'\rangle\}$*

Example 2. Consider again process *Protocol* of Example 1. The following, is a possible trace of such a process:

$$\text{key}(k_{AB},A,B) :: \text{new}(M) :: \text{new}(n_B) :: \text{out}(n_B) :: \text{in}(n_B) :: \text{run}(A,B) :: A \text{ encrypts } \{n_B,M\}_{K_{AB}} ::$$
$$\text{out}(\{n_B,M\}_{K_{AB}}) :: \text{in}(\{n_B,M\}_{K_{AB}}) :: B \text{ decrypts } \{n_B,M\}_{K_{AB}} :: \text{commit}(B,A)$$

Notice that this trace represents only one of the possible interleavings of *Initiator* and *Responder* actions [2] In this particular trace, commit(B,A) is preceded by run(A,B). According to the Woo and Lam idea of correspondence assertions [29], this trace is safe, since B is convinced to communicate with A (action commit(B,A)) and A is indeed running the protocol with B (action run(A,B)).

This protocol is not correct when multiple sessions (with A and B running both as Initiator and as Responder) are considered, as it suffers of a *reflection* attack. As mentioned above, multiple sessions can be modeled by replicating processes *Initiator* and *Responder*. For example, to model two protocol sessions, we may consider the following process:

$$\begin{aligned} Protocol_2 \triangleq \quad & \mathrm{key}(K_{AB},A,B).\ (A : Initiator(A,B) \mid B : Responder(B,A) \mid \\ & \qquad B : Initiator(B,A) \mid A : Responder(A,B)) \end{aligned}$$

Notice that A and B run both as Initiator and as Responder. Indeed, the new processes $Initiator(B,A)$ and $Responder(A,B)$ model B running as Initiator and A running as Responder, respectively. Consider the following trace for $Protocol_2$:

$$\begin{aligned} &\mathrm{key}(k_{AB},A,B) :: \mathrm{new}(M) :: \mathrm{new}(n_B) :: \mathrm{out}(n_B) :: \mathrm{in}(n_B) :: \mathrm{run}(\mathbf{B},\mathbf{A}) :: \mathbf{B} \text{ encrypts } \{n_B,M\}_{K_{AB}} :: \\ &\quad \mathrm{out}(\{n_B,M\}_{K_{AB}}) :: \mathrm{in}(\{n_B,M\}_{K_{AB}}) :: B \text{ decrypts } \{n_B,M\}_{K_{AB}} :: \mathrm{commit}(B,A) \end{aligned}$$

It is the same as above, except that B is running as initiator instead of A (as pointed out in bold font). Indeed B, is running the protocol with himself while A is doing nothing. Notice that the attack is revealed by the absence of correspondence: commit(B,A) is not matched by any run(A,B). The trace corresponds to the following (well-known) reflection attack:

$$\begin{aligned} &\text{1.a) } B \rightarrow E(A) : N_B \\ &\text{1.b) } E(A) \rightarrow B : N_B \\ &\text{2.a) } B \rightarrow E(A) : \{M,N_B\}_{K_{AB}} \\ &\text{2.b) } E(A) \rightarrow B : \{M,N_B\}_{K_{AB}} \end{aligned}$$

We conclude this section by formalizing the definition of safety based on correspondence assertions. As in [13], we say that a trace is *safe* if every $commit(B,A)$ is preceded by a distinct $run(A,B)$. Moreover a protocol guarantees *entity authentication* if all of its traces are safe. Formally:

Definition 2 (Safety). *A trace s is safe if and only if whenever $s = s_1 :: commit(B,A) :: s_2$, then $s_1 = s_1' :: run(A,B) :: s_1''$, and $s_1' :: s_1'' :: s_2$ is safe. A process P is safe if, $\forall s \in T(P), s$ is safe.*

Notice that the trace corresponding to the attack discussed in Example 2 is not safe since commit(B,A) is not matched by run(A,B).

4 A Compositional Proof Technique

The proof technique applies only to protocol narrations that may be represented as ρ-spi processes of the form

$$\overrightarrow{\mathrm{keys}}.(I_1 : S_1 | \ldots | I_m : S_m)$$

where $\overrightarrow{\mathrm{keys}}$ represent a sequence of key assignments $\mathrm{key}(k_1,I_1,I_1'),\ldots,\mathrm{key}(k_z,I_z,I_z')$ and $S_1,\ldots,S_m$ are m sequential processes. As we noted earlier, the protocols of interest may indeed be represented as

[2] As observed, the semantic rules guarantee that a local name can never be read if it has not been previously sent out. For example, the sequence key(k_{AB},A,B)::new(M)::new(n_B)::in(n_B):: ... does not represent a valid trace, as in(n_B) is performed before out(n_B).

processes in the above form: hence there is no significant loss of expressive power in our choice of a specific process format. In addition, we make the assumption that the processes are alpha-renamed so as to remove any clash among different names and variables. The analysis proceeds by examining each single process $\overrightarrow{\text{keys}}.I_1 : S_i$, and attempting to validate each of the S_i under the key assignment determined by the prefix $\overrightarrow{\text{keys}}$.

The analysis is formalized by means of the proof rules collected in Table 3. We denote with σ a string $c_1.c_2.\ldots.c_m$ of ρ-spi sequential constructs (i.e., all constructs except parallel composition) and we write $c \in \sigma$ to denote that construct c belongs to the sequence σ. Moreover, we implicitly assume a symmetry for long-term key assignments and we write key(k_l, I_1, I_2)$\in \sigma$ to require that either key(k_l, I_1, I_2) or key(k_l, I_2, I_1) belongs to σ. The proof rules derive judgements of the form $I \models \sigma$ (read 'σ is correct with respect to identity I') where σ is any sequence of syntactically legal constructs from ρ-spi (but note that σ itself need not be a ρ-spi process, since it does not necessarily terminate with a **0** or a commit). The proof rules exploit the structural constraints on encrypted messages enforced by the tagging mechanism we illustrated in Section 2. We remark here that the positional constraints help identifying the authentication blocks of the protocol. Specifically, if k_l is a long-term key, we know that a message $\{M_1, \ldots, M_n\}_{k_l}$, is an authentication block if M_1 is a role: in that case, we also know that M_2 is an identity, M_3 is a nonce and M_4 *may* be a session key, depending on the specific value of M_1.

We consider two possible ways for a principal A to authenticate another principal B:

AUTHENTICATION 1 A decrypts (and checks the nonce of) a message $\{\mathsf{Claim}, B, a, \ldots\}_{k_l}$, i.e., a message in which B plays the claimant role; this message must have been encrypted either by a TTP or by B;

AUTHENTICATION 2 A decrypts (and checks the nonce of) a message $\{\mathsf{Owner}, B, a, k, \ldots\}_{k_l}$, encrypted by a TTP, declaring that k is a fresh session key owned by B; then, A decrypts at least one message $\{N_1 \ldots N_m\}_k$ that she did not generate.

Notice that these guarantees require a very careful use of roles inside encrypted messages. In particular, the message $\{\mathsf{Claim}, B, a, \ldots\}_{k_l}$ should be generated only if B is indeed willing to authenticate with A; similarly, $\{\mathsf{Owner}, B, a, k, \ldots\}_{k_l}$ should only be generated if k is a fresh session key shared between A and B. This 'intended' use of roles is enforced by six invariants, or *principles*, that predicate the generation of encrypted messages with roles to appropriate conditions, which we summarize informally below[3]
Principles 1, 2 and 6 specify how A may declare her willingness to authenticate with B:

PRINCIPLE 1 A sends message $\{\mathsf{Claim}, A, a, \ldots\}_{k_{AB}}$ directly to B using a long-term key k_{AB} she shares with B;

PRINCIPLE 2 A sends a message $\{\mathsf{Verif}, B, a, \ldots\}_{k_{AT}}$ to a TTP T using a long-term key K_{AT} shared with T; notice that the role Verif is used, as expected, to inform a TTP about the intention of authenticating with someone else;

PRINCIPLE 6 A may send a message $\{N_1, \ldots, N_m\}_k$, if she has decrypted (and checked the nonce of) a message $\{\mathsf{Owner}, B, a, k, \ldots\}_{k_{AT}}$, declaring that k is a fresh key shared with B; again, K_{AT} is a long-term key shared with T.

Principles 3, 4 and 5 define safe behaviors of TTPs. Principles 3 and 4, regulate the generation of messages of the form $\{\mathsf{Claim}, A, a, \ldots\}_{k_l}$ by requiring that

[3] The numbering of the principles is taken from [6]. This is why they are not ordered in Table 3

PRINCIPLE 3 A TTP may generate a message $\{\mathsf{Claim}, A, a, \ldots\}_{K_{BT}}$ if it has previously decrypted (and checked the nonce b of) a message of the form $\{\mathsf{Verif}, B, b, \ldots\}_{k_{AT}}$, and K_{AT} and K_{BT} are shared with A and B respectively;

PRINCIPLE 4 A TTP may generate a message $\{\mathsf{Claim}, A, a, \ldots\}_{K_{BT}}$ if it has previously decrypted (without checking the nonce) a message of the form $\{\mathsf{Verif}, B, a, \ldots\}_{k_{AT}}$, and K_{AT} and K_{BT} are shared with A and B respectively; notice that a is forwarded to B so that B can check the freshness of the first message;

PRINCIPLE 5 TTPs declare new session keys through messages of the form $\{\mathsf{Owner}, I, a, k, \ldots\}_{k_l}$.

The correctness of session-key distribution is provided by a further condition (see Definition 4) requiring that session keys need to be freshly generated (through construct new) and can be distributed to at most two different principals.

We now describe more in detail how the rules of Table 3 work. The only axiom is INIT which states that every sequence of key assignments $\overrightarrow{\text{keys}}$ is correct with respect to the identity I, given that it is followed by a sequential process tagged by I. The other rules are all of the form

$$\textsc{Rule name}\quad \frac{I \models \sigma \qquad \text{(side conditions)}}{I \models \sigma.c}$$

Given a legal string σ, they establish the conditions by which the construct c may legally be appended to σ. In particular, rule RUN has no hypotheses: construct run can be always added to σ. Then we have eight rules corresponding to AUTHENTICATION 1 and 2 plus the six principles described informally above. Notice that, in all such rules, to check that some action has been previously engaged, we ask that the corresponding construct be in σ. For example, AUTHENTICATION 1 allows A to commit with B only if A has previously decrypted a message $\{\mathsf{Claim}, B, a; \ldots\}_{k_l}$ and k_l is shared either with T or with B (the constraint "k_l is shared between I and I'" becomes key$(k_l, I, I') \in \sigma$). The last constraint of AUTHENTICATION 2:

$$\text{encrypt } \{N'_1, \ldots, N'_m\}_y \text{ as } z \in \sigma \text{ implies } \exists i \in [1, j] \text{ s.t. } N_i, N'_i \in \mathcal{N},\ N_i \neq N'_i$$

is used to guarantee that A did not generate the message $\{N_1, \ldots, N_j, x_{j+1}, \ldots, x_m\}_y$ she is decrypting. The constraint states that whenever A has generated a message encrypted with the same key y, and with the same number of components m, then the message must differ in at least one (bound) component N_i.[4] Principles 1,2 and 6 represents the start of new runs, thus they all require that run(A, B) has been previously executed. The fact that run should be the last action performed guarantees that these principles are applied only once for each protocol run. Principles 3,4, and 5 are developed similarly and do not contain particularly complicate constraints. Rules NEW, OUTPUT and INPUT do not have any side conditions. Decryption is handled through two rules:

DECRYPTION is for decryptions in which no nonce-check is present; notice that whenever the message is an authentication block (N_1 is a role) then the rule requires that the number of matched values in decryption is less then 3; since the nonce is the third value this means that this decryption never checks nonces;

DECRYPTION WITH CHECK is for decryptions in which a nonce is checked; notice that name a, in third position, is before the semi-colon meaning that it should be matched; the rule requires that a has not been previously checked and that a is a freshly generated name.

[4] A typical example of this situation is when one principal sends an encrypted nonce $\{N_A\}_K$ and expects to receive back the encrypted nonce incremented by one, i.e., $\{N_A + 1\}_K$.

Finally, rule ENCRYPTION is used to account for all the encryptions that do not fall in previous cases. The only constraint is related to the special case of session-key encryptions that can be performed without following PRINCIPLE 6: this can happen only if such a principle has been previously applied for another encryption with the same session key. This is captured by requiring that $\text{run}(A,B).\text{encrypt}\ \{\ldots\}_K$ as $x \in \sigma$.

As mentioned above, in addition to complying with the constraints imposed by the proof rules, a process $\overrightarrow{\text{keys}}.I : S$ should also handle session keys with great care. In particular if the process is TTP, then it should distribute session keys to two principals, by way of two corresponding suitable authentication blocks with role Owner. No other principal should ever send out session keys, either in clear (of course) or within ciphertexts.

Definition 3 (Session Key Handling). *We say a process $\overrightarrow{keys}.I : S$* handles session keys safely *with respect to identity I if:*

Case $I \in I_{\mathcal{P}}$ **(I is a Principal)** *: if decrypt y as $\{\gamma \quad y,\ldots\}_{k_l} \in S$, with $0 \leq j \leq 3$, and y occurs in construct $c \in S$, then it must be either c = encrypt $\{N_1,\ldots,N_m\}_y$ as y or c = decrypt z as $\{N_1,\ldots,N_m\}_y$, and y should not occur in $N_1,\ldots,N_m$. γ is defined as in Table 3.*

Case $I \in I_{\mathcal{T}}$ **(I is a TTP)** *if encrypt $\{\mathsf{Owner},B,x_1,k,\ldots\}_{k_{l_1}}$ as $z \in S$ then (i) is of the form: $\sigma.new(k).\sigma'$. encrypt $\{\mathsf{Owner},B,x_1,k,\ldots\}_{k_{l_1}}$ as $z.\sigma''$. encrypt $\{\mathsf{Owner},A,x_2,k,\ldots\}_{k_{l_2}}$ as y. σ''', (ii) k does not occur in $\sigma.\sigma'.\sigma''.\sigma'''$, and (iii) key(k_{l_1},I,A)$\in \overrightarrow{keys}$ and key(k_{l_2},I,B)$\in \overrightarrow{keys}$.*

Local Correctness, for a sequential process, requires that the sequence of the component constructs be validated by the proof rules, and that the process handle session keys safely.

Definition 4 (Local Correctness). *A process $\overrightarrow{keys}.I : S$ is* locally correct *if $I \models I : \overrightarrow{keys}.S$ is derivable, and $\overrightarrow{keys}.I : S$ handles sessions keys safely with respect to I.*

Finally, a process is correct if all of its sequential components are locally correct.

Definition 5 (Process Correctness). *Let P be a process of the form $\overrightarrow{keys}.(I_1 : S_1|\ldots|I_m : S_m)$. P is* correct *if the process $\overrightarrow{keys}.I_i : S_i$ is locally correct, for all $i = 1,\ldots,m$.*

Our main results states that correctness, in the sense of Definition 5, is a sufficient condition for safety (Definition 2) and, consequently, is a sufficient condition to guarantee entity authentication. Formally:

Theorem 1 (Traces Safety). *Let P be a ρ-spi process. If P is correct, then it is safe.*

Proof (sketch). The proof is based on showing that the static principles of local correctness imply the dynamic invariants on traces, defined in [6]. For lack of space, we only give the sketch of the proof. In [6] dynamic invariants are formulated through a judgment $\vdash$. Intuitively if $\vdash s$ then the trace s preserves the dynamic invariants. The main theorem states that $\vdash s$ implies the safety of s. Thus, to prove our theorem it is sufficient to show that if P is correct, then every trace s of P is such that $\vdash s$. This would directly prove the thesis, i.e., that s is safe. We proceed as follows, by induction of the length of s:

(i) the empty trace ε is, of course, safe;

(ii) we consider the last action c of a non-empty trace $s = s' :: c$ and we consider the invariant on s' that should hold because of the execution of c. This invariant requires some property on s'; we consider the sequential process S_i that has executed c. Since S_i is locally correct (by hypothesis), we obtain that one of the static principles has been applied to prove the correctness of the construct that

Table 3 Local Correctness

Claimant and Verifier Rules

RUN

$$\frac{A \models \sigma}{A \models \sigma.\mathrm{run}(A,B)}$$

AUTHENTICATION 1

$$\frac{A \models \sigma \qquad \text{decrypt } x \text{ as } \{\mathsf{Claim},B,a;\ldots\}_{k_l} \in \sigma \qquad \mathrm{key}(k_l,A,T) \vee \mathrm{key}(k_l,A,B) \in \sigma}{A \models \sigma.\mathrm{commit}(A,B)}$$

AUTHENTICATION 2

$$\frac{\begin{array}{c} A \models \sigma \qquad \text{decrypt } x \text{ as } \{\mathsf{Owner},B,a;y,\ldots\}_{k_l} \in \sigma \qquad \mathrm{key}(k_l,A,T) \in \sigma \\ \text{decrypt } z \text{ as } \{N_1,\ldots,N_j;x_{j+1},\ldots,x_m\}_y \in \sigma \qquad \text{encrypt } \{N'_1,\ldots,N'_m\}_y \text{ as } z \in \sigma \text{ implies } \exists i \in [1,j] \text{ s.t. } N_i,N'_i \in \mathcal{N}, N_i \neq N'_i \end{array}}{A \models \sigma.\mathrm{commit}(A,B)}$$

PRINCIPLE 1

$$\frac{A \models \sigma \qquad \sigma = \sigma'.\mathrm{run}(A,B) \qquad \mathrm{key}(k_l,A,B) \in \sigma}{A \models \sigma.\text{encrypt } \{\mathsf{Claim},A,x,\ldots\}_{k_l} \text{ as } y}$$

PRINCIPLE 2

$$\frac{A \models \sigma \qquad \sigma = \sigma'.\mathrm{run}(A,B) \qquad \mathrm{key}(k_l,A,T) \in \sigma}{A \models \sigma.\text{encrypt } \{\mathsf{Verif},B,x,\ldots\}_{k_l} \text{ as } y}$$

PRINCIPLE 6

$$\frac{\begin{array}{c} A \models \sigma \\ N_1 \notin \{\mathsf{Claim},\mathsf{Verif},\mathsf{Owner}\} \qquad \sigma = \sigma'.\mathrm{run}(A,B) \qquad \text{decrypt } z \text{ as } \{\mathsf{Owner},B,a;x,\ldots\}_{k_l} \in \sigma \qquad \mathrm{key}(k_l,A,T) \in \sigma \end{array}}{A \models \sigma.\text{encrypt } \{N_1,\ldots,N_m\}_x \text{ as } y}$$

TTP Rules

PRINCIPLE 3

$$\frac{\begin{array}{c} T \models \sigma \qquad \text{decrypt } x \text{ as } \{\mathsf{Verif},B,a;\ldots\}_{k_{l2}} \in \sigma \\ \text{encrypt } \{\mathsf{Claim},A,y,\ldots\}_{k_{l1}} \text{ as } y \notin \sigma \qquad \mathrm{key}(k_{l1},B,T) \in \sigma \qquad \mathrm{key}(k_{l2},A,T) \in \sigma \end{array}}{T \models \sigma.\text{encrypt } \{\mathsf{Claim},A,x,\ldots\}_{k_{l1}} \text{ as } z}$$

PRINCIPLE 4

$$\frac{T \models \sigma \qquad \text{decrypt } y \text{ as } \{\mathsf{Verif},B;x,\ldots\}_{k_{l2}} \in \sigma \qquad \mathrm{key}(k_{l1},B,T) \in \sigma \text{ and } \mathrm{key}(k_{l2},A,T) \in \sigma}{T \models \sigma.\text{encrypt } \{\mathsf{Claim},A,x,\ldots\}_{k_{l1}} \text{ as } y}$$

PRINCIPLE 5

$$\frac{T \models \sigma}{T \models \sigma.\text{encrypt } \{\mathsf{Owner},A,x,k,\ldots\}_{k_l} \text{ as } y}$$

Generic Principal Rules

INIT

$$I \models \overrightarrow{\mathrm{keys}}.I :$$

NEW

$$\frac{I \models \sigma}{I \models \sigma.\mathrm{new}(a)}$$

OUTPUT

$$\frac{I \models \sigma}{I \models \sigma.\mathrm{out}(\ldots)}$$

INPUT

$$\frac{I \models \sigma}{I \models \sigma.\mathrm{in}(\ldots)}$$

NIL

$$\frac{I \models \sigma}{I \models \sigma.\mathbf{0}}$$

DECRYPTION

$$\frac{I \models \sigma \qquad N_1 \in \{\mathsf{Claim},\mathsf{Owner},\mathsf{Verif}\} \text{ implies } j < 3}{I \models \sigma.\text{decrypt } y \text{ as } \{N_1,\ldots,N_j;x_{j+1},\ldots,x_m\}_K}$$

DECRYPTION WITH CHECK

$$\frac{I \models \sigma \qquad \text{decrypt } x \text{ as } \{R',J,a;\ldots\}_{k'_l} \notin \sigma \qquad \mathrm{new}(a) \in \sigma}{I \models \sigma.\text{decrypt } y \text{ as } \{R,A,a;\ldots\}_{k_l}}$$

ENCRYPTION

$$\frac{\begin{array}{c} I \models \sigma \\ N_1 \notin \{\mathsf{Claim},\mathsf{Owner},\mathsf{Verif}\} \qquad (K \in V \wedge \text{decrypt } \{\gamma \quad K,\ldots\}_{k_l} \text{ as } y \in \sigma) \text{ implies } \mathrm{run}(I,B).\text{encrypt } \{\ldots\}_K \text{ as } x \in \sigma \end{array}}{I \models \sigma.\text{encrypt } \{N_1,\ldots,N_m\}_K \text{ as } y}$$

In Rule ENCRYPTION:
$\gamma =; x_1,x_2,x_3$, or $\mathsf{Owner};x_1,x_2$, or $\mathsf{Owner},B;x_1$, or Owner,B,b;
Rules for proving $I \models \sigma : c$ have the same hypotheses of the ones for proving $I \models \sigma.c$

Table 4 ISO Two-steps Unilateral Authentication Protocol

$Protocol_{ISO} \triangleq \text{key}(K_{AB},A,B).(A : Initiator_{ISO}(K_{AB},A,B) \mid B :\ Responder_{ISO}(K_{AB},B,A))$

$Initiator_{ISO}(K_{AB},A,B) \triangleq$		$Responder_{ISO}(K_{AB},B,A) \triangleq$	
$\text{new}(M).$	NEW	$\text{new}(n_B)$	NEW
$\text{in}(;x).$	INPUT	$\text{out}(n_B).$	OUTPUT
$\text{run}(A,B).$	RUN	$\text{in}(;y).$	INPUT
$\text{encrypt}\{\mathsf{Claim},A,x,M\}_{K_{AB}}$ as y.	PRINCIPLE 1	decrypt y as $\{\mathsf{Claim},A,n_B;z\}_{K_{AB}}$	DECRYPTION WITH CHECK
$\text{out}(y)$	OUTPUT	$\text{commit}(B,A)$	AUTHENTICATION 1

generated c. The hypotheses of the principle requires that some other action has or has not been previously performed by S_i. These hypotheses are sufficient to imply the invariant on subtrace s'. (ii) is iterated for all different actions c.

4.1 Principles at work: A simple example

As a first illustration of the analysis, consider again the ρ-spi specification of the protocol in Table 2. As discussed in Example 2, the protocol is subject to attacks when runs in multiple sessions. Indeed, the flaw in the protocol attack is revealed by failure to validate one of its components. The problem is with the responder, as we illustrate showing that the process $\text{key}(K_{AB},A,B).B : Responder(B,A)$ is not locally correct. By definition of $Responder(B,A)$, we have

$$\text{key}(K_{AB},A,B).B : Responder(B,A) =$$
$$\underbrace{\text{key}(K_{AB},A,B)\ .\ B :\ \text{new}(n_B)\ .\ \text{out}(n_B)\ .\ \text{in}(;y)\ .\ \text{decrypt } y \text{ as } \{n_B;z\}_{K_{AB}}}_{\sigma}.\ \text{commit}(B,A)$$

Now observe that in order for $I \models \sigma.commit(B,A)$ to be derivable, one needs $I = B$. However, we easily see that, indeed, $B \not\models \sigma.commit(B,A)$. In fact, for $B \models \sigma\ .\ \text{commit}(B,A)$ to be derivable, by rules AUTHENTICATION 1 and 2, decrypt x as $\{R,A,n_b;\ldots\}_{K_{AB}}$ should belong to σ, with $R \in \{\mathsf{Claim},\mathsf{Owner}\}$, but the only decrypt construct in σ is decrypt y as $\{n_B;z\}_{K_{AB}}$. Thus, $B \not\models \sigma\ .\ \text{commit}(B,A)$.

Notice that local correctness does not fail only because a role is not specified. Indeed, even adding a role $R \in \{\mathsf{Claim},\mathsf{Owner}\}$ to the ciphertext, the hypotheses of the two authentication rules would not hold, since the ciphertext is missing the the identity label A. The attack presented in Section 3 exploits precisely this flaw.

The correct version of this protocol is known as ISO Two-Pass Unilateral Authentication Protocol [16], reported below:

$$\begin{array}{l} 1)\ B \rightarrow A : N \\ 2)\ A \rightarrow B : \{A,N,M\}_{K_{AB}} \end{array}$$

The difference between this protocol and the flawed version is exactly the presence of the identity label A in the ciphertext sent by A. Its specification in ρ-spi is reported in Table 4. Notice that we have already made explicit the role (claimant) of A in the encrypted message. This is the only overhead required by our technique in order to prove that the protocol is correct. Checking $A \models \text{key}(K_{AB},A,B).A : Initiator_{ISO}(K_{AB},A,B)$ and $B \models \text{key}(K_{AB},A,B).B : Responder_{ISO}(K_{AB},B,A)$ is indeed straightforward. The rule names on the right side of every construct of Table 4 refer to the rules applied for verifying local correctness. In particular, for $A \models \text{key}(K_{AB},A,B).A : Initiator_{ISO}(K_{AB},A,B)$, the hypotheses of rules INIT, NEW, INPUT, RUN and OUTPUT are trivially verified, and the only interesting case is PRINCIPLE 1, which is applied on the sequence:

$$\underbrace{\text{key}(K_{AB},A,B)\ .A:\ \text{new}(M)\ .\ \text{in}(;x)\ .\ \text{run}(A,B)}_{\sigma}\ .\ \text{encrypt}\{\mathsf{Claim},A,x,M\}_{K_{AB}}\ \text{as}\ y$$

PRINCIPLE 1 requires that $\sigma = \sigma'\ .\ \text{run}(A,B)$ and that $\text{key}(K_{AB},A,B) \in \sigma$, which are both true. Let us now write key(K_{AB},A,B).B : $Responder_{ISO}(K_{AB},B,A)$ as σ.commit(B,A). AUTHENTICATION 1 hypotheses hold since decrypt x as $\{\mathsf{Claim},A,n_B;z\}_{K_{AB}} \in \sigma$ and key(K_{AB},A,B)$\in \sigma$. Similarly, rule DECRYPTION WITH CHECK can be applied successfully, since n_B is new and is not checked anywhere else. Rules INIT, NEW, OUTPUT and INPUT are trivially satisfied. Thus $B \models$key(K_{AB},A,B).B : $Responder_{ISO}$ (K_{AB},B,A).

Finally, since the protocol uses no session keys, it clearly handles those safely, and hence can be judged correct. By Theorem 1, we directly obtain that every trace of $Protocol_{ISO}$ is safe, thus no authentication attack is possible.

Notice that $Protocol_{ISO}$ models only one protocol session, thus one may argue that attacks could arise on multiple sessions. However this is not true as our technique is fully compositional and guarantees that $Protocol_{ISO}$ is safe also when sequential processes $Initiator_{ISO}$ and $Responder_{ISO}$ are arbitrarily replicated.

The specification can be easily generalized to an arbitrary number m of entities, acting both as initiator and as responder and arbitrarily replicated.

$$\begin{aligned} Protocol_{ISO} - m \triangleq\ & \text{key}^{m}_{\substack{i,j=1\\ i<j}}(K_{ij},I_i,I_j). \\ & |^{m}_{\substack{i,j=1,\\ i\neq j}}(I_i : Initiator_{ISO}(K_{ij},I_i,I_j)|\ldots|I_i : Initiator_{ISO}(K_{ij},I_i,I_j)) \\ & |^{m}_{\substack{i,j=1,\\ i\neq j}}(I_i : Responder_{ISO}(K_{ij},I_i,I_j)|\ldots|I_i : Responder_{ISO}(K_{ij},I_i,I_j)) \end{aligned}$$

K_{ij} represents a long term key shared between entities I_i and I_j. We assume that $K_{ij} = K_{ji}$. For this reason key assignment is done only for keys K_{ij} with $i < j$. $Initiator_{ISO}$ and $Responder_{ISO}$ are arbitrarily replicated for every pair of entities I_i, I_j, using the appropriate long-term key. It is easy to see that the proofs of correctness of processes $I_i : Initiator_{ISO}(K_{ij},I_i,I_j)$ and $I_i : Responder_{ISO}(K_{ij},I_i,I_j)$ are just the same as the ones for $A : Initiator_{ISO}(K_{AB},A,B)$ and $B : Responder_{ISO}(K_{AB},B,A)$, up to a renaming of identities and keys. As before, replicated identical processes does not need to be re-proved correct.

5 Case Study: the nonce-based Wide Mouthed Frog protocol

In this section, we apply our technique to prove the correctness of the nonce-based version of *Wide Mouthed Frog Protocol* (WMF) [1], in its simplified version analysed in [13]. The protocol is composed of the following six messages:

1) $A \rightarrow T : A$
2) $T \rightarrow A : N_T$
3) $A \rightarrow T : A,\{B,K_{AB},N_T\}_{K_{AT}}$
4) $T \rightarrow B : *$
5) $B \rightarrow T : N_B$
6) $T \rightarrow B : \{A,K_{AB},N_B\}_{K_{BT}}$

Table 5 Wide Mouthed Frog Protocol in ρ-spi

$Protocol_{WMF} \triangleq$
$\quad \text{key}^{m}_{i=1}(K_i, I_i, I_0).$
$\quad |^{m}_{\substack{i,j=1\\ i\neq j}} I_i : Initiator_{WMF}(K_i, I_i, I_j) | \ldots | I_i : Initiator_{WMF}(K_i, I_i, I_j)$
$\quad |^{m}_{\substack{i,j=1\\ i\neq j}} I_j : Responder_{WMF}(K_j, I_j, I_i) | \ldots | I_j : Responder_{WMF}(K_j, I_j, I_i)$
$\quad |^{m}_{\substack{i,j=1\\ i\neq j}} I_0 : Server_{WMF}(K_i, I_i, K_j, I_j) | \ldots | I_0 : Server_{WMF}(K_i, I_i, K_j, I_j)$

$Initiator_{WMF}(K_i, I_i, I_j) \triangleq$	
$\text{new}(K_{ij})$	NEW
$\text{out}(I_i).$	OUTPUT
$\text{in}(;x_T).$	INPUT
$\text{run}(I_i, I_j).$	RUN
$\text{encrypt}\{\mathsf{Verif}, I_j, x_T, K_{ij}\}_{K_i}$ as x.	PRINCIPLE 2
$\text{out}(I_i, x)$	OUTPUT

$Server_{WMF}(K_i, I_i, K_j, I_j) \triangleq$	
$\text{new}(n_T)$	NEW
$\text{in}(I_i;).$	INPUT
$\text{out}(n_T).$	OUTPUT
$\text{in}(I_i; x_i).$	INPUT
decrypt x_i as $\{\mathsf{Verif}, I_j, n_T; x_{ij}\}_{K_i}$.	DEC. WITH CHECK
$\text{in}(;x_B).$	INPUT
$\text{encrypt}\{\mathsf{Claim}, I_i, x_B, x_{ij}\}_{K_j}$ as x.	PRINCIPLE 3
$\text{out}(x)$	OUTPUT

$Responder_{WMF}(K_j, I_j, I_i) \triangleq$	
$\text{new}(n_B)$	NEW
$\text{out}(n_B).$	OUTPUT
$\text{in}(;x).$	INPUT
decrypt x as $\{\mathsf{Claim}, I_i, n_B; x_{ab}\}_{K_j}$	DEC. WITH CHECK
$\text{commit}(I_j, I_i)$	AUTHENTICATION 1

The aim of the protocol is to establish a session key K_{AB} between A and B, using a TTP T. Long-term keys K_{AT} and K_{BT} are shared between A and T and between B and T, respectively. The protocol works as follows: in the first message A sends to T her own identity label and, in the second message, T replies to A by sending a fresh nonce N_T. In the third message, A generates a fresh session key K_{AB} and sends the encrypted message $\{B, K_{AB}, N_T\}_{K_{AT}}$ to T, meaning that she is asking T to transport the session-key K_{AB} to B. N_T is the nonce received from T. By checking it, T is ensured about the freshness of the received ciphertext. Identity label A (sent as plaintext) is just a hint for choosing the correct decryption key. The (void) fourth message is a way for T to start the protocol with B. In the fifth message B sends to T a fresh nonce N_B and in the last message T sends to B a ciphertext containing the identity label A, the key K_{AB} and the nonce N_B, used again for ensuring freshness. The last message indicate to B that A is willing to share key K_{AB} with him.

In Table 5 we give a specification of the WMF protocol in ρ-spi . $Protocol_{WMF}$ is divided into two parts: in the former, m long term keys K_i are associated to the principal I_i and to the TTP T through the construct $\text{key}(K_i, I_i, I_0)$ iterated for all $i = 1, \ldots, m$. We assume $K_i \in \mathcal{LN}$, $I_0 \in I_{\mathcal{T}}$ and $I_i \in I_{\mathcal{P}}$ for all $i = 1, \ldots, m$. The latter part consists of the parallel composition of principals and trusted servers. Every principal is parameterized by a specific session partner. Also, every trusted server is parameterized by a specific pair composed of a claimant and a verifier (with the respective keys).

Rules on the right of every construct show how to verify the local correctness of every sequential component. We briefly discuss the most interesting cases:

- $I_i \models \text{key}_{i=1,\ldots,m}(K_i, I_i, I_0).I_i : Initiator_{WMF}(K_i, I_i, I_j)$. The only interesting case is PRINCIPLE 2 , which is satisfied since $\text{run}(I_i, I_j)$ is just before encryption and $\text{key}(K_i, I_i, I_0)$ is performed at the beginning;
- $I_j \models \text{key}_{i=1,\ldots,m}(K_i, I_i, I_0).I_j : Responder_{WMF}(K_j, I_j, I_i)$. The only interesting case is AUTHENTICATION 1 which holds since decrypt x as $\{\mathsf{Claim}, I_i, n_B; x_{ab}\}_{K_j}$ and $\text{key}(K_j, I_j, I_0)$ are previously executed;
- $I_0 \models \text{key}_{i=1,\ldots,m}(K_i, I_i, I_0).I_0 : Server_{WMF}(I_i, I_j)$. Here, PRINCIPLE 3 hypotheses hold since decrypt x_i as $\{\mathsf{Verif}, I_j, n_T; x_{ij}\}_{K_i}$, $\text{key}(K_i, I_i, I_0)$ and $\text{key}(K_j, I_j, I_0)$ are previously executed. Moreover no construct of the form encrypt $\{\mathsf{Claim}, I_i, x', \ldots\}_{K_j}$ as y is present.

Session Key Handling is trivially satisfied, as it only applies to keys distributed by TTPs. Thus, $Protocol_{WMF}$ is correct. Theorem 1 ensures that all the traces generated by $Protocol_{WMF}$ are safe.

6 Conclusion

We have applied our framework to the analysis of various authentication protocols taken from literature. The results of our experiments are displayed in Table 6: as indicated, the analysis validated all the correct versions of the protocols [5], while it consistently failed to validate the flawed versions.

Besides these initial results, our technique seems to be promising especially for the simplicity of the analysis. On the other hand, it is fair to observe that the analysis cannot validate every possible (correct) authentication protocol, since it could be the case that entity authentication is achieved without necessarily combining together the six principles we have considered here. However, the present principles appear general enough to validate many existing protocols and help understand the underlying authentication mechanisms. Furthermore, the framework is scalable as new conditions may be added, if needed to validate new protocols (of course, as long as the conditions may be formulated in a compositional way). Another advantage of our approach is that separating (static) principles and (dynamic) invariants should make the proof of safety easily reusable, even when other specification languages are considered.

Some extensions are desirable for analyzing a wider range of authentication protocols, e.g., protocols based on public-key encryption. We also plan to consider a wider set of shared-key protocols, in order to test more deeply the generality of the approach.

Table 6 Case Studies

Protocols	**Correct**	**Flawed**
Iso Symmetric Key Two-Pass Unilateral Authentication Protocol	X	
Iso Symmetric Key Three-Pass Mutual Authentication Protocol	X	
Andrew Secure RPC Protocol		X
Needham Schroeder Protocol [27]		X
Needham Schroeder Amended Version [24]	X	
Otway and Rees Protocol [7]		X
Carlsen's Secret Key Initiator Protocol [8]	X	
Wide Mouthed Frog Protocol,Nonce Based Version [13]	X	
Woo and Lam Protocol [29]		X
Woo and Lam Amended Version [13]	X	
Yahalom Protocol [12]		X
Yahalom Amended Protocol [6]	X	

References

1. M. Abadi and A. D. Gordon. "A Calculus for Cryptographic Protocols: The Spi Calculus". *Information and Computation*, 148(1):1–70, 1999.
2. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.
3. Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[5] Indeed, the Yahalom Amended Protocol can be analyzed through a slight variant the invariants introduced [6] for dealing with a slightly different mechanism for session key distribution.

4. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *In proceedings of 16th IEEE Computer Security Foundations Workshop (CSFW 16)*, June 2003.
5. Michele Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 01*, volume 2076, pages 667–681. LNCS 2076, Springer Verlag, 2001.
6. M. Bugliesi, R. Focardi, M. Maffei, and F. Tudone. Principles for entity authentication (full version). Technical Report CS-2002-16, Università Ca' Foscari, Venezia, November 2002. http://www.dsi.unive.it/$\sim$focardi/TR/principles.ps.gz. An extended abstract will in Proceedings of the 5th International Conference Perspectives of System Informatics (PSI 2003), July 9-12 2003, Springer LNCS.
7. M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". *Proceedings of the Royal Society of London*, 426(1871):233–271, 1989.
8. U. Carlsen. Generating formal cryptographic protocol specifications. In *IEEE Symposium on Research in Security and Privacy*, pages 137–146. IEEE Computer Society, 1994.
9. J. Clark and J. Jacob. "A Survey of Authentication Protocol Literature: Version 1.0". http://www.cs.york.ac.uk/$\sim$jac/papers/drareview.ps.gz, November 1997.
10. R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of ICALP'00*, pages 354–372. Springer LNCS 1853, July 2000.
11. D. Gollmann. "What do we mean by Entity Authentication". In *Proceedings of the 1996 Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society Press, 1996.
12. Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.
13. A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In 14th IEEE Computer Security Foundations Workshop (CSFW-14),pages 145-159, June 2001.
14. Andrew D. Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. In *15th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 77–91, Cape Breton, Canada, 24–26 June 2002. IEEE Computer Society Press.
15. James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *13th IEEE Computer Security Foundations Workshop — CSFW'00*, pages 255–268, Cambridge, UK, 3–5 July 2000. IEEE Computer Society Press.
16. ISO/IEC. "Entity Authentication Using Symmetric Techniques". Report ISO/IEC JTC1.27.02.2 (20.03.1.2), June 1990.
17. ISO/IEC. *Information Technology-Security Tecniques-Entity Authentication Mechanisms, Part 1:General Model*. 1991.
18. G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.
19. G. Lowe and B. Roscoe. "Using CSP to detect errors in the TMN protocol". *IEEE Transaction on Software Engineering*, 23(10):659–669, October 1997.
20. 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.
21. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
22. J. C. Mitchell, M. Mitchell, and U. Stern. "Automated Analysis of Cryptographic Protocols Using Murϕ". In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.
23. John Mitchell Nancy Durgin and Dusko Pavlovic. A compositional logic for proving security properties of protocols, 2001. 14-th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, June 11-13.
24. R M Needham and M D Schroeder. Authentication revisited. *ACM SIGOPS Operating Systems Review*, 21(1):7–7, 1987.
25. Lawrence C. Paulson. "The Inductive Approach to Verifying Cryptographic Protocols". *Journal of Computer Security*, 6:85–128, 1998.
26. Lawrence C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.
27. M.D. Schroeder R.M. Needham. Using encryption for authentication in large networks of computers. *ACM Communication*, 21(12):993–999, 1978.
28. Paul Syverson and Iliano Cervesato. The logic of authentication protocols. In *Foundations of Security Analysis and Design*, volume 2171, pages 63–136. Lecture Notes in Computer Science, 2001.
29. T.Y.C. Woo and S.S. Lam. "Authentication for distributed systems". *IEEE Computer*, 25(3):39–51, 1992.
30. T.Y.C. Woo and S.S. Lam. "A Semantic Model for Authentication Protocols". In *Proceedings of 1993 IEEE Symposium on Security and Privacy*, pages 178–194, 1993.

A Trace semantics of ρ-spi calculus

Table 7 Transition System for ρ-spi

Transition rules: We omit the symmetric version of PAR. On Rule ENCRYPTION and DECRYPTION: $E \in \mathcal{K}_{\mathcal{L}}$ implies key$(E, I_1, I_2) \in s$ and $I_1 = I \vee I_2 = I$. The function $\text{new}_{\mathcal{LN}}(\langle s, P\rangle)$ yields a local name $a \notin fn(P) \cup fn(s)$.

OUTPUT

$$\langle s, I : \text{out}(M_1, \ldots, M_m).S\rangle \longrightarrow \langle s :: \text{out}(M_1, \ldots, M_m), I : S\rangle$$

INPUT

$$\frac{\forall i = 1, \ldots, m \qquad s \vdash M_i \qquad M_i \quad closed}{\langle s, I : \text{in}(M_1, \ldots, M_j, x_{j+1}, \ldots, x_m).S\rangle \longrightarrow \langle s :: \text{in}(M_1, \ldots, M_m), I : S[M_l/x_l \quad l = j+1, \ldots, m]\rangle}$$

RES

$$\frac{a = \text{new}_{\mathcal{LN}}(\langle s, I : \text{new}(a)S\rangle)}{\langle s, I : \text{new}(a)S\rangle \longrightarrow \langle s :: \text{new}(a), I : S\rangle}$$

PAR

$$\frac{\langle s, P\rangle \longrightarrow \langle s :: t, P'\rangle \qquad a = \text{new}_{\mathcal{LN}}(\langle s, P\rangle) \qquad b = \text{new}_{\mathcal{LN}}(\langle s, P|Q\rangle)}{\langle s, P|Q\rangle \longrightarrow \langle s :: t[b/a], P'[b/a]|Q\rangle}$$

ENCRYPTION

$$\langle s, I : \text{encrypt}\{M_1, \ldots, M_m\}_E \text{ as } x.S\rangle \longrightarrow \langle s :: I \text{ encrypts } \{M_1, \ldots, M_m\}_E, I : S[\{M_1, \ldots, M_m\}_E/x]\rangle$$

DECRYPTION

$$\langle s, I : \text{decrypt}\{M_1, \ldots, M_m\}_E \text{ as } \{M_1, \ldots, M_j; x_{j+1}, \ldots, x_m\}_E.S\rangle \longrightarrow$$
$$\langle s :: I \text{ decrypts } \{M_1, \ldots, M_m\}_E, I : S[M_l/x_l \quad l = j+1, \ldots, m]\rangle$$

RUN

$$\langle s, I_1 : \text{run}(I_1, I_2).S\rangle \longrightarrow \langle s :: \text{run}(I_1, I_2), I_1 : S\rangle$$

COMMIT

$$\langle s, I_1 : \text{commit}(I_1, I_2)\rangle \longrightarrow \langle s :: \text{commit}(I_1, I_2), I_1 : \mathbf{0}\rangle$$

KEY

$$\frac{\text{key}(k_l, I_1', I_2') \notin s}{\langle s, \text{key}(k_l, I_1, I_2).P\rangle \longrightarrow \langle s :: \text{key}(k_l, I_1, I_2), P\rangle}$$

Message manipulation rules: required by rule INPUT.

AX
$$\frac{M \in s}{s \vdash M}$$

ENV
$$\frac{\underline{a} \in \mathcal{EN}}{s \vdash \underline{a}}$$

DEC
$$\frac{s \vdash \{M_1, \ldots, M_m\}_E \qquad s \vdash E}{s \vdash M_l \quad l = 1, \ldots, m}$$

ENC
$$\frac{s \vdash M_l \quad l = 1, \ldots, m \qquad s \vdash E}{s \vdash \{M_1, \ldots, M_m\}_E}$$

The trace semantics is given in terms of a transition relation $\longrightarrow$ between *configurations*, i.e., between pairs $\langle s, P\rangle$, where $s \in Act^*$ is a trace, P is a (closed) process. The transition relation on configurations simulates the execution of a command of P and records the corresponding action t by adding it to s. The effect is to reach a new configuration $\langle s :: t, P'\rangle$, where P' is P after executing t. Notice that rules managing sequential processes preserve the identity of the sequential proecess in the initial configuration, that is such rules have the form $\langle s, I : c.S\rangle \longrightarrow \langle s', I : S\rangle$. By this technique, it is possible to keep during the reduction of a sequential process the information about its identity.

Rule OUTPUT performs an output of m messages. The configuration $\langle s, I : \text{out}(M_1, \ldots, M_m).S\rangle$ moves into $\langle s :: \text{out}(M_1, \ldots, M_m), I : S\rangle$, i.e., the action $out(M_1, \ldots, M_m)$ is added to s in the final configuration. Then, the process behaves as S. Rule INPUT describes the input of m messages. The process behaves as $P[M_l/x_l$, for $l = j+1, \ldots, m]$ if the match of the messages before the semicolon succeeds. An input can read all the messages produced by the environment through the deductive system for message manipulation, illustrated below. In Rule RES, $\langle s, \text{new}(a) I : S\rangle$ moves into $\langle s :: \text{new}(a), I : S\rangle$. $\text{new}(a)$ is a binder and $a = \text{new}_{\mathcal{LN}}(\langle s, I : new(a)S\rangle)$ is a local name not occurring free in the initial configuration. Notice that this condition is always met modulo α-renaming. Since an authentication protocol is usually encoded as a parallel composition of processes that are always associated to a given identity (for instance every initiator, server or responder has its own identity), rule INIT adds to the process in the reached configuration the tag I. Such a tag records the identity of the process. In Rule PAR, if $\langle s, P\rangle \longrightarrow \langle s :: t, P'\rangle$, then $\langle s, P|Q\rangle$ moves into $\langle s :: t[b/a], P'[b/a]|Q\rangle$. Substitution $[b/a]$ in P' and t avoids conflicts of new names possibly generated by NEW in the transition performed by P. Rule ENCRYPTION describes the encryption process. $\langle s, I : \text{encrypt } \{M_1, \ldots, M_m\}_E \text{ as } x.S\rangle$ moves into $\langle s :: I \text{ encrypts } \{M_1, \ldots, M_m\}_E, I : S[\{M_1, \ldots, M_m\}_E/x]\rangle$. Rule DECRYPTION describes the decryption process. Matching of names before the semicolon is similar to the one in INPUT. Both ENCRYPTION and DECRYPTION rules allow a process to use a long-term key k_l only if the process identity I has been associated to such a key through either key(k_l, I, I') or key(k_l, I', I). RUN, COMMIT and KEY, add to the trace of a process the information needed for authentication analysis. Notice that run(I_1, I_2) and commit(I_1, I_2) are executed only if the process identity is I_1. Moreover, rule KEY forbids multiple key assignments of the same long-term key.

The second part of Table 7 presents the proof system for message manipulation used in INPUT rule. The system formalizes the capabilities of the Dolev-Yao intruder, which we represent implicitly via the environment, and its capabilities to compute a message from its knowledge and from the trace s executed so far. Rule AX says that if a message M belongs to the trace s, then M can be produced by the environment. $M \in s$ if s contains the action out($\ldots, M, \ldots$).Rule ENV says that an environment name $\underline{a}$ can be always produced by the environment. Rule DEC says that if the environment knows a ciphertext and the corresponding decryption key, then it can produce the content of the ciphertext. Rule ENC says that if the environment knows m messages M_l and a key E, then it can produce $\{M_1, \ldots, M_m\}_E$.