Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 99 (2004) 267–293

www.elsevier.com/locate/entcs

# The ρ-spi Calculus at Work: Authentication Case Studies

Riccardo Focardi [a,1,2] Matteo Maffei[a,1,3]

[a] *Dipartimento di Informatica*
*Università Ca'Foscari*
*Venezia, Italia*

Abstract

In [10], we introduce a process calculus for describing security protocols and we propose a static and compositional analysis of entity authentication. In this paper we apply such a technique on well-known shared key authentication protocols. The analysis helps clarifying the protocol logics, suggests simplifications and reveals some attacks. Moreover we discuss how our analysis scales up to multi-protocol systems.

*Keywords:* Security, Process Calculi, Static Analysis

## 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 (e.g., [12,15,20,22,25,27]). In most cases, such attacks dwell on flaws in the protocols' logic, rather than on breaches in the underlying cryptosystem.

[1] 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).

[2] Email: focardi@dsi.unive.it

[3] Email: maffei@dsi.unive.it

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 ( e.g., [22,23,24,26,27]). On the other hand, static techniques, based on type systems and control-flow analyses have proved effective in providing static guarantees on the protocols' correctness [1,3,2,6,16,17].

**Overview** Static analysis is also at the basis of the technique we propose in [10,9]: we formulate a set of decidable conditions on the protocols' executions that imply formal guarantees of entity authentication for a large class of protocols. 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, i.e., any composition of compliant and validated principals is immune to attacks mounted by any protocol intruder [4] . Our technique in based on a new process calculus, named ρ-spi calculus , that we use to specify the authentication protocols of interest: ρ-spi calculus is a dialect of the spi calculus [4] and includes a set of authentication-specific constructs inspired by the process calculus for protocol narrations *Lysa* [6].

In this paper, we apply the above mentioned technique to some well-known protocols: Iso Symmetric Key Two Pass Unilateral Authentication Protocol [21], the nonce-based version of the Wide Mouthed Frog Protocol [4], the Woo and Lam Authentication Protocols [29] and the Amended Needham Schroeder Shared-Key Protocol [28]. Our aim is to show that the analysis is applicable to a wide set of authentication protocols. Based on our analysis, we prove the safety of correct protocols for an unbounded number of sessions and, in some cases, we propose simplifications. On the other side, our analysis fails in validating flawed protocols suggesting possible attacks. Moreover, we show that our analysis helps clarifying and formalizing the logics behind the protocols. We give also an example of a multi-protocol system, where entities run multiple sessions of different protocols. We analyze the interleavings among different sessions and discuss how our analysis guarantees their safety.

**Structure of the paper.** §2 gives a brief outline of ρ-spi calculus and its operational semantics. §3 summarizes our static analysis, and its main properties. § 4 presents some case studies. In § 5 we give an example of multi-protocol system and in § 6

[4] We implicitly appeal to the standard Dolev-Yao intruder model [14]. Intruders may intercept, reply and forge new messages, but never decrypt messages without knowing the corresponding keys.

some concluding remarks.

## 2 The ρ-spi calculus

The ρ-spi calculus derives from the spi calculus [4], and inherits many of the features of *Lysa*, a version of the spi calculus proposed in [6] for the analysis of authentication protocols. The ρ-spi calculus differs from both calculi in several respects: $(i)$ it incorporates the notion of tagged message exchange from [9], $(ii)$ it provides primitives for declaring process identities and (shared) long-term keys, $(iii)$ it provides new useful authentication-specific constructs.

**Table 1** Syntax

**Notation**: $d$ ranges over simple names and simple variables, $I$ over identity labels, $A, B$ over principal labels, $T$ over TTP labels, $C$ over tags: $\{\mathsf{Claim}, \mathsf{Owner}, \mathsf{Verif}, \mathsf{Key}, \mathsf{Id}\}$,

| $D ::=$ *Data* | | $P, Q ::=$ *Processes* | |
|---|---|---|---|
| $n$ | simple names | $I \triangleright S$ | (principal) |
| $n : C$ | tagged names | $I \triangleright !S$ | (replication) |
| $x$ | simple variables | $P \vert Q$ | (composition) |
| $x : C$ | tagged variables | $\mathsf{let}\ k = \mathsf{key}(I_1, I_2).P$ | (key assignment) |

| $S ::=$ *Sequential processes* | |
|---|---|
| $\mathbf{0}$ | (nil) |
| $\mathsf{in}(D_1, \ldots, D_m).S$ | (input) |
| $\mathsf{out}(D_1, \ldots, D_m).S$ | (output) |
| $\mathsf{new}(n).S$ | (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) |

**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. Tags are a special category of names and include roles (the three special names $\mathsf{Claim}, \mathsf{Owner}, \mathsf{Verif}$), the identity tag $\mathsf{Id}$ and the session key tag $\mathsf{Key}$ (tags are explained in Section 3 and their use is illustrated in Section 4). *Identities* $I\mathcal{D}$ are a subset of names, $I\mathcal{D} \subset N$, further partitioned into *principals* $I_{\mathcal{P}}$ and *trusted third parties* (TTPs) $I_{\mathcal{T}}$.

*Processes* (or *protocols*), ranged over by $P, Q$ are formed as parallel composition of, possibly replicated, *sequential processes*, ranged over by $S$. The process form $I \triangleright S$ represents the sequential process $S$ running on behalf of a certain entity identified by $I$. 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 calculus 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 $P$.

The reading of the process forms is as follows. Process **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, new and decryption are essentially the same as in the calculus *Lysa*. In particular, as in *Lysa*, we presuppose a unique (anonymous) public channel from/to which messages are read/sent, a technique that models well the worst-case situation of an intruder with a complete control of the network. Similarly to *Lysa*, our input and decryption constructs may test part of the message read (decrypted), by pattern-matching. Specifically, process $\mathsf{in}(D_1, \ldots, D_m).S$ reads a message composed of $m$ parts matching the respective $D_i$ and continues as $S$, where all the variables instantiated by the pattern-matching are replaced by their actual values. Process $\mathsf{out}(D_1, \ldots, D_m).S$ outputs a messages composed of $m$ parts and then continues as $S$. 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 $n : C$ is tagged). Similarly, $x : \mathsf{Claim}$ matches $n : \mathsf{Claim}$ but does not match $n : \mathsf{Verif}$.

Pattern matching also applies upon decryption. Process $\mathsf{decrypt}\ x\ \mathsf{as}\ \{D_1, \ldots, D_m\}_d.S$ continues only if $x$ contains a message encrypted with key $d$ and composed of $m$ parts that match the respective $D_i$. As for *input*, all the variables instantiated by the pattern-matching are replaced in the continuation $S$.

Unlike the spi calculus and Lysa, in ρ-spi calculus we introduce 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 $S$. This is useful for our analysis, as it eases 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 $\mathsf{commit}(I_1, I_2)$ 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].

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

$$\begin{array}{lccl} 1) & B \to A & : & n_B \\ 2) & A \to B & : & \{n_B, m\}_{k_{AB}} \end{array}$$

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 simply 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.

**Table 2** An example of protocol narration in ρ-spi calculus

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

**Operational Semantics.** The operational semantics of ρ-spi calculus is given in terms of *traces*, after [7]. The category $M$ of *Messages* is defined by the following productions: $M ::= x \mid n \mid \{M_1,\ldots,M_m\}_M \mid M : C$, and includes variables, names, and ciphertexts, possibly tagged. A trace is either empty, noted $\varepsilon$, or formed as $s :: \alpha$ where $s$ a trace and $\alpha$ and *action*. Every prefix form generates a corresponding action. The set of all possible actions, noted $Act$, includes the action $key(k,I_1,I_2)$ generated by key assignment, $I \triangleright in(M_1,\ldots,M_m)$ generated by input, $I \triangleright out(M_1,\ldots,M_m)$ by output, $I \triangleright new(n)$ by restriction, $I \triangleright \{M_1,\ldots,M_m\}_M$ by encryption and $I \triangleright \{M_1,\ldots,M_m\}_M$ by decryption, *run(A,B)* and *commit(A,B)* by ‘run’ and ‘commit’.

In ρ-spi calculus , communication is always performed from/to the environment (processes never synchronize directly). The intruder is modeled implicitly in the environment, by encoding it with rules that formalize the possible ways to manipulate intercepted messages. We denote with $T(P)$ the set of traces of $P$. The transition semantics is defined formally in [10].

