Compositional Analysis of 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

Abstract.

We propose a new method for the static analysis of entity authentication 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) 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.

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., [10, 13, 20, 25, 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 have lead to the discovery of several attacks (see, e.g., [24, 26]). On the other hand, static techniques, based on type systems and control-flow analyses have proved effective in providing static guarantees of correctness [1, 2, 5, 16, 17].

Overview and main results. 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 [14, 22]. Our approach is based on a new tagging mechanism for messages that we introduce to support an analysis of the role that certain message components play in the authentication protocol. Based on that, and drawing on our earlier work in [8], we formulate

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

ESOP 2004, LNCS 2986, pages 140–154, 2004.

a set of decidable conditions on the protocols' executions that imply formal guarantees of entity authentication for a large class of protocols. Interestingly, such conditions may be checked by means of a static, and fully compositional, analysis that inspects the protocol participants and the structure of the tagged messages they exchange. The analysis is carried out in isolation on each participant: each validated principal is decreed *locally correct*. Our main result, then, is a *safety theorem* stating that protocols composed of *locally correct* principals are safe, i.e., immune to attacks mounted by any protocol intruder[1]. The safety theorem relies critically on the assumption that the messages exchanged in the protocol are tagged: hence our tags play a static as well a dynamic role, and the safety theorem assumes that the semantics of protocols is itself tagged. While this may be seen as a limitation, our tagging mechanism turns out to be less demanding than those employed to resolve message ambiguities in many existing protocol implementations (cf. §6).

We formalize our technique in a new process calculus, named ρ-spi, that we use to specify the authentication protocols of interest: ρ-spi is a dialect of the spi-calculus [2] and includes a set of authentication-specific constructs inspired by the process calculus *Lysa* [5]. Our approach appears interesting in several respects:

(*i*) The compositional nature of the analysis makes it applicable for validating unboundedly many protocol sessions: a safety proof for one protocol session is also a safety proof for an unbounded number of sessions;
(*ii*) The only human effort needed for the analysis is the encoding of the protocol narration in ρ-spi, which in turn requires one to identify the correct tags for the ciphertext components. On the one hand, this is a relatively straightforward task; on the other hand, we argue that stating the purpose of message components explicitly in the tags represents a good practice for the design of robust protocols.
(*iii*) Even though we focus on shared-key authentication protocols, our approach is fairly general and can easily be extended to deal with a wider class of authentication protocols (e.g. based on public-key cryptography).

Structure of the paper. §2 motivates and illustrates our mechanism for the tagging of messages. §3 introduces ρ-spi and its operational semantics. §4 gives a formal description of our static analysis, and its main properties. §5 illustrates the analysis with a simple example. §6 concludes the presentation with final remarks.

2 Tagged Messages

A typical source of flaws in security protocols, and specifically in authentication protocols, is a poor interpretation of messages, by which certain messages are believed to convey more guarantees than they actually do. Prudent protocol engineering principles [3] suggest instead that "every message should say what it means, i.e., its interpretation should depend only on its content". A simple instance of a message exchange that fails to comply with this principle is the one-step protocol $A \rightarrow B : \{M\}_{K_{AB}}$. Here *Alice* is

[1] We implicitly appeal to a Dolev-Yao intruder model [11]: intruders may intercept, reply and forge new messages, but never decrypt messages without knowing the corresponding keys.

sending to *Bob* the message M 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:

$$a)\ B \rightarrow E(A)\ :\ \{M\}_{K_{AB}} \qquad\qquad b)\ E(A) \rightarrow B : \{M\}_{K_{AB}}$$

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 $A \rightarrow B : \{A, M\}_{K_{AB}}$, where the first component now signals that A is the 'claimant' of the protocol.

Our approach is best understood as a generalization of this idea: we envision a tagging mechanism for messages that makes the interpretation of certain, critical message components unambiguous. The untagged part of a message forms the message's payload, while the tagged components include entity identifiers, tagged with Id, session keys, tagged with Key, and nonces (quantities generated for the purpose of being recent, or *fresh*). Nonce tags are more elaborate, as they convey information on the role that nonces play in the authentication protocol: Claim, in messages authenticating a claimant, Verif, in messages specifying an intended verifier, or Owner in messages authenticating a session key.

To motivate, consider a message $\{A, N, M\}_{K_{BS}}$ encrypted under a long-term key K_{BS} shared by a principal B and a server S. A naive tagging of this message, such as one signaling that A is an identifier, N a nonce, and M the payload would still leave several, reasonable, interpretations of the function of this message: (i) B is asking S to authenticate with A; (ii) S is telling B that A wants to authenticate with him; and (iii) S is telling B that M is or contains a fresh session key shared with A. Our nonce tags eliminate such ambiguity: the tagging $\{A : \mathsf{Id}, N : \mathsf{Claim}, M\}_{K_{BS}}$ enforces interpretation (i), while the taggings $\{A : \mathsf{Id}, N : \mathsf{Verif}, M\}_{K_{BS}}$ and $\{A : \mathsf{Id}, N : \mathsf{Owner}, M : \mathsf{Key}\}_{K_{BS}}$ enforce interpretations (ii) and (iii), respectively.

3 The ρ-spi calculus

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 [5] for the analysis of authentication protocols. ρ-spi differs from both calculi in several respects: it incorporates the notion of tagged message exchange from [8], it provides new authentication-specific constructs, and primitives for declaring process identities and (shared) long-term keys.

Syntax. The syntax of the calculus is presented in Table 1. We presuppose two countable sets: $\mathcal{N}$, of names and $\mathcal{V}$ of variables. We reserve a, b, k, m, n for names and x, y, z for variables. Both names and variables can be tagged, noted $n : C$ and $x : C$, respectively.

Table 1 The syntax of ρ-spi calculus.

Notation: d ranges over untagged names and variables, I over identity labels, D over, possibly tagged, names and variables

P,Q ::= *Processes*		S ::= *Sequential processes*	
$I \triangleright S$	(principal)	$\mathbf{0}$	(nil)
$I \triangleright !S$	(replication)	$\mathsf{in}(D_1,\ldots,D_m).S$	(input)
$P \mid Q$	(composition)	$\mathsf{out}(D_1,\ldots,D_m).S$	(output)
$\mathsf{let}\ k = \mathsf{key}(I_1,I_2).P$	(key assignment)	$\mathsf{new}(n).S$	(restriction)
		$\mathsf{decrypt}\ x\ \mathsf{as}\ \{D_1,\ldots,D_m\}_d.S$	(decryption)
		$\mathsf{encrypt}\ \{D_1,\ldots,D_m\}_d\ \mathsf{as}\ x.S$	(encryption)
		$\mathsf{run}(I_1,I_2).S$	(run)
		$\mathsf{commit}(I_1,I_2).S$	(commit)

Tags, denoted by C, are a special category of names and include roles (the three special names $\mathsf{Claim}, \mathsf{Owner}, \mathsf{Verif}$), the identity tag Id and the session key tag Key. *Identities* $I\mathcal{D}$ are a subset of names, further partitioned into *principals* $I_{\mathcal{P}}$, ranged over A and B, and *trusted third parties* (TTPs) $I_{\mathcal{T}}$, ranged over by T.

Processes (or *protocols*), ranged over by P,Q are formed as parallel composition of principals. To allow the sharing of long-term keys among principals, we provide ρ-spi with let-bindings as in $\mathsf{let}\ k = \mathsf{key}(I_1,I_2).P$ to declare (and bind) the long-term key k shared between I_1 and I_2 in the scope P. Each principal is a sequential process with an associated identity, noted $I \triangleright S$. The replicated form $I \triangleright !S$ indicates an arbitrary number of copies of S associated with identity I.

Sequential processes may never fork into parallel components: this assumption helps assign unique identities to (sequential) processes, and involves no significant loss of expressive power as protocol principals are typically specified as sequential processes, possibly sharing some long-term keys. The sequential process $\mathbf{0}$ is the null process that does nothing, as usual. Process $\mathsf{new}(n).S$ generates a fresh name n local to S. The constructs for input, output and decryption are essentially the same as in the calculus *Lysa*. In particular, we presuppose a unique (anonymous) public channel, the network, from/to which all principals, including intruders, read/send messages. Similarly to *Lysa*, our input and decryption constructs may test part of the message read (decrypted), by pattern-matching. The pattern matching mechanism is so defined as to ensure that untagged patterns only match untagged messages, while tagged patterns only match tagged messages (provided that the tags also match). Accordingly, the pattern x matches the message n (and binds x to n) but it does not match $n : \mathsf{Claim}$ (as x is untagged, while $n : \mathsf{Claim}$ is tagged). Similarly, $x : \mathsf{Claim}$ matches $n : \mathsf{Claim}$ but does not match $n : \mathsf{Verif}$.

Distinctive of ρ-spi is the presence of an explicit construct for encryption: process $\mathsf{encrypt}\ \{D_1,\ldots,D_m\}_d\ \mathsf{as}\ x.S$ binds variable x to the encrypted message $\{D_1,\ldots,D_m\}_d$ in the continuation S. The syntactic structure of ρ-spi requires all the encrypted messages to be explicitly formed in an encryption prefix, and forbids ciphertexts to be sent directly on the output channel. These restrictions are useful for our analysis, as they

Table 2 A sample protocol narration ρ-spi

Protocol	$\triangleq$ let $k_{AB} = \text{key}(A,B)(A \triangleright \mathit{Initiator}(A,B) \mid B \triangleright \mathit{Responder}(B,A))$
Initiator(A,B)	$\triangleq$ $\text{new}(m).\text{in}(x).\text{run}(A,B).\text{encrypt}\{x,m\}_{k_{AB}}$ as $y.\text{out}(y)$
Responder(B,A)	$\triangleq$ $\text{new}(n_B).\text{out}(n_B).\text{in}(y).$decrypt y as $\{n_B,z\}_{k_{AB}}.\text{commit}(B,A)$

ease the reasoning based on the structural inspection of the encrypted messages of a protocol. Finally, the process forms $\mathsf{run}(I_1,I_2).S$ and $\mathsf{commit}(I_1,I_2).S$ declare that the sequential process I_1 is starting, respectively committing, a protocol session with I_2. These constructs are used to check the *correspondence assertions* as done in [16].

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 in components D_i; in all cases the scope of the binders is the continuation process. The notions of free/bound names and variables arise as expected.

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

$$1)\ B \to A : n_B \qquad 2)\ A \to B : \{n_B, m\}_{k_{AB}}$$

We assume k_{AB} to be known only by A and B and n_B 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_B. This protocol can be formalized in our calculus as shown in Table 2. Process *Initiator*, generates a fresh message $m \in \mathcal{N}$. 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* generates a fresh nonce n_B 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 mechanism of decryption). If the match is successful, the variable z gets bound to a message (from A) and the principal commits through $\text{commit}(B,A)$. 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 [6], and formalize it by means of a transition relation between *configurations*, i.e., pairs $\langle s, P\rangle$, where $s \in \mathit{Act}^*$ is a trace, P is a (closed) process. Each transition $\langle s,P\rangle \longrightarrow \langle s :: \alpha, P'\rangle$ simulates one computation step in P and records the corresponding action in the trace. The transitions involving a sequential process preserve the identity identifiers associated with the process, as in $\langle s, I \triangleright \pi.S\rangle \longrightarrow \langle s :: \alpha, I \triangleright S\rangle$, where α is the action corresponding to the prefix π. The transitions, in Table 3, are mostly standard. As in companion transition systems, see, e.g. [7], we identify processes up to renaming of bound variables and names, i.e., up to α-equivalence. Moreover we assume two infinite sets of bound names and free names so that bound names are distinct from free names and not touched by substitutions. 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. Rule

Table 3 Transition System for ρ-spi

Transition rules: $P[M/x]$ is the result of substituting M for all the free occurrences of x in P, and define $P[M:C/x:C] \triangleq P[M/x]$; the category M of *Messages* is defined by the following productions: $M ::= x \mid n \mid \{M_1,\ldots,M_m\}_M \mid M:C$; $\mathrm{bn}(s)$ is the set of bound names in s; the symmetric rule of PAR is omitted.

REPLICATION
$\langle s, I \triangleright !S\rangle \;\rightarrow\; \langle s, I \triangleright S | I \triangleright !S\rangle$

PAR
$$\frac{\langle s,P\rangle \;\rightarrow\; \langle s',P'\rangle}{\langle s,P|Q\rangle \;\rightarrow\; \langle s',P'|Q\rangle}$$

RES
$$\frac{n \notin \mathrm{bn}(s)}{\langle s, I \triangleright \mathsf{new}(n).S\rangle \rightarrow \langle s :: I \triangleright new(n), I \triangleright S\rangle}$$

RUN
$\langle s, A \triangleright \mathsf{run}(A,B).S\rangle \rightarrow \langle s :: run(A,B), A \triangleright S\rangle$

COMMIT
$\langle s, A \triangleright \mathsf{commit}(A,B).S\rangle \rightarrow \langle s :: commit(A,B), A \triangleright S\rangle$

INPUT
$$\frac{\forall i = 1,\ldots,m \qquad s \vdash M_i \qquad D_i \text{ matches } M_i}{\langle s, I \triangleright \mathsf{in}(D_1,\ldots,D_m).S\rangle \;\rightarrow\; \langle s :: I \triangleright in(M_1,\ldots,M_m), I \triangleright S[M_1/D_1,\ldots M_m/D_m]\rangle}$$

OUTPUT
$\langle s, I \triangleright \mathsf{out}(M_1,\ldots,M_m).S\rangle \;\rightarrow\; \langle s :: I \triangleright out(M_1,\ldots,M_m), I \triangleright S\rangle$

KEY
$$\frac{k \notin \mathrm{bn}(s)}{\langle s, \mathsf{let}\ k = \mathsf{key}(I_1,I_2).P\rangle \;\rightarrow\; \langle s :: key(k,I_1,I_2), P\rangle}$$

DECRYPTION
$$\frac{\forall i = 1,\ldots,m \qquad D_i \text{ matches } M_i \qquad \mathrm{key}(M,I_1,I_2) \in s \Rightarrow I \in \{I_1,I_2\}}{\langle s, I \triangleright \mathsf{decrypt}\{M_1,\ldots,M_m\}_M \,\mathsf{as}\, \{D_1,\ldots,D_m\}_M.S\rangle \rightarrow \langle s :: I \triangleright dec\{M_1,\ldots,M_m\}_M, I \triangleright S[M_i/D_i]\rangle}$$

ENCRYPTION
$$\frac{\mathrm{key}(M,I_1,I_2) \in s \Rightarrow I \in \{I_1,I_2\}}{\langle s, I \triangleright \mathsf{encrypt}\{M_1,\ldots,M_m\}_M \,\mathsf{as}\, x.S\rangle \rightarrow \langle s :: I \triangleright enc\{M_1,\ldots,M_m\}_M, I \triangleright S[\{M_1,\ldots,M_m\}_M/x]\rangle}$$

Message manipulation rules: required by rule INPUT.

AX
$$\frac{I \triangleright out(M_1,\ldots,M_m) \in s}{s \vdash M_l \quad l = 1\ldots m}$$

TAG
$$\frac{s \vdash M}{s \vdash M : C}$$

UNTAG
$$\frac{s \vdash M : C}{s \vdash M}$$

ENV
$$\frac{n \notin \mathrm{bn}(s)}{s \vdash n}$$

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

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

ENV provides the environment with the power of generating a new bound name not occurring in the trace.

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, $T(P) = \{s \mid \exists P' \text{ s.t. } \langle \varepsilon, P \rangle \longrightarrow^* \langle s, P' \rangle\}$*

The notion of safety is standard (cf. [16]) and based on correspondence assertions. We say that a trace is *safe* if every $commit(B,A)$ is preceded by a distinct $run(A,B)$.

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.

Example 2. We look again at the protocol of Example 1, and now consider a multiple-session version of the protocol, with A and B running both as Initiator and as Responder: the new version of the protocol is subject to the following, well-known, *reflection* attack.

$$\begin{array}{ll} 1.a)\ B \rightarrow E(A) : n_B & 2.a)\ B \rightarrow E(A) : \{m, n_B\}_{k_{AB}} \\ 1.b)\ E(A) \rightarrow B : n_B & 2.b)\ E(A) \rightarrow B : \{m, n_B\}_{k_{AB}} \end{array}$$

We show that the attack is captured by our notion of safety. As we mentioned earlier, we can model multiple sessions, directly by replication. For the protocol in question, this amounts to analyzing the following process:

$$\begin{array}{rl} Protocol_2 \triangleq & \mathsf{let}\ k_{AB} = \mathsf{key}(A,B).\ (A \triangleright !Initiator(A,B) \mid A \triangleright !Responder(A,B)) \mid \\ & \qquad B \triangleright !Responder(B,A) \mid B \triangleright !Initiator(B,A)) \end{array}$$

where A and B run both as Initiator and as Responder (the two processes $Initiator(B,A)$ and $Responder(A,B)$ model B running as Initiator and A running as Responder, respectively). Consider now the following trace for $Protocol_2$:

$$\begin{array}{c} \mathrm{key}(k_{AB},A,B) :: \mathbf{B} \triangleright \mathrm{new}(m) :: B \triangleright \mathrm{new}(n_B) :: B \triangleright \mathrm{out}(n_B) :: \mathbf{B} \triangleright \mathrm{in}(n_B) :: \mathrm{run}(\mathbf{B},\mathbf{A}) :: \\ \mathbf{B} \triangleright \mathrm{enc}\{n_B,m\}_{k_{AB}} :: \mathbf{B} \triangleright \mathrm{out}(\{n_B,m\}_{k_{AB}}) :: B \triangleright \mathrm{in}(\{n_B,m\}_{k_{AB}}) :: B\ \mathrm{dec}\ \{n_B,m\}_{k_{AB}} :: \mathrm{commit}(B,A) \end{array}$$

where B is running as initiator instead of A (as pointed out in bold font). The trace is easily seen to be unsafe, as commit(B,A) is not matched by any run(A,B). Indeed, B is running the protocol with himself while A is doing nothing. Interestingly, the unsafe trace corresponds to the attack displayed above.

4 A Compositional Proof Technique

The proof technique we propose applies to protocol narrations that may be coded as ρ-spi terms of the form $\mathbf{keys}(k_1,\ldots,k_n).(I_1 \triangleright !S_1 \mid \ldots \mid I_m \triangleright !S_m)$, where $\mathbf{keys}(k_1,\ldots,k_n)$ represents a sequence of let binding for the long-term keys $k_1,\ldots,k_n$. As we noted earlier, the protocols of interest may directly be represented as processes in the above form: hence there is no significant loss of expressive power in our choice of a specific process format. The analysis proceeds by examining each process $\mathbf{keys}(k_1,\ldots,k_n).I_i \triangleright S_i$, and attempts to validate S_i under the key assignment determined by $\mathbf{keys}(k_1,\ldots,k_n)$.

Table 4 Local correctness: Principal and TTP rules

$\Pi(x) = enc\{\ldots\}_d$ indicates that $x \mapsto enc\{\ldots\}_d \in \Pi$, and similarly for the other entries. We write $\Pi(\bullet) = enc\{\ldots\}_d$ to mean that there exists x such that $\Pi(x) = enc\{\ldots\}_d$.

Claimant and Verifier Rules

AUTHENTICATE CLAIM

$$\frac{A;\Gamma,n:\mathsf{checked};\Pi \vdash S \qquad \Pi(\bullet) = dec\{B:\mathsf{Id},n:\mathsf{Claim},\ldots\}_k \qquad \Pi(k) \in \{key(A,T),key(A,B)\}}{A;\Gamma,n:\mathsf{unchecked};\Pi \vdash \mathsf{commit}(A,B).S}$$

AUTHENTICATE OWNER

$$\frac{\begin{array}{c}A;\Gamma,n:\mathsf{checked};\Pi \vdash S \qquad \Pi(\bullet) = dec\{B:\mathsf{Id},n:\mathsf{Owner},y:\mathsf{Key},\ldots\}_k \\ \Pi(k) = key(A,T) \qquad \Pi(\bullet) = dec\{D_1,\ldots,D_m\}_y \\ (\Pi(\bullet) = enc\{D'_1,\ldots,D'_m\}_{y'} \quad \text{implies } \exists i \text{ s.t. } D'_i \text{ does not match } D_i)\end{array}}{A;\Gamma,n:\mathsf{unchecked};\Pi \vdash \mathsf{commit}(A,B).S}$$

CLAIMANT

$$\frac{A;\Gamma;\Pi,B \mapsto run, y \mapsto enc\{A:\mathsf{Id},x:\mathsf{Claim}\}_k \vdash S \qquad \Pi(k) = key(A,B)}{A;\Gamma;\Pi,B \mapsto run \vdash \mathsf{encrypt}\{A:\mathsf{Id},x:\mathsf{Claim},\ldots\}_k \text{ as } y.S}$$

VERIFIER

$$\frac{A;\Gamma;\Pi,B \mapsto run, y \mapsto enc\{B:\mathsf{Id},x:\mathsf{Verif}\}_k \vdash S \qquad \Pi(k) = \mathsf{key}(A,T)}{A;\Gamma;\Pi,B \mapsto run \vdash \mathsf{encrypt}\{B:\mathsf{Id},x:\mathsf{Verif},\ldots\}_k \text{ as } y.S}$$

OWNER

$$\frac{A;\Gamma;\Pi,B \mapsto run, y \mapsto enc\{D_1,\ldots,D_m\}_x \vdash S \qquad \Pi(\bullet) = dec\{B:\mathsf{Id},n:\mathsf{Owner},x:\mathsf{Key},\ldots\}_k \qquad \Pi(k) = key(A,T)}{A;\Gamma;\Pi,B \mapsto run \vdash \mathsf{encrypt}\{D_1,\ldots,D_m\}_x \text{ as } y.S}$$

RUN

$$\frac{A;\Gamma;\Pi,B \mapsto run \vdash S}{A;\Gamma;\Pi \vdash \mathsf{run}(A,B).S}$$

TTP Rules

TTP FORWARD & CHECK

$$\frac{T;\Gamma,n:\mathsf{checked};\Pi,y \mapsto enc\{A:\mathsf{Id},x:\mathsf{Claim}\}_{k_{BT}} \vdash S \qquad \Pi(\bullet) = dec\{B:\mathsf{Id},n:\mathsf{Verif},\ldots\}_{k_{AT}} \qquad \Pi(k_{BT}) = key(B,T) \qquad \Pi(k_{AT}) = key(A,T)}{T;\Gamma,n:\mathsf{unchecked};\Pi \vdash \mathsf{encrypt}\ \{A:\mathsf{Id},x:\mathsf{Claim},\ldots\}_{k_{BT}} \text{ as } y.S}$$

TTP FORWARD

$$\frac{T;\Gamma;\Pi,y \mapsto enc\{A:\mathsf{Id},x:\mathsf{Claim}\}_{k_{BT}} \vdash S \qquad \Pi(\bullet) = dec\{B:\mathsf{Id},x:\mathsf{Verif},\ldots\}_{k_{AT}} \qquad \Pi(k_{BT}) = key(B,T) \qquad \Pi(k_{AT}) = key(A,T)}{T;\Gamma;\Pi \vdash \mathsf{encrypt}\ \{A:\mathsf{Id},x:\mathsf{Claim},\ldots\}_{k_{BT}} \text{ as } y.S}$$

TTP DISTRIBUTE

$$\frac{T;\Gamma;\Pi,y \mapsto enc\{A:\mathsf{Id},x:\mathsf{Owner},k_s:\mathsf{Key}\}_k \vdash S}{T;\Gamma;\Pi \vdash \mathsf{encrypt}\ \{A:\mathsf{Id},x:\mathsf{Owner},k_s:\mathsf{Key},\ldots\}_k \text{ as } y.S}$$

Table 5 Local Correctness: Generic Rules

NIL
$$\frac{}{I;\Gamma;\Pi \vdash \mathbf{0}}$$

NEW
$$\frac{I;\Gamma,n : \text{unchecked};\Pi \vdash S \qquad n \text{ fresh in } \Gamma}{I;\Gamma;\Pi \vdash \mathsf{new}(n).S}$$

INPUT
$$\frac{I;\Gamma;\Pi \vdash S}{I;\Gamma;\Pi \vdash \mathsf{in}(\ldots).S}$$

OUTPUT
$$\frac{I;\Gamma;\Pi \vdash S}{I;\Gamma;\Pi \vdash \mathsf{out}(\ldots).S}$$

ENCRYPTION
$$\frac{I;\Gamma;\Pi.y \mapsto enc\{d_1,\ldots,d_m\}_d \vdash S \qquad \Pi(\bullet) = dec\{\ldots,d : \mathsf{Key},\ldots\}_k \text{ implies } \Pi = \Pi'.B \mapsto run.x \mapsto enc\{\ldots\}_d,\Pi''}{I;\Gamma;\Pi \vdash \mathsf{encrypt}\{d_1,\ldots,d_m\}_d \text{ as } y.S}$$

DECRYPTION
$$\frac{I;\Gamma;\Pi,y \mapsto dec\{D_1,\ldots,D_m\}_d \vdash S}{I;\Gamma;\Pi \vdash \mathsf{decrypt}\ y \text{ as } \{D_1,\ldots,D_m\}_d.S}$$

4.1 Proof Rules

The proof rules (tables 4 and 5) derive judgments of the form $I;\Gamma;\Pi \vdash S$ validating the sequential process S relative to the identity I and the two environments Γ and Π. The history environment Π traces the ordered list of encryptions, decryptions, run and key assignment events performed by the party in question. The nonce environment Γ holds the nonces that the party generates. A nonce is first included in Γ as *unchecked*, when it is introduced by a new prefix. Subsequently, the nonce may be *checked* by pattern matching, and marked as such: the proof rules are so defined as to ensure that each nonce may be checked at most once, as desired. We make a few, important, assumptions which we leave implicit in the rules to avoid cluttering the notation. The environments Γ and Π may contain at most one binding for each name (key, nonce) or variable in their domain: this may require alpha renaming on the processes being analyzed. In addition, all the output or encrypted tuples as well as all the tuples of input or decryption patterns occurring in the rules are assumed to contain at most one component with tag Id, at most one component with tag $R \in \{\mathsf{Claim},\mathsf{Verif},\mathsf{Owner}\}$ and at most one component with tag Key. The relative order among the encrypted elements is immaterial.

The first two rules in Tables 4 formalize the two possible ways for a principal A to authenticate another principal B. Specifically, rule (AUTHENTICATE CLAIM) states that A may legally commit (hence authenticate) B only if A has previously generated a nonce n and decrypted a message $\{B : \mathsf{Id}, n : \mathsf{Claim}\ldots\}_k$ (with the same nonce) with k shared either with T or with B. Rule (AUTHENTICATE OWNER), in turn, states that A may commit B if she has decrypted (i) a message $\{B : \mathsf{Id}, n : \mathsf{Owner}, y : \mathsf{Key},\ldots\}_k$, encrypted by a TTP and including a fresh session key y owned by B and nonce n that A previously generated; and (ii) at least one message $\{D_1,\ldots,D_m\}_y$ that she did not generate. The first decryption guarantees that the session key y is fresh and shared by A and B; then, with the second decryption she is guaranteed that the message comes

from B. Since the encryption key is fresh, so must be the message, and A may safely authenticate B.

The authentication guarantees we just discussed require a very careful use of the role tags attached to the encrypted message components. In particular, the message $\{B : \mathsf{Id}, n : \mathsf{Claim}, \ldots\}_k$ should be generated only if B is indeed willing to authenticate with A; similarly, $\{B : \mathsf{Id}, n : \mathsf{Owner}, k_s : \mathsf{Key}, \ldots\}_k$ should only be generated if k_s is a fresh session key shared between A and B. This intended use of role tags is enforced by the remaining rules in Table 4, which we comment below.

Rules (CLAIMANT), (VERIFIER) and (OWNER) formalize the ways in which A may declare her willingness to authenticate with B. In particular, in rule (CLAIMANT), A may request B to authenticate herself as claimant by sending a message $\{A : \mathsf{Id}, x : \mathsf{Claim}, \ldots\}_k$ directly to B using a long-term key k she shares with B. In an ideal protocol run, x is intended to be a nonce sent as a challenge by B, that B will check with (AUTHENTICATE CLAIM). On the other hand, rule (CLAIMANT) in itself does not impose any condition on x as it is B, rather than A, which is in charge of checking the nonce for the purpose of authentication.

In rule (VERIFIER), A may request a TTP T to authenticate herself with B as verifier, by sending a message $\{B : \mathsf{Id}, x : \mathsf{Verif}, \ldots\}_k$ to T using a long-term key k shared with T. The role Verif informs T about A's intention of authenticating with someone else (B), while x is again meant to represent the challenge, originated by T.

In rule (OWNER), A may send a message $\{D_1, \ldots, D_m\}_y$ to confirm to have received the fresh session key y, provided that (i) she has previously decrypted a message $\{B : \mathsf{Id}, n : \mathsf{Owner}, y : \mathsf{Key}, \ldots\}_k$ originated by a TTP T and declaring that y is a fresh key shared with B, and (ii) she has previously performed a run with B.

The three rules all validate the start of new runs. Correspondingly, they all require A to have previously marked the start of a run by issuing run(A, B), with rule (RUN). The *run* event must be the last action performed, to guarantee that these rules are applied only once for each protocol run.

Rules (TTP FORWARD & CHECK), (TTP FORWARD) and (TTP DISTRIBUTE) govern the behavior of TTPs. In rule (TTP FORWARD & CHECK), T generates a message $\{A : \mathsf{Id}, x : \mathsf{Claim}, \ldots\}_{k_{BT}}$ if it has previously decrypted (and checked the nonce n of) a message of the form $\{B : \mathsf{Id}, n : \mathsf{Verif}, \ldots\}_{k_{AT}}$, with k_{AT} and k_{BT} shared with A and B respectively.

In rule (TTP FORWARD), T may generate a message $\{A : \mathsf{Id}, x : \mathsf{Claim}, \ldots\}_{k_{BT}}$ if it has previously decrypted (without checking the nonce) a message of the form $\{B : \mathsf{Id}, x : \mathsf{Verif}, \ldots\}_{k_{AT}}$ (again k_{AT} and k_{BT} are shared with A and B); notice that x is forwarded to B so that B can check the freshness of the first message.

Finally, in rule (TTP DISTRIBUTE), T declares new session keys through messages of the form $\{A : \mathsf{Id}, x : \mathsf{Owner}, k_s : \mathsf{Key}, \ldots\}_k$.

The rules in Table 5 complete the axiomatization. Rules (NEW), (OUTPUT), (KEY) and (INPUT) are standard. Rules (DECRYPTION) and (ENCRYPTION) handle the cases of the corresponding constructs in which none of the rules in Table 4 apply. In fact, the rule (ENCRYPTION) does overlap with (OWNER) in case of encryptions with session keys: this is safe only if the start of the authentication session has previously been asserted (with a *run* event), in which case (OWNER) must have previously been applied

for another encryption with the same session key. The safety check is implemented by requiring that the two consecutive bindings $A \mapsto \mathsf{run}(B).x \mapsto enc\{\ldots\}_d$ be part of the history environment Π.

4.2 Safety Theorem

The safety proof for a protocol requires further hypotheses on the way that long-term and session keys are circulated among principals and trusted third parties. Specifically, given a process $\mathbf{keys}(k_1,\ldots,k_n).I \triangleright S$, we make the following assumptions: (a) long-term keys are never circulated in clear or within encrypted messages; (b) only principals (not TTP's) may receive session keys, without leaking them; and (c) fresh session keys are only distributed by TTP's at most to two parties, only once. When these assumptions are satisfied, we say that $I \triangleright S$ is $\{k_1,\ldots,k_n\}$-safe.

Definition 3 (Local Correctness). Let P be the process $\mathbf{keys}(k_1,\ldots,k_n).I \triangleright S$, with k_i shared between I_i and J_i. We say that P is *locally correct* iff $I \triangleright S$ is $\{k_1,\ldots,k_n\}$-safe and the judgment $I;\varnothing;\Pi_{k_1,\ldots,k_n} \vdash S$ is derivable, for $\Pi_{k_1,\ldots,k_n} = k_1 \mapsto key(I_1,J_1),\ldots,k_n \mapsto key(I_n,J_n)$.

A protocol is correct if all of its sequential components are locally correct. Formally, the process $\mathbf{keys}(k_1,\ldots,k_n).(I_1 \triangleright !S_1 \mid \cdots \mid I_m \triangleright !S_m).$ is *correct* if for all $i \in \{1,\ldots,m\}$ the process $\mathbf{keys}(k_1,\ldots,k_n).I_i \triangleright S_i$ is locally correct. Our safety theorem states that correct processes are safe: hence, the local correctness of the protocol participants is a sufficient condition for the safety of the protocol as a whole (the proof, worked out in the full version of the paper, is omitted here for the lack of space).

Theorem 1 (Safety). *Let $P = \mathbf{keys}(k_1,\ldots,k_n).(I_1 \triangleright !S_1 \mid \cdots \mid I_m \triangleright !S_m)$. If P is correct, then it is safe.*

5 An Example

We illustrate the analysis on the ρ-spi specification of the protocol in Table 2. As discussed in Example 2, the protocol is subject to attacks when running in multiple sessions: with our analysis the flaw is unveiled by a failure to validate the responder, as we illustrate showing that the principal $\mathsf{let}\ k_{AB}{=}\mathsf{key}(k_{AB},A,B).B \triangleright Responder(B,A)$ is not locally correct. To see that, note that for $B;\Gamma;\Pi \vdash \mathsf{commit}(B,A)$ to be derivable, by rules (AUTHENTICATE CLAIM) and (AUTHENTICATE OWNER), there must exists y such that $\Pi(y) = dec\{A:\mathsf{Id}, n_b:R,\ldots\}_{k_{AB}}$, with $R \in \{\mathsf{Claim},\mathsf{Owner}\}$. On the other hand, the only decryption in S is $\mathsf{decrypt}\ y\ \mathsf{as}\ \{n_B,z\}_{k_{AB}}$, which implies $B;\Gamma;\Pi \nvdash \mathsf{commit}(B,A)$. The failure of local correctness persists when we add a role $R \in \{\mathsf{Claim},\mathsf{Owner}\}$ to the ciphertext. In fact, even in that case, the hypotheses of the two authentication rules do not hold because the ciphertext is still missing the identity label A. Interestingly, the attack presented in Section 3 exploits precisely this flaw. The reflection attack is prevented in the ISO Two-Pass Unilateral Authentication Protocol [21] which fixes the problem by including the identity label A in the ciphertext sent by A (in fact the original version from [21] includes B instead of A in the ciphertext).

Table 6 A variant of the ISO Two-steps Unilateral Authentication Protocol

$Protocol_{ISO} \triangleq \mathsf{let}\ k_{AB} = \mathsf{key}(A,B).(A \triangleright Initiator_{ISO}(k_{AB},A,B) \mid B \triangleright Responder_{ISO}(k_{AB},B,A))$

$Initiator_{ISO}(k_{AB},A,B) \triangleq$		$Responder_{ISO}(k_{AB},B,A) \triangleq$	
$\mathsf{new}(m)$.	NEW	$\mathsf{new}(n_B)$.	NEW
$\mathsf{in}(x)$.	IN	$\mathsf{out}(n_B)$.	OUT
$\mathsf{run}(A,B)$.	RUN	$\mathsf{in}(y)$.	IN
$\mathsf{enc}.\{A:\mathsf{Id},x:\mathsf{Claim},m\}_{k_{AB}}$ as y.	CLAIM	dec. y as $\{A:\mathsf{Id},n_B:\mathsf{Claim},z\}_{k_{AB}}$.	DEC
$\mathsf{out}(y)$	OUT	$\mathsf{commit}(B,A)$	AUTH CLAIM

$$1)\ B \rightarrow A : n_B \qquad 2)\ A \rightarrow B : \{A,n_B,m\}_{k_{AB}}$$

The specification of the new protocol in ρ-spi is given in Table 6. The only non-obvious aspect of the specification is in the choice of the claimant role for the label A in the encrypted message. Given this choice the protocol is easily proved correct, as we outline next. The second and fourth columns in Table 6 refer to the rules applied for verifying local correctness of the two principals. In particular, for $A;\oslash;\Pi_{k_{AB}} \vdash Initiator_{ISO}(k_{AB},A,B)$, the hypotheses of the rules (NEW), (INPUT), (RUN) and (OUTPUT) are trivially verified. The only interesting case is rule (CLAIMANT), which is applied in the proof of

$$A;\{m:\text{unchecked}\};\Pi_A \vdash \mathsf{encrypt}\{A:\mathsf{Id},x:\mathsf{Claim},m\}_{k_{AB}}\ \mathsf{as}\ y.\mathsf{out}(y)$$

Rule (CLAIMANT) requires $\Pi_A = \Pi'_A.B \mapsto run$ and $\Pi_A(k_{AB}) = key(A,B)$, which are both true. As for $Responder_{ISO}(k_{AB},B,A)$, proving $B;\Gamma_B;\Pi_B \vdash \mathsf{commit}(B,A)$ is straightforward. The hypotheses for rule (AUTHENTICATE CLAIM) hold since $\Pi_B(\bullet) = dec\{A:\mathsf{Id},n_B:\mathsf{Claim},z\}_{k_{AB}}$, $\Gamma_B(n_b) = unchecked$, and $\Pi_B(k_{AB}) = key(A,B)$. Now one derives $B;\oslash;\Pi_{k_{AB}} \vdash Responder_{ISO}\ (k_{AB},B,A)$, routinely, by rules (NEW), (OUTPUT), (INPUT) and (DECRYPTION).

Finally, both principals handle keys safely, as they use no session keys and never send long term keys on the net or encrypted. By Theorem 1, we know that every trace of $Protocol_{ISO}$ is safe, hence the protocol is immune to authentication attacks.

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

$$Protocol_{ISO} - m \triangleq \mathsf{let}^m_{i,j=1,i<j} k_{ij} = \mathsf{key}(I_i,I_j).\\ (|^m_{i,j=1,i\neq j} I_i \triangleright !Initiator_{ISO}(k_{ij},I_i,I_j) | I_i \triangleright !Responder_{ISO}(k_{ij},I_i,I_j)))$$

Here k_{ij} represents a long term key shared between entities I_i and I_j. We assume that $k_{ij} = k_{ji}$ (and correspondingly define the key assignment only for keys k_{ij} with $i < j$). The correctness proof for this version of the protocol derives directly, by compositionality, from the correctness proof we just outlined.

6 Conclusion

We have proposed a new, compositional, analysis of entity authentication protocols, based on a new tagging mechanism for encrypted messages. The impact of the tagging

Table 7 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		X
Needham Schroeder Amended Version	X	
Otway and Rees Protocol		X
Carlsen's Secret Key Initiator Protocol	X	
Wide Mouthed Frog Protocol,Nonce Based Version	X	
Woo and Lam Protocol		X
Woo and Lam Amended Version	X	

mechanism is twofold. On one hand, it is a specification tool: stating the purpose of message components explicitly leads to more robust protocol design. On the other hand, it is directly useful as an implementation technique which allows a non-ambiguous interpretation of messages, as other tagging techniques employed in protocol implementations, while at the same time supporting formal correctness proofs.

The analysis is appealing for its relative simplicity: verifying the conditions of local correctness is intuitive and easily implemented by an automatic tool. We have conducted various (hand written) tests on authentication protocols from the literature, some of which are given in Table 7: for all the correct versions of the protocols the analysis admits safety proofs, while it consistently fails to validate the flawed versions.

The authentication patterns we have investigated are certainly not exhaustive, but fairly general. Also, the framework is scalable, as new patterns may be added, as needed. Desirable extensions include new patterns needed to validate, e.g, protocols based on public-key encryption and other kinds of nonce-challenges [18, 19]. Our current work shows that the approach fits very well these extensions.

Related Work. Tagging is not a new idea and it is proposed and used for verification purposes in [3, 4, 16, 17, 20]. Typically, tagging amounts to add a different label to each encrypted protocol message, so that ciphertexts cannot be confused. Our tagging is less demanding, as we do not require that every message is unambiguously tagged since we tag only certain components. In particular, for protocols implemented with stronger tagging techniques, our tags can be safely removed without compromising the protocols' safety. Our tagging seems also well suited for multi-protocol systems. In [23], Syverson and Meadows observe that "problems can arise when a protocol is interacting with another protocol that does not use a tagging scheme, or tags data in a different way". Our tagging does not require that different protocols use different tags, since it is related to the security informations conveyed by a ciphertext. If it was uniformly applied in every protocol the safety result would scale up to the parallel composition of different authentication protocols.

In [16, 17] 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 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. Moreover the human effort required for the type definitions is relevant, while our tags range on a finite set and their definition is simple and relatively mechanical.

The recent work by Bodei *et al.* on a control-flow analysis for message authentication in *Lysa* [5] 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. Indeed, our analysis provides a strong form of entity authentication guarantee [22, 28]: 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 , like, e.g., BAN [9], GNY [15], 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 [12] by Durgin, Mitchell and Pavlovic. They propose a logic for security protocols which is designed around a process calculus derived from Strand Spaces [19, 27], 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 comparable to ours.

‘

References

1. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theor. Comput. Sci.*, 298(3):387–415, 2003.
2. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
3. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.
4. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. *Proceedings of Foundations of Software Science and Computation Structures*, pages 136–152, 2003.
5. 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)*, pages 126–140, June 2003.
6. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 01*, volume 2076, pages 667–681. LNCS 2076, Springer Verlag, 2001.

7. M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Logic in Computer Science*, pages 157–166, 1999.
8. M. Bugliesi, R. Focardi, and M. Maffei. Principles for entity authentication. In *Proceedings of 5th International Conference Perspectives of System Informatics (PSI 2003)*, volume 2890, pages 294–307, July 2003.
9. M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". *Proceedings of the Royal Society of London*, 426(1871):233–271, 1989.
10. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. http://www.cs.york.ac.uk/∼jac/papers/drareview.ps.gz, November 1997.
11. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
12. N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11, 2003.
13. 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.
14. 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.
15. L. Gong, R. Needham, and R. 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.
16. 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.
17. A. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. In *15th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 77–91. IEEE Computer Society Press, 24–26 June 2002.
18. J. Guttman. Security protocol design via authentication tests. In *15th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 92–103, Cape Breton, Canada, 24–26 June 2002. IEEE Computer Society Press.
19. Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.
20. J. Heather, G. Lowe, and S. 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.
21. ISO/IEC. "Entity Authentication Using Symmetric Techniques". Report ISO/IEC JTC1.27.02.2 (20.03.1.2), June 1990.
22. G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*, pages 31–44. IEEE press, 1997.
23. C. Meadows and P. Syverson. Formal specification and analysis of the group domain of intrepretation protocol using npatrl and the nrl protocol analyzer, 2003. to appear in Journal of Computer Security.
24. 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.
25. R M Needham and M D Schroeder. Authentication revisited. *ACM SIGOPS Operating Systems Review*, 21(1):7–7, 1987.
26. L. C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.
27. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. Journal of Computer Security, 1999. 15.
28. T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(3):39–51, 1992.