**Example 2**: Consider again process $Protocol$ of the previous example. The following, is a possible trace of that process:

$$\text{key}(k_{AB},A,B) :: A \triangleright \text{new}(m) :: B \triangleright \text{new}(n_B) :: B \triangleright \text{out}(n_B) :: A \triangleright \text{in}(n_B) :: \text{run}(A,B) :: A \triangleright$$

$\text{encrypt}\{n_B,m\}_{k_{AB}} :: A \triangleright \text{out}(\{n_B,m\}_{k_{AB}}) :: B \triangleright \text{in}(\{n_B,m\}_{k_{AB}}) :: B \triangleright \text{decrypt}\{n_B,m\}_{k_{AB}} :: \text{commit}(B,A)$

Notice that this trace represents only one of the possible interleavings of *Initiator* and *Responder* actions. In this particular trace, commit$(B,A)$ is preceded by run$(A,B)$. According to the Woo and Lam idea of correspondence assertions [30], 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 multiple protocol sessions, we may consider the following process:

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

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 $Protocol_2$:

$$\begin{gathered} \text{key}(k_{AB},A,B) :: \mathbf{B} \triangleright \text{new}(m) :: B \triangleright \text{new}(n_B) :: B \triangleright \text{out}(n_B) :: \mathbf{B} \triangleright \text{in}(n_B) :: \text{run}(\mathbf{B},\mathbf{A}) :: \mathbf{B} \triangleright \text{encrypt}\{n_B,m\}_{k_{AB}} :: \\ \mathbf{B} \triangleright \text{out}(\{n_B,m\}_{k_{AB}}) :: B \triangleright \text{in}(\{n_B,m\}_{k_{AB}}) :: B \text{ decrypts } \{n_B,m\}_{k_{AB}} :: \text{commit}(B,A) \end{gathered}$$

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:

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

We conclude this section by formalizing the definition of safety based on correspondence assertions. As in [16], we say that a trace is *safe* if every $commit(B,A)$ is preceded by a distinct $run(A,B)$. A protocol guarantees *entity authentication* if all of its traces are safe.

**Definition 2.1** [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$(A,B)$.

## 3 A Compositional Proof Technique

Our analysis exploits 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 $\mathsf{Id}$, session keys, tagged with $\mathsf{Key}$, and nonces. Nonce tags are more elaborate, as they convey information on the role that nonces play in the authentication protocol: specifically, nonces may be tagged with $\mathsf{Claim}$, in messages that authenticate a claimant, with $\mathsf{Verif}$, in messages that specify an intended verifier, or with $\mathsf{Owner}$ in messages that authenticate a session key.

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

$$\mathbf{keys}(k_1,\ldots,k_n).(I_1 \rhd !S_1 \mid \ldots \mid I_m \rhd !S_m)$$

where $\mathbf{keys}(k_1,\ldots,k_n)$ represents a sequence of let binding for the long-term keys $k_1,\ldots,k_n$. The analysis proceeds by examining each process $\mathbf{keys}(k_1,\ldots,k_n).I_i \rhd S_i$, and attempts to validate $S_i$ under the key assignment determined by the prefix $\mathbf{keys}(k_1,\ldots,k_n)$.

Given a process $\mathbf{keys}(k_1,\ldots,k_n).I \rhd S$, we make the following assumptions on the way that long-term and session keys are circulated among principals and trusted third parties, namely: $(a)$ long-term keys are never circulated in clear or within encrypted packets; $(b)$ only principals 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 $\mathbf{keys}(k_1,\ldots,k_n).I \rhd S$ is $\{k_1,\ldots,k_n\}$-safe. This is fully formalized in [10].

The proof rules (tables 3 and 4) validate each principal and trusted party of the protocol, relative to a given identity, and two environments, i.e., they derive judgments of the form $I;\Gamma;\Pi \vdash S$ for the sequential process $S$ relative to the identity $I$ and the two environments $\Gamma$ and $\Pi$. The history environment $\Pi$ keeps track of each encryption, decryption, run and key assignment occurring in the sequence of actions performed by the party in question. The nonce environment $\Gamma$ holds the nonces that the party generates [5]. A nonce is first included in $\Gamma$ as *unchecked*, when it is introduced by a $\mathsf{new}$ prefix. Subsequently, the nonce may be *checked* by pattern matching, and marked as such, in rules (Authenticate Claim), (Authenticate Owner) and (Authenticate Verif) (in Table 3). The proof rules are so defined as to ensure that each nonce may be checked at most once, as desired. We will explain the intuition behind rules when applying them in our case studies.

[5] We write $\Pi(x) = enc\{\ldots\}_d$ to say 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 $\Pi(x) = enc\{\ldots\}_d$.

The *local correctness*, for a sequential process, can be established provided that the process complies with the key safety assumption stated above.

**Definition 3.1** [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 $J_i$. $P$ is *locally correct* if the judgment $I;\varnothing;\Pi_{k_1,\ldots,k_n} \vdash S$ is

**Table 3** Local correctness: Principal and TTP rules

## Claimant and Verifier Rules

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

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

Authenticate Verif
$$\frac{A;\Gamma,n:\text{checked};\Pi\vdash S \qquad \Pi(\bullet)=dec\{A:\mathsf{Id},n:\mathsf{Verif}\ldots\}_{k_{AB}} \qquad \Pi(k_{AB})=key(A,B)}{A;\Gamma,n:\text{unchecked};\Pi\vdash \mathsf{commit}(A,B).S}$$

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

Owner
$$\frac{\begin{array}{c}A;\Gamma;\Pi.y\mapsto enc\{D_1,\ldots,D_m\}_x\vdash S\\ \Pi(\bullet)=dec\{B:\mathsf{Id},n:\mathsf{Owner},x:\mathsf{Key}\ldots\}_{k_{AT}} \qquad \Pi(k_{AT})=key(A,T) \qquad \Pi=\Pi'.B\mapsto run\end{array}}{A;\Gamma;\Pi\vdash \mathsf{encrypt}\{D_1,\ldots,D_m\}_x\ \mathsf{as}\ y.S}$$

Verifier
$$\frac{A;\Gamma;\Pi.y\mapsto enc\{B:\mathsf{Id},x:\mathsf{Verif}\}_k\vdash S \qquad \Pi(k)\in\{key(A,T),key(A,B)\} \qquad \Pi=\Pi'.B\mapsto run}{A;\Gamma;\Pi\vdash \mathsf{encrypt}\{B:\mathsf{Id},x:\mathsf{Verif}\ldots\}_k\ \mathsf{as}\ y.S}$$

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

## TTP Rules

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

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

Ttp Distribute
$$\frac{T;\Gamma;\Pi.y\mapsto enc\{A:\mathsf{Id},x:\mathsf{Owner},k_s:\mathsf{Key}\}_{k_{BT}}\vdash S}{T;\Gamma;\Pi\vdash \mathsf{encrypt}\ \{A:\mathsf{Id},x:\mathsf{Owner},k_s:\mathsf{Key}\ldots\}_{k_{BT}}\ \mathsf{as}\ y.S}$$

**Table 4** Local Correctness: Generic Principal Rules

**Generic Principal Rules**

Nil

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

New

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

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

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

derivable, for $\Pi_{k_1,\ldots,k_n} = k_1 \mapsto key(I_1,J_1),\ldots,k_n \mapsto key(I_n,J_n)$, and $S$ is $\{k_1,\ldots,k_n\}$-safe.

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

**Definition 3.2** [Process Correctness] Let $P = \mathbf{keys}(k_1,\ldots,k_n).(I_1 \triangleright !S_1 \mid \cdots \mid I_m \triangleright !S_m)$. $P$ is *correct* if the process $\mathbf{keys}(k_1,\ldots,k_n).I_i \triangleright S_i$ is locally correct, for all $i \in \{1,\ldots,m\}$

Our main result states that correctness, in the sense of Definition 3.2, is a sufficient condition for safety (Definition 2.1) and, consequently, is a sufficient condition to guarantee entity authentication. Formally:

**Theorem 3.3 (Global 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 [10].*

## 4 Case Studies

In this section we analyze some well-known authentication protocols. We give the ρ-spi calculus code of the protocol and either we prove its safety or, in the case the protocol is flawed, we show how it can be fixed in order to be proved correct. In 4.1, we analyze the ISO Symmetric Key Two-Pass Unilateral Authentication Protocol [21], in 4.2 the nonce-based version of the Wide Mouthed Frog Protocol [4], in 4.3 the Woo and Lam Authentication Protocols [29] and in 4.4 the Amended Needham Schroeder Shared-Key Protocol [28].**??** the Carlsen's Secret Key Initiator Protocol [11].

### 4.1 ISO Symmetric Key Two-Pass Unilateral Authentication Protocol

In this section, we apply our analysis to the ISO Symmetric Key Two-Pass Unilateral Authentication Protocol [21]. The aim of this direct authentication protocol is to authenticate $A$ with $B$.

$$\begin{array}{lllll} (1) & B \rightarrow A & : & n_B \\ (2) & A \rightarrow B & : & \{B, n_B, m\}_{k_{AB}} \end{array}$$

In the first message, $B$ sends to $A$ a challenge. $A$ receives the nonce and encrypts it together with the identity label of $B$. $B$ receives the ciphertext, checks the freshness of the message by checking the equality between the nonce sent just before and the nonce inside the ciphertext, and, in the case the operation is successful, accepts the authentication request from $A$. The identity label $B$ inside the ciphertext avoids reflection attacks (see Section 2). The first step for analyzing this protocol by our technique, is to suitably tag it: $n_B$ is used for checking the freshness of a message where the identity label $B$ specifies the intended verifier. Thus $B$ is tagged as $\mathsf{Id}$ and $n_B$ by $\mathsf{Verif}$. Notice that tags makes clearer the role of message components.

$$\begin{array}{lllll} (1) & B \rightarrow A & : & n_B \\ (2) & A \rightarrow B & : & \{B : \mathsf{Id}, n_B : \mathsf{Verif}, m\}_{k_{AB}} \end{array}$$

The code of the protocol is in Table 5. The rule names, on the right side of every construct, summarize the steps of the analysis.

**Table 5** ISO Symmetric Key Two-Pass 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 |
| $\mathsf{in}(x).$ | INPUT | $\mathsf{out}(n_B).$ | OUTPUT |
| $\mathsf{run}(A,B).$ | RUN | $\mathsf{in}(y).$ | INPUT |
| $\mathsf{encrypt}\{B : \mathsf{Id}, x : \mathsf{Verif}, m\}_{k_{AB}}\ \mathsf{as}\ y.$ | VERIFIER | $\mathsf{decrypt}\ y\ \mathsf{as}\ \{B : \mathsf{Id}, n_B : \mathsf{Verif}, z\}_{k_{AB}}.$ | DECRYPTION |
| $\mathsf{out}(y)$ | OUTPUT | $\mathsf{commit}(B,A)$ | AUTHENTICATE VERIF |

The local correctness of $\mathsf{let}\ k_{AB} = \mathsf{key}(A,B).A \triangleright Initiator_{iso}(k_{AB},A,B)$ can be proved as in Table 6.
The hypotheses of rules (NEW), (INPUT), (RUN) and (OUTPUT) are trivially verified; the only interesting case is (VERIFIER), which is applied for proving

$$A; \{m : \text{unchecked}\}; k_{AB} \mapsto key(A,B).B \mapsto run \vdash \mathsf{encrypt}\{B : \mathsf{Id}, x : \mathsf{Verif}, m\}_{k_{AB}}\ \mathsf{as}\ y.\mathsf{out}(y)$$

Rules (CLAIMANT), (VERIFIER) and (OWNER) formalize the ways in which $A$ may declare her willingness to authenticate with $B$. This is why they require that $\mathsf{run}(A,B)$

**Table 6** $\mathit{Initiator}_{iso}(k_{AB}, A, B)$: proof of safety

| | |
|---|---|
| $A;\emptyset;k_{AB} \mapsto \mathit{key}(A,B) \vdash$ | $\mathsf{new}(m).\mathsf{in}(x).\mathsf{run}(A,B).\mathsf{encrypt}\{B:\mathsf{Id}, x:\mathsf{Verif}, m\}_{k_{AB}}$ as $y.\mathsf{out}(y)$ |
| | $\downarrow$ (New) |
| $A;\{m:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B) \vdash$ | $\mathsf{in}(x).\mathsf{run}(A,B).\mathsf{encrypt}\{B:\mathsf{Id}, x:\mathsf{Verif}, m\}_{k_{AB}}$ as $y.\mathsf{out}(y)$ |
| | $\downarrow$ (Input) |
| $A;\{m:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B) \vdash$ | $\mathsf{run}(A,B).\mathsf{encrypt}\{B:\mathsf{Id}, x:\mathsf{Verif}, m\}_{k_{AB}}$ as $y.\mathsf{out}(y)$ |
| | $\downarrow$ (Run) |
| $A;\{m:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B).B \mapsto \mathit{run} \vdash$ | $\mathsf{encrypt}\{B:\mathsf{Id}, x:\mathsf{Verif}, m\}_{k_{AB}}$ as $y.\mathsf{out}(y)$ |
| | $\downarrow$ (Verifier) |
| $A;\{m:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B).B \mapsto \mathit{run}.$ | |
| $y \mapsto \mathit{enc}\{B:\mathsf{Id}, x:\mathsf{Verif}, m\}_{k_{AB}} \vdash$ | $\mathsf{out}(y)$ |
| | $\downarrow$ (Output) |
| $A;\{m:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B).B \mapsto \mathit{run}.$ | |
| $y \mapsto \mathit{enc}\{B:\mathsf{Id}, x:\mathsf{Verif}, m\}_{k_{AB}} \vdash$ | $\mathbf{0}$ |
| | (Nil) |

**Table 7** $\mathit{Responder}_{iso}(k_{AB}, B, A)$: proof of safety

| | |
|---|---|
| $B;\emptyset;k_{AB} \mapsto \mathit{key}(A,B) \vdash$ | $\mathsf{new}(n_B).\mathsf{in}(y).\mathsf{decrypt}\ y$ as $\{B:\mathsf{Id}, n_B:\mathsf{Verif}, z\}_{k_{AB}}.\mathsf{commit}(B,A)$ |
| | $\downarrow$ (New) |
| $B;\{n_B:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B) \vdash$ | $\mathsf{in}(y).\mathsf{decrypt}\ y$ as $\{B:\mathsf{Id}, n_B:\mathsf{Verif}, z\}_{k_{AB}}.\mathsf{commit}(B,A)$ |
| | $\downarrow$ (Input) |
| $B;\{n_B:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B) \vdash$ | $\mathsf{decrypt}\ y$ as $\{B:\mathsf{Id}, n_B:\mathsf{Verif}, z\}_{k_{AB}}.\mathsf{commit}(B,A)$ |
| | $\downarrow$ (Decryption) |
| $B;\{n_B:\text{unchecked}\};k_{AB} \mapsto \mathit{key}(A,B).$ | |
| $y \mapsto \mathit{dec}\{B:\mathsf{Id}, n_B:\mathsf{Verif}, z\}_{k_{AB}} \vdash$ | $\mathsf{commit}(B,A)$ |
| | $\downarrow$ (Authenticate Verif) |
| $B;\{n_B:\text{checked}\};k_{AB} \mapsto \mathit{key}(A,B).$ | |
| $y \mapsto \mathit{dec}\{B:\mathsf{Id}, n_B:\mathsf{Verif}, z\}_{k_{AB}} \vdash$ | $\mathbf{0}$ |
| | (Nil) |

has been previously executed. In particular, in (Verifier), $A$ may request $B$ to authenticate herself, by sending a message $\{B:\mathsf{Id}, x:\mathsf{Verif}\ldots\}_{k_{AB}}$ to $B$ using a long-term key $k_{AB}$ shared with $B$. The role Verif is used, as expected, to inform $B$ that he is the intended verifier of the current authentication session. Rule (Verifier) can be also used by $A$ to inform a TTP $T$ that $B$ is the intended verifier. In that case the ciphertext is encrypted with a long-term key shared between $A$ and $T$ (see Sections 4.2 and 4.3).

(Verifier) requires that $B \mapsto \mathit{run}$ is the last element in the history environment and that $k_{AB} \mapsto \mathit{key}(A,B)$ belongs to the history environment, which are both true. As far as $\mathsf{let}\ k_{AB} = \mathsf{key}(A,B).\ \mathit{Responder}_{iso}(k_{AB}, B, A)$ is concerned, local correctness can be proved as in Table 7.

Proving the local correctness of $\mathsf{let}\ k_{AB} = \mathsf{key}(A,B).B \triangleright Responder_{iso}(k_{AB},B,A)$ is straightforward. The only interesting case is (Authenticate Verif), which is applied for proving

$$B;\{n_B : \text{unchecked}\};k_{AB} \mapsto key(A,B).y \mapsto dec\{B : \mathsf{Id}, n_B : \mathsf{Verif}, z\}_{k_{AB}} \vdash \text{commit}(B,A)$$

Rules (Authenticate Claim), (Authenticate Owner) and (Authenticate Verif) formalize the three possible ways for a principal $A$ to authenticate another principal $B$. Specifically, (Authenticate Verif) states that $A$ may legally commit (hence authenticate) $B$ only if $A$ has previously generated a nonce $n$ and decrypted a message $\{A : \mathsf{Id}, n : \mathsf{Verif} \dots\}_{k_{AB}}$ (with the same nonce) with $k$ shared with $B$.

(Authenticate Verif) hypotheses hold since $y \mapsto dec\{B : \mathsf{Id}, n_B : \mathsf{Verif}, z\}_{k_{AB}}$ belongs to the history environment and $n_b$:unchecked to the nonce environment. Notice that this rule checks the nonce: this is why the binding of $n_B$ moves to checked. Moreover, as required by (Authenticate Verif), the history environment contains also $k_{AB} \mapsto key(A,B)$. Since (New), (Output), (Input) and (Decryption) are trivially satisfied, $B;\emptyset;k_{AB} \mapsto key(A,B) \vdash Responder_{iso}\ (k_{AB},B,A)$.

Finally, the protocol uses no session keys and long term keys are neither sent on the net nor encrypted. Thus both the initiator and responder code are key safe and, hence, can be judged correct. By Theorem 3.3, we directly obtain that every trace of $Protocol_{iso}$ is safe, thus no authentication attack is possible. This example clarifies how direct authentication protocols are analyzed by our proof system: the idea is that a $\mathsf{commit}(B,A)$ can be proved only if a ciphertext of a particular form has been previously decrypted. That ciphertext must have been encrypted by following one of the principles, in this case (Verifier). This guarantees that a $\mathsf{run}(A,B)$ has been previously asserted. Since nonces can be checked only once, in every trace generated by the protocol, every $commit(B,A)$ is preceded by a *distinct* $run(A,B)$.

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 even when sequential processes $Initiator_{iso}$ and $Responder_{iso}$ are arbitrarily replicated.

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

$$\begin{aligned} Protocol_{iso} - m \triangleq\ & \mathsf{let}^{m}_{i,j=1,i<j} k_{ij} = \mathsf{key}(I_i,I_j). \\ & (|^{m}_{i,j=1,i\neq j} I_i \triangleright !Initiator_{iso}(k_{ij},I_i,I_j) | I_i \triangleright !Responder_{iso}(k_{ij},I_i,I_j))) \end{aligned}$$

$k_{ij}$ 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 $i < j$). $Initiator_{iso}$ and $Responder_{iso}$ are arbitrarily replicated for every pair of entities $I_i$, $I_j$, using the appropriate long-term key. The proofs of correctness of processes $\mathsf{let}^m_{i,j=1,i<j} k_{ij} = \mathsf{key}(I_i,I_j).I_i \triangleright \mathit{Initiator}_{iso}(K_{ij},I_i,I_j)$ and $\mathsf{let}^m_{i,j=1,i<j} k_{ij} = \mathsf{key}(I_i,I_j).I_i \triangleright \mathit{Responder}_{iso}(K_{ij},I_i,I_j)$ are the same as those for the process $\mathsf{let}\ k_{AB} = \mathsf{key}(k_{AB},A,B).\ A \triangleright \mathit{Initiator}_{iso}(k_{AB},A,B)$ and for $\mathsf{let}\ k_{AB} = \mathsf{key}\ (k_{AB},A,B).\ B \triangleright \mathit{Responder}_{iso}(k_{AB},B,A)$, up to a renaming of identities and keys. As specified by Definition 3.2, replicated identical processes do not need to be re-proved correct.

## *A variant of the ISO Symmetric Key Two-Pass Unilateral Authentication Protocol*

A slight variant of the Iso Symmetric Key Unilateral Authentication Protocol is the following:

$$\begin{array}{llcl} (1) & B \rightarrow A & : & n_B \\ (2) & A \rightarrow B & : & \{A, n_B, m\}_{K_{AB}} \end{array}$$

The only difference is the identity label in $(2)$: $A$ instead of $B$, namely the claimant label instead of the intended verifier one. The tagged version of the protocol is:

$$\begin{array}{llcl} (1) & B \rightarrow A & : & n_b \\ (2) & A \rightarrow B & : & \{A:\mathsf{Id}, n_B:\mathsf{Claim}, m\}_{K_{AB}} \end{array}$$

The protocol can be analyzed similarly to the protocol of Table 5. The only difference is that the encryptions with a nonce tagged by Claim are checked by rule (Claimant) instead of rule (Verifier). Moreover the commit generated by the decryption of such ciphertexts are proved by (Authenticate Claim) instead of (Authenticate Verif). (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 $B$, in the case of direct authentication protocols, or with a TTP $T$, if a TTP is used for achieving authentication (see Sections 4.2 and 4.3).

## *A flawed protocol*

Let us consider again the protocol in Table 2. As discussed in Example 2, the protocol is subject to attacks when running in multiple sessions. Indeed, the flaw in the protocol's logics is revealed by failure to validate one of its components. The problem is with the responder, as we illustrate showing that the process $\mathsf{let}\ k_{AB}{=}\mathsf{key}(k_{AB},A,B).B \triangleright \mathit{Responder}(B,A)$ is not locally correct. Notice that in order for $I;\Gamma;\Pi \vdash \mathsf{commit}(B,A)$ to be derivable, one needs $I = B$. However, we easily see that $\nexists \Pi$ such that $B;\Gamma;\Pi \vdash \mathsf{commit}(B,A)$. In fact, for $B;\Gamma;\Pi \vdash \mathsf{commit}(B,A)$ to be derivable, by rules (Authenticate Claim), (Authenticate Owner), (Authenticate Verif), it has to exists $y$ such that $\Pi(y) = dec\{A:\mathsf{Id}, n_b : R\ldots\}_{k_{AB}}$, with $R \in \{\mathsf{Claim}, \mathsf{Owner}, \mathsf{Verif}\}$, but the only decrypt construct in $S$ is $\mathsf{decrypt}\ y\ \mathsf{as}\ \{n_B, z\}_{k_{AB}}$. Thus, $B;\Gamma;\Pi \nvdash \mathsf{commit}(B,A)$.

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 nonce, the hypotheses of the two authentication rules would not hold, since the ciphertext is missing the identity label $A$. The attack presented in Section 2 exploits precisely this flaw.

## 4.2 The nonce-based version of the Wide Mouthed Frog Protocol

In this section, we apply our technique for proving the correctness of the nonce-based version of the *Wide Mouthed Frog Protocol* (WMF) [4], as presented in [16]. The protocol is composed of the following six messages:

**Table 8** Wide Mouthed Frog Protocol in ρ-spi calculus

| | | |
|---|---|---|
| $Protocol_{WMF} \triangleq$ | $Server_{WMF}(k_i,I_i,k_j,I_j) \triangleq$ | |
| let $k_i = \mathsf{key}^m_{i=1}(I_i,I_0)$. | $\mathsf{new}(n_T)$ | NEW |
| $(\vert^m_{i,j=1,\ i\neq j}\ I_i \triangleright !Initiator_{WMF}(k_i,I_i,I_j) \vert$ | $\mathsf{in}(I_i)$. | INPUT |
| | $\mathsf{out}(n_T)$. | OUTPUT |
| $I_j \triangleright !Responder_{WMF}(k_j,I_j,I_i) \vert$ | $\mathsf{in}(I_i,x_i)$. | INPUT |
| $I_0 \triangleright !Server_{WMF}(k_i,I_i,k_j,I_j))$ | decrypt $x_i$ as $\{I_j : \mathsf{Id}, x_{ij}, n_T : \mathsf{Verif}\}_{k_i}$. | DECRYPTION |
| | $\mathsf{in}(x_B)$. | INPUT |
| | encrypt$\{I_i : \mathsf{Id}, x_{ij}, x_B : \mathsf{Claim}\}_{k_j}$ as $x$. | TTP FORW. & CHECK |
| | $\mathsf{out}(x)$ | OUTPUT |

| | | | |
|---|---|---|---|
| $Initiator_{WMF}(k_i,I_i,I_j) \triangleq$ | | $Responder_{WMF}(k_j,I_j,I_i) \triangleq$ | |
| $\mathsf{new}(k_{ij})$ | NEW | $\mathsf{new}(n_B)$ | NEW |
| $\mathsf{out}(I_i)$. | OUTPUT | $\mathsf{out}(n_B)$. | OUTPUT |
| $\mathsf{in}(x_T)$. | INPUT | $\mathsf{in}(x)$. | INPUT |
| $\mathsf{run}(I_i,I_j)$. | RUN | decrypt $x$ as $\{I_i : \mathsf{Id}, x_{ab}, n_B : \mathsf{Claim}\}_{k_j}$ | DECRYPTION |
| encrypt$\{I_j : \mathsf{Id}, k_{ij}, x_T : \mathsf{Verif}\}_{k_i}$ as $x$. | VERIF | $\mathsf{commit}(I_j,I_i)$ | AUTH CLAIM |
| $\mathsf{out}(I_i,x)$ | OUTPUT | | |

$$
\begin{array}{llll}
(1) & A \rightarrow T & : & A \\
(2) & T \rightarrow A & : & n_T \\
(3) & A \rightarrow T & : & A, \{B, k_{AB}, n_T\}_{k_{AT}} \\
(4) & T \rightarrow B & : & * \\
(5) & B \rightarrow T & : & n_B \\
(6) & T \rightarrow B & : & \{A, k_{AB}, n_B\}_{k_{BT}}
\end{array}
$$

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 $B$. By checking $n_T$, $T$ is ensured about the freshness of the received ciphertext. Identity label $A$ (sent as plaintext) is a hint for choosing the correct decryption key. The (void) fourth message is just a request from $T$ of starting 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. With the last message, $T$ communicates to $B$ that $A$ is willing to share the key $k_{AB}$ with him.

We give a specification of the WMF protocol in ρ-spi calculus in Table 8 . $Protocol_{WMF}$ 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 let $k_i$=key$(I_i, I_0)$ iterated for all $i = 1, \ldots, m$. We assume $I_0 \in \mathcal{I}_{\mathcal{T}}$ and $I_i \in \mathcal{I}_{\mathcal{P}}$ 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).

The rule names, on the right side of every construct, show how to verify the local correctness of every sequential component. We briefly discuss the most interesting cases:

$Initiator_{WMF}(k_i, I_i, I_j)$ The only interesting case is (Verifier) which is applied for proving $I_i; \Gamma_i; \Pi_i \vdash$ encrypt$\{I_j : \mathsf{Id}, k_{ij}, x_T : \mathsf{Verif}\}_{k_i}$ as $x$.out$(I_i, x)$. All the hypotheses are satisfied since $\Pi_i = \Pi'_i.I_j \mapsto run$ and $\Pi_i(k_i) = key(I_i, I_0)$.

$Responder_{WMF}(k_j, I_j, I_i)$ The only interesting case is (Authenticate Claim) which is applied for proving $I_j; \Gamma_j; \Pi_j \vdash$ commit$(I_j, I_i)$. All the hypotheses hold since $\Pi_j(x) = dec\{I_i : \mathsf{Id}, x_{ab}, n_B : \mathsf{Claim}\}_{k_j}$ and $\Pi_j(k_j) = key(I_j, I_0)$. Moreover $\Gamma_j(n_B) =$ unchecked, since $n_B$ has not been checked before..

$Server_{WMF}(k_i, I_i, k_j, I_j)$ The only interesting case is (Ttp Forward & Check) which is applied for proving $I_0; \Gamma_0; \Pi_0 \vdash$ encrypt$\{I_i : \mathsf{Id}, x_{ij}, x_B : \mathsf{Claim}\}_{k_j}$ as $x$. out(x). Rules (Ttp Forward & Check), (Ttp Forward) and (Ttp Distribute) govern the behavior of TTPs. The first two rules regulate the generation of messages of the form $\{\ldots, A : \mathsf{Id}, \ldots, x : \mathsf{Claim}, \ldots\}_k$, the third the generation of session keys. Specifically, in rule (Ttp Forward & Check), $T$ generates a message $\{\ldots, A : \mathsf{Id}, \ldots, x : \mathsf{Claim}, \ldots\}_{k_{BT}}$ if it has previously decrypted (and checked the nonce $n$ of) a message of the form $\{\ldots, B : \mathsf{Id}, \ldots, n : \mathsf{Verif}, \ldots\}_{k_{AT}}$, and $k_{AT}$ and $k_{BT}$ are shared with $A$ and $B$ respectively. All the hypotheses hold since $\Pi_0(x_i) = dec\{I_j : \mathsf{Id}, x_{ij}, n_T : \mathsf{Verif}\}_{k_i}$ and $\Gamma_0(n_T) =$ unchecked. Moreover $\Pi_0(k_i) = key(I_i, I_0)$ and $\Pi_0(k_j) = key(I_j, I_0)$.

Key Safety is trivially satisfied, as long term keys are never extruded and no session key is tagged by Key: this means no session key is used for authentication. Thus, $Protocol_{WMF}$ is correct.

*Type Flaws and Tagging*

Actually the protocol discussed above is affected by the following type flaw attack:

$$\begin{array}{lll}
(1.b) & A \rightarrow E(T) & : \quad A \\
(4.a) & E(T) \rightarrow A & : \quad * \\
(5.a) & A \rightarrow E(T) & : \quad n_A \\
(2.b) & E(T) \rightarrow A & : \quad n_A \\
(3.b) & A \rightarrow E(T) & : \quad A, \{B, k_{AB}, n_A\}_{K_{AT}} \\
(6.a) & E(T) \rightarrow A & : \quad \{B, k_{AB}, n_A\}_{K_{BT}}
\end{array}$$

The sessions $a$ and $b$ are interleaved. The enemy waits for an authentication request from $A$, $(1.b)$. Then the enemy impersonates $T$ and starts a new session with $A$, $(4.a)$, who generates a new nonce $n_A$, $(5.a)$. The enemy, by impersonating the TTP $T$, sends back to $A$ the nonce $n_A$, $(2.b)$. $A$ generates the ciphertext where the verifier of the authentication session is specified, $(3.b)$. The enemy intercepts the ciphertext. Notice that the form of the ciphertext in $(3)$ is the same as the ciphertext in $(6)$. Thus the enemy can reply to $A$ the ciphertext generated by $A$ herself, $(6.a)$. $A$ believes that the identity label $B$ represents the claimant of the authentication session and accepts the (false) authentication request from $B$.

Notice that the attack is possible only on the untagged version of the protocol: when messages are tagged, the ciphertext sent in (3.b) differs from the one that $B$ expects to receive in $(6.b)$ because of the different tag. This shows how tagging solves message ambiguities. Indeed, we assume that protocol messages are effectively tagged when implemented. Notice that the same protocol, with tagged ciphertexts, is type-checked and considered safe also in [16]. The tagging used there adds a different label to each encrypted protocol message, so that ciphertexts cannot be confused. That tagging is very reasonable and can be easily incorporated in protocol implementations. Our tagging is strictly less demanding: we do not require that every message is unambiguously tagged since we tag only certain components: two ciphertexts of two different protocol messages could be confused if they have the same tags. As a consequence, if the protocol is implemented with the reasonable tagging technique used in [16], our tags can be safely removed without compromising the protocol safety.

### 4.3 The Woo and Lam Authentication Protocols

The following protocols are proposed by Woo and Lam in [29]. They start by protocol $\Pi_f$ and, step by step, simplify it to $\Pi$. The final simplification results in a flawed protocol.
Protocol $\Pi_f$:

$$\begin{array}{ll}
(1)\ A \to B: & A \\
(2)\ B \to A: & n_B \\
(3)\ A \to B: & \{A,B,n_B\}_{k_{AT}} \\
(4)\ B \to T: & \{A,B,n_B,\{A,B,n_B\}_{k_{AT}}\}_{k_{BT}} \\
(5)\ T \to B: & \{A,B,n_B\}_{k_{BT}}
\end{array}$$

The aim of this protocol is to authenticate $A$ with $B$. $A$ starts the protocol by sending her own identity label to $B$, $(1)$. $B$ responds with a challenge nonce $n_B$, $(2)$. That nonce is encrypted by $A$, with the long term key shared with the TTP $T$, together with her own identity label and the identity label of $B$, $(3)$. $A$ represents the claimant of the authentication session, while $B$ the intended verifier. $B$ receives the ciphertext and encrypts it, with the long term key shared with $T$, together with the two identity labels $A$ and $B$ and the nonce $n_B$. By that ciphertext, $B$ asks $T$ for confirming the authentication request from $A$. $T$ receives the ciphertext, and encrypts the two identity labels and the nonce $n_B$ with the long term key shared with $B$, $(5)$. $B$ receives the ciphertext, checks its freshness and accepts the authentication request of $A$.
Protocol $\Pi_1$, obtained from $\Pi_f$ by removing the first occurrence of $n_B$ from $(4)$:

$$\begin{array}{ll}
(1)\ A \to B: & A \\
(2)\ B \to A: & n_B \\
(3)\ A \to B: & \{A,B,n_B\}_{k_{AT}} \\
(4)\ B \to T: & \{A,B,\{A,B,n_B\}_{k_{AT}}\}_{k_{BT}} \\
(5)\ T \to B: & \{A,B,n_B\}_{k_{BT}}
\end{array}$$

Protocol $\Pi_2$, obtained from $\Pi_1$ by removing every occurrence of $B$:

$$\begin{array}{ll}
(1)\ A \to B: & A \\
(2)\ B \to A: & n_B \\
(3)\ A \to B: & \{A,n_B\}_{k_{AT}} \\
(4)\ B \to T: & \{A,\{A,n_B\}_{k_{AT}}\}_{k_{BT}} \\
(5)\ T \to B: & \{A,n_B\}_{k_{BT}}
\end{array}$$

Protocol $\Pi_3$, obtained from $\Pi_2$ by removing the identity label $A$ from $(3)$ and the second occurrence of $A$ from $(4)$:

$$\begin{array}{ll}
(1)\ A \rightarrow B: & A \\
(2)\ B \rightarrow A: & n_B \\
(3)\ A \rightarrow B: & \{n_B\}_{k_{AT}} \\
(4)\ B \rightarrow T: & \{A, \{n_B\}_{k_{AT}}\}_{k_{BT}} \\
(5)\ T \rightarrow B: & \{A, n_B\}_{k_{BT}}
\end{array}$$

Protocol $\Pi$, obtained from $\Pi_3$ by removing the identity label $A$ from $(5)$:

$$\begin{array}{ll}
(1)\ A \rightarrow B: & A \\
(2)\ B \rightarrow A: & n_B \\
(3)\ A \rightarrow B: & \{n_B\}_{k_{AT}} \\
(4)\ B \rightarrow T: & \{A, \{n_B\}_{k_{AT}}\}_{k_{BT}} \\
(5)\ T \rightarrow B: & \{n_B\}_{k_{BT}}
\end{array}$$

An attack on $\Pi$ is reported in [29]. A list of attacks on $\Pi, \Pi_1, \Pi_2, \Pi_3$ and $\Pi_f$ can be found in [12]. We report from [5] an attack on $\Pi_2$ (and consequently on $\Pi_3$ and $\Pi$):

$$\begin{array}{ll}
(1.a)\ A \rightarrow I(C): & A \\
(1.b)\ I(A) \rightarrow B: & A \\
(2.b)\ B \rightarrow I(A): & n_B \\
(2.a)\ I(C) \rightarrow A: & n_B \\
(3.a)\ A \rightarrow I(C): & \{A, n_B\}_{k_{AT}} \\
(3.b)\ I(A) \rightarrow B: & \{A, n_B\}_{k_{AT}} \\
(4.b)\ B \rightarrow T: & \{A, \{A, n_B\}_{k_{AT}}\}_{k_{BT}} \\
(5.b)\ T \rightarrow B: & \{A, n_B\}_{k_{BT}}
\end{array}$$

The attack presupposes an authentication request from $A$ to $C$ $(1.a)$, an arbitrary principal different from $B$. The enemy begins another authentication session with $B$, $(1.b)$, by impersonating $A$. $B$ sends to $A$ the nonce $n_B$, $(2.b)$, actually intercepted by the attacker and replicated back to $A$, $(2.a)$. $A$ encrypts in the ciphertext, $(3.a)$, the nonce received with her own identity (here is the problem: $A$ wants to authenticate herself with $B$, but in the ciphertext there is no information regarding $B$) and sends the ciphertext to $C$. The ciphertext is captured by the attacker and routed to $B$, $(3.b)$. $B$ sends the ciphertext to $T$, $(4.b)$, and the server completes the protocol with the last message exchange, $(5.b)$. $B$ accepts the authentication request from $A$ but $A$ wanted to authenticate herself with $C$. The tagged version of the protocol follows:

|  |  |
|---|---|
| (1) $A \rightarrow B:$ | $A$ |
| (2) $B \rightarrow A:$ | $n_B$ |
| (3) $A \rightarrow B:$ | $\{A:\mathsf{Id}, n_B:\mathsf{Claim}\}_{k_{AT}}$ |
| (4) $B \rightarrow T:$ | $\{A:\mathsf{Id}, \{A:\mathsf{Id}, n_B:\mathsf{Claim}\}_{k_{AT}}\}_{k_{BT}}$ |
| (5) $T \rightarrow B:$ | $\{A:\mathsf{Id}, n_B:\mathsf{Claim}\}_{k_{BT}}$ |

**Table 9** Simplified Woo and Lam Protocol in ρ-spi calculus

$Protocol_{WL} \triangleq$

$\mathsf{let}\ k_i = \mathsf{key}_{i=1}^{m}(I_i, I_0).$

$$(|_{\substack{i,j=1\\ i\neq j}}^{m}\ I_i \triangleright !Initiator_{WL}(k_i, I_i, I_j)|$$
$$I_i \triangleright !Responder_{WL}(k_i, I_j, I_i)|$$
$$I_0 \triangleright !Server_{WL}(k_i, I_i, k_j, I_j))$$

$Initiator_{WL}(k_i, I_i, I_j) \triangleq$

|  |  |
|---|---|
| $\mathsf{out}(I_i).$ | OUTPUT |
| $\mathsf{in}(x_B).$ | INPUT |
| $\mathsf{run}(I_i, I_j).$ | RUN |
| $\mathsf{encrypt}\{I_j:\mathsf{Id}, x_B:\mathsf{Verif}\}_{k_i}\ \mathsf{as}\ x.$ | VERIFIER |
| $\mathsf{out}(x)$ | OUTPUT |

$Server_{WL}(k_i, I_i, k_j, I_j) \triangleq$

|  |  |
|---|---|
| $\mathsf{in}(I_i, z).$ | INPUT |
| $\mathsf{decrypt}\ z\ \mathsf{as}\ \{I_j:\mathsf{Id}, x_i:\mathsf{Verif}\}_{k_i}.$ | DECRYPTION |
| $\mathsf{encrypt}\{I_i:\mathsf{Id}, x_i:\mathsf{Claim}\}_{k_j}\ \mathsf{as}\ y.$ | TTP FORWARD |
| $\mathsf{out}(y)$ | OUTPUT |

$Responder_{WL}(K_j, I_j, I_i) \triangleq$

|  |  |
|---|---|
| $\mathsf{new}(n_B)$ | NEW |
| $\mathsf{out}(n_B).$ | OUTPUT |
| $\mathsf{in}(x).$ | INPUT |
| $\mathsf{out}(I_j, x).$ | OUTPUT |
| $\mathsf{in}(y).$ | INPUT |
| $\mathsf{decrypt}\ y\ \mathsf{as}\ \{I_i:\mathsf{Id}, n_B:\mathsf{Claim}\}_{k_j}$ | DECRYPTION |
| $\mathsf{commit}(I_j, I_i)$ | AUTH CLAIM |

We cannot check this protocol since $A$ specifies her own identity in the ciphertext encrypted with the long term key owned by $A$ and $T$ (Message 3). No rule allows this kind of encryption, since the verifier identity is missing and the ciphertext is useless for completing a safe authentication session. Moreover the server encrypts a ciphertext, with the long term key owned by $B$ and $T$, where a nonce tagged by Claim and $A$ by Id appear. There is no rule for this kind of encryption, since the server has no evidence about the willingness of $A$ to authenticate with $B$.

The attacks on $\Pi_f$ and $\Pi_1$ are based on type flaws and can be prevented by tagging. Indeed, we are able to check in ρ-spi calculus only $\Pi_f$ and $\Pi_1$. The tagged version of $\Pi_1$ is the following:

|  |  |
|---|---|
| (1) $A \rightarrow B:$ | $A$ |
| (2) $B \rightarrow A:$ | $n_B$ |
| (3) $A \rightarrow B:$ | $\{A, B:\mathsf{Id}, n_B:\mathsf{Verif}\}_{k_{AT}}$ |
| (4) $B \rightarrow T:$ | $\{A, B, \{A, B:\mathsf{Id}, n_B:\mathsf{Verif}\}_{k_{AT}}\}_{k_{BT}}$ |
| (5) $T \rightarrow B:$ | $\{A:\mathsf{Id}, B, n_B:\mathsf{Claim}\}_{k_{BT}}$ |

For checking the protocol, the identity label $B$ in $(3)$ is required together with the nonce tagged by $\mathsf{Verif}$. This confirms that the attack to $\Pi_2$ arises because $A$ does not specify the intended verifier in the ciphertext $(3)$ and the server has no way to derive which principal the authentication session is directed to.

Our rules suggest a simplification of the protocol: since the identity label $A$ in $(3)$ is not tagged, it can be safely removed as well as the outside ciphertext in $(4)$. The resulting protocol is :

$$\begin{array}{lll}
(1) & A \rightarrow B: & A \\
(2) & B \rightarrow A: & n_B \\
(3) & A \rightarrow B: & \{B:\mathsf{Id}, n_B:\mathsf{Verif}\}_{k_{AT}} \\
(4) & B \rightarrow T: & A, \{B:\mathsf{Id}, n_B:\mathsf{Verif}\}_{k_{AT}} \\
(5) & T \rightarrow B: & \{A:\mathsf{Id}, n_B:\mathsf{Claim}\}_{k_{BT}}
\end{array}$$

This simplification is also suggested in [16]. The ρ-spi calculus specification of the protocol is given in Table 9. In order to check the correctness of the server, Ttp Forward is applied. Such a rule states that $T$ 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 $\{B:\mathsf{Id}, x:\mathsf{Verif}, \ldots\}_{k_{AT}}$ ($k_{AT}$ and $k_{BT}$ are shared with $A$ and $B$ respectively); notice that $x$ is forwarded to $B$ so that $B$ can check the freshness of the first message.

## 4.4 The Amended Needham Schroeder Shared-Key Protocol

The original version of this protocol was proposed in [28] and was affected by different attacks. Some of them rely on cryptographic assumptions [8], other on the protocol's logics [13]. A correct version of the protocol is the Amended Needham-Schroeder Protocol, suggested by Needham and Schroeder in [25]:

$$\begin{array}{lll}
(1) & A \rightarrow B: & A \\
(2) & B \rightarrow A: & \{A, n_B^0\}_{k_{BT}} \\
(3) & A \rightarrow T: & A, B, n_A, \{A, n_B^0\}_{k_{BT}} \\
(4) & T \rightarrow A: & \{n_A, B, k_{AB}, \{k_{AB}, n_B^0, A\}_{k_{BT}}\}_{k_{AT}} \\
(5) & A \rightarrow B: & \{k_{AB}, n_B^0, A\}_{k_{BT}} \\
(6) & B \rightarrow A: & \{n_B\}_{k_{AB}} \\
(7) & A \rightarrow B: & \{n_B - 1\}_{k_{AB}}
\end{array}$$

The tagged version of the protocol may help understanding how our system works with authentication through session keys, and clarifying the protocol's logics:

| | | |
|---|---|---|
| (1) | $A \to B :$ | $A$ |
| (2) | $B \to A :$ | $\{A, n_B^0\}_{k_{BT}}$ |
| (3) | $A \to T :$ | $A, B, n_A, \{A, n_B^0\}_{k_{BT}}$ |
| (4) | $T \to A :$ | $\{n_A : \mathsf{Owner}, B : \mathsf{Id}, k_{AB} : \mathsf{Key}, \{k_{AB} : \mathsf{Key}, n_B^0 : \mathsf{Owner}, A : \mathsf{Id}\}_{k_{BT}}\}_{k_{AT}}$ |
| (5) | $A \to B :$ | $\{k_{AB} : \mathsf{Key}, n_B^0 : \mathsf{Owner}, A : \mathsf{Id}\}_{k_{BT}}$ |
| (6) | $B \to A :$ | $\{n_B\}_{k_{AB}}$ |
| (7) | $A \to B :$ | $\{n_B - 1\}_{k_{AB}}$ |

$A$ starts the protocol sending her own identity label to $B$ as cleartext, $(1)$. In $(2)$, $B$ encrypts, with the long term key shared with $T$, a fresh nonce $n_B^0$ together with the identity label $A$. By this ciphertext, $B$ communicates to $T$ he is willing to authenticate himself with $A$. In $(3)$, $A$ sends to $T$ the ciphertext received by $B$ and, as cleartext, also a fresh nonce $n_A$ and the two identity labels $A$ and $B$. The server distributes, by means of two ciphertexts, a session key $k_{AB}$ to $A$ and $B$, $(4)$. The identity label $B$ is inserted in the ciphertext encrypted by $k_{AT}$ for communicating to $A$ that $B$ is the owner of the session key $k_{AB}$. Similarly the identity label $A$, in the ciphertext encrypted by $k_{BT}$, informs $B$ that $A$ is the owner of $k_{AB}$. Thus $n_A$ and $n_B$ are tagged by $\mathsf{Owner}$ and $k_{AB}$ by $\mathsf{Key}$. In $(6)$, $A$ receives a ciphertext encrypted with $k_{AB}$. By the nonce-check performed on the ciphertext received in $(4)$, $A$ knows that $k_{AB}$ is fresh. Since $A$ has not generated any ciphertext encrypted with $k_{AB}$ before, that ciphertext has been originated by $B$. Thus $B$ is alive and $A$ can authenticate him. Similarly, in $(7)$, $B$ receives a ciphertext encrypted with $k_{AB}$, different from the one encrypted in $(6)$. Thus that ciphertext has been originated by $A$ and $B$ is allowed to authenticate her. Let us suppose to provide ρ-spi calculus with integer numbers as spi calculus [4], and specifically to add prev($d$) and succ($d$) to data. The specification of the protocol is in Table 10. The protocol passes our analysis and, hence, is safe. We briefly discuss the interesting rules of every sequential component:

**(**Ttp Distribute**)** is used for checking the safety of the server. The rule allows a TTP $T$ to declare new session keys through messages of the form $\{I : \mathsf{Id}, x : \mathsf{Owner}, k_s : \mathsf{Key} \ldots\}_k$. (Ttp Distribute) is trivially satisfied since it has no side conditions.

**(**Owner**)** is used for checking the safety of both the initiator and the responder. The rule allows $A$ to send a message $\{D_1, \ldots, D_m\}_y$ for confirming to have received the fresh session key $y$, provided that she has previously decrypted a message $\{B : \mathsf{Id}, n : \mathsf{Owner}, y : \mathsf{Key} \ldots\}_{k_{AT}}$, declaring that $y$ is a fresh key shared with $B$, and she has previously performed a run with $B$. Both the conditions are satisfied in $Initiator_{NS}(k_i, I_i, I_j)$ and $Responder_{NS}(k_j, I_j, I_i)$.

**(**Authenticate Owner**)** states that $A$ may commit $B$ if she has decrypted $(i)$ a mes-

**Table 10** Amended Needham Schroeder Shared-Key Protocol in ρ-spi calculus

| | | |
|---|---|---|
| $Protocol_{NS} \triangleq$ | $Server_{NS}(k_i,I_i,k_j,I_j). \triangleq$ | |
| let $k_i = \mathsf{key}^m_{i=1}(I_i,I_0)$. | in$(I_i,I_j,x_i,x)$. | INPUT |
| $(\vert^m_{i,j=1,\, i\neq j}\ I_i \triangleright !Initiator_{NS}(k_i,I_i,I_j)\vert$ | decrypt $x$ as $\{I_i,x_j\}_{k_j}$. | DECRYPTION |
| | new$(k_{ij})$. | NEW |
| $I_j \triangleright !Responder_{NS}(k_i,I_j,I_i)\vert$ | encrypt $\{x_j : \mathsf{Owner}, I_i : \mathsf{Id}, k_{ij} : \mathsf{Key}\}_{k_j}$ as $y_j$. | TTP DISTRIBUTE |
| $I_0 \triangleright !Server_{NS}(k_i,I_i,k_j,I_j))$ | encrypt $\{x_i : \mathsf{Owner}, I_j : \mathsf{Id}, k_{ij} : \mathsf{Key}, y_j\}_{k_i}$ as $y$. | TTP DISTRIBUTE |
| | out$(y)$. | OUTPUT |

| | | | |
|---|---|---|---|
| $Initiator_{NS}(k_i,I_i,I_j) \triangleq$ | | $Responder_{NS}(k_j,I_j,I_i) \triangleq$ | |
| out$(I_i)$. | OUTPUT | in$(I_i)$. | INPUT |
| in$(x)$. | INPUT | new$(n_B)$. | NEW |
| new$(n_A)$. | NEW | encrypt $\{I_j,n_B\}_{k_j}$ as $x$. | ENCRYPTION |
| out$(I_i,I_j,n_A,x)$. | OUTPUT | out$(x)$. | OUTPUT |
| in$(x_T)$. | INPUT | in$(x_T)$. | INPUT |
| decrypt $x_T$ as $\{n_a : \mathsf{Owner},$ | | decrypt $x_T$ as $\{n_B : \mathsf{Owner}; I_i : \mathsf{Id}, x_{ij} : \mathsf{Key}\}_{k_j}$. | DECRYPTION |
| $I_j : \mathsf{Id}, x_{ij} : \mathsf{Key}, x_B\}_{k_i}$. | DECRYPTION | new$(a)$. | NEW |
| out$(x_B)$. | OUTPUT | encrypt $\{a\}_{x_{ij}}$ as $y$. | OWNER |
| in$(y_B)$. | INPUT | out$(y)$. | OUTPUT |
| decrypt $y_B$ as $\{y\}_{x_{ij}}$ | DECRYPTION | in$(y_A)$. | INPUT |
| commit$(I_i,I_j)$. | AUTH OWNER | decrypt $y_A$ as $\{prec(a)\}_{x_{ij}}$. | DECRYPTION |
| run$(I_i,I_j)$. | RUN | commit$(I_j,I_i)$ | AUTH OWNER |
| encrypt $\{prec(y)\}_{x_{ij}}$ as $x$. | OWNER | | |
| out$(x)$ | OUTPUT | | |

sage $\{B : \mathsf{Id}, n : \mathsf{Owner}, y : \mathsf{Key} \ldots\}_{k_{AT}}$, encrypted by a TTP, including a fresh session key $y$ owned by $B$ and a nonce $n$ that $A$ previously generated; and $(ii)$ at least one message $\{D_1,\ldots,D_m\}_y$ that she did not generate [6]. (AUTHENTICATE OWNER) is satisfied in $Initiator_{NS}(k_i,I_i,I_j)$ as well as in $Responder_{NS}(k_j,I_j,I_i)$.

Notice that also key safety is satisfied, since long term keys and received session keys are never extruded, and the TTP distributes correctly the session key $k_{ij}$ to $I_i$ and $I_j$. An interesting point is that the identity label and the nonce in the ciphertext $(2)$ sent by $B$ are not tagged. This means that such an encryption is useless for the safety of $Protocol_{NS}$. Moreover, the nested encryption $(4)$ produced by the server is useless too: the ciphertexts could be safely split up. The resulting protocol is known as Carlsen's Secret Key Initiator Protocol [11] which, similarly to the Amended Needham Schroeder Shared-Key Protocol, passes our analysis.

[6] in this case the principal sends an encrypted nonce $\{n_A\}_K$ and expects to receive back the encrypted nonce decremented by one, i.e., $\{n_A - 1\}_K$.

## 5 An example of multi-protocol system

In this section we present an example of multi-protocol system. Particularly, we consider the interaction among principals running both the Wide Mouthed Frog Protocol (Section 4.2) and a slight variant of the Woo and Lam Authentication Protocol (Section 4.3):

$$\begin{array}{lll}
(1) & A \rightarrow B: & A \\
(2) & B \rightarrow A: & n_B \\
(3) & A \rightarrow B: & \{B, M, n_B\}_{k_{AT}} \\
(4) & B \rightarrow T: & A, \{B, M, n_B\}_{k_{AT}} \\
(5) & T \rightarrow B: & \{A, M, n_B\}_{k_{BT}}
\end{array}$$

This protocol differs from the simplified version of Section 4.3 only because of the presence of a payload $M$ sent by $A$ and transmitted by $T$ to $B$.

### 5.1 An untagged interleaving

Let us consider what may happen when principals run untagged versions of the protocols: the multi-protocol system is flawed. In Section 4.2 an attack on the untagged Wide Mouthed Frog Protocol has been presented: a similar attack can be performed also in a multi-protocol system. In this case $A$ acts both as initiator, following the Wide Mouthed Frog Protocol, and Responder, following the Woo and Lam Authentication Protocol:

$$\begin{array}{lll}
(1.WMF) & A \rightarrow E(T): & A \\
(1.WL) & E(B) \rightarrow A: & B \\
(2.WL) & A \rightarrow E(B): & n_B \\
(2.WMF) & E(T) \rightarrow A: & n_B \\
(3.WMF) & A \rightarrow E(T): & \{B, M, n_B\}_{k_{AT}} \\
(3.WL) & E(B) \rightarrow A: & M' \\
(4.WL) & A \rightarrow E(T): & A, M' \\
(5.WL) & E(T) \rightarrow A: & \{B, M, n_B\}_{k_{AT}}
\end{array}$$

At the end, $A$ believes that $B$ is willing to authenticate himself: $A$ accepts the authentication request from $B$ even if $B$ is not present in the authentication session.

### 5.2 A tagged interleaving

Let us now consider the tagged versions of the protocols. The previous attack is impossible, since $B$ is tagged by Verif in $(3.WMF)$ while is tagged by Claim in $(5.WL)$. As a direct consequence of Theorem 3.3, given the safety of $Protocol_{WL}$ and $Protocol_{WMF}$, the parallel composition (Table 11) of principals running the two

**Table 11** An example of multi-protocol system

$$\begin{array}{l}
\mathit{MultiProtocol} \triangleq \\
\quad \text{let } k_i = \mathsf{key}^{m}_{i=1}(I_i, I_0). \\
\quad (|^{m}_{\substack{i,j=1\\ i\neq j}} \; I_i \rhd !\mathit{Initiator}_{WMF}(k_i, I_i, I_j) | I_i \rhd !\mathit{Initiator}_{WL}(k_i, I_i, I_j) | \\
\qquad I_i \rhd !\mathit{Responder}_{WMF}(k_i, I_j, I_i) | I_i \rhd !\mathit{Responder}_{WL}(k_i, I_j, I_i) | \\
\qquad I_0 \rhd !\mathit{Server}_{WMF}(k_i, I_i, k_j, I_j) | I_0 \rhd !\mathit{Server}_{WL}(k_i, I_i, k_j, I_j))
\end{array}$$

protocols is safe. An interesting point is the possible interleaving of authentication sessions. For instance let us consider the following scenario:

$$\begin{array}{lll}
(1.WMF) & A \to B(T): & A \\
(2.WL) & B(T) \to A: & n_B \\
(3.WMF) & A \to B(T): & \{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}} \\
(4.WL) & B \to T: & A, \{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}} \\
(5.WL) & T \to B: & \{A : \mathsf{Id}, M, n_B : \mathsf{Claim}\}_{k_{BT}}
\end{array}$$

$A$ is running as initiator in the Wide Mouthed Frog Protocol, while $B$ as responder in the Woo and Lam. $B$ intercepts the message sent by $A$ to $T$ and exploits it for completing the authentication session. The interleaving arises since the second message of the two protocols has the same structure: an identity label, a payload and a nonce. Also the tags are the same. The corresponding trace is

$$\begin{array}{l}
A \rhd \mathit{out}(A) :: \\
B \rhd \mathit{in}(A) :: B \rhd \mathit{new}(n_B) :: \; B \rhd \mathit{out}(n_B) :: \\
A \rhd \mathit{in}(n_B) :: \textbf{run(A,B)} :: A \rhd \mathit{encrypt}\{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}} :: A \rhd \mathit{out}(\{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}}) :: \\
B \rhd \mathit{in}(\{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}}) :: B \rhd \mathit{out}(A, \{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}}) :: \\
T \rhd \mathit{in}(A, \{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}}) :: T \rhd \mathit{decrypt}\{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}} :: \\
T \rhd \mathit{encrypt}\{A : \mathsf{Id}, M, n_B : \mathsf{Claim}\}_{k_{BT}} :: T \rhd \mathit{out}(\{A : \mathsf{Id}, M, n_B : \mathsf{Claim}\}_{k_{BT}}) :: \\
B \rhd \mathit{in}(\{A : \mathsf{Id}, M, n_B : \mathsf{Claim}\}_{k_{BT}}) :: B \rhd \mathit{decrypt}\{A : \mathsf{Id}, M, n_B : \mathsf{Claim}\}_{k_{BT}} :: \textbf{commit(B,A)}
\end{array}$$

which is safe. Our tagging is fair general and can be applied uniformly over authentication protocols. The main result ensures that if every sequential component is locally correct, then all their possible interleavings in a multi-protocol tagged systems are safe. This result is justified by our tagging approach: it conveys and formalizes authentication informations and if two messages can be confused, then they provide the same authentication information (in the example case, $\{B : \mathsf{Id}, M, n_B : \mathsf{Verif}\}_{k_{AT}}$ conveys the following information: $A$ has started an authentication session with $B$).

## 6 Conclusion

**Table 12** Summary of the proof rules

**(Authenticate Claim)** After receiving $\{B : \mathsf{Id}, n : \mathsf{Claim} \ldots\}_k$ and checking the nonce $n$, $A$ authenticates $B$ as the claimant of the authentication session. See Sections 4.1,4.2 and 4.3.

**(Authenticate Owner)** After receiving $\{B : \mathsf{Id}, n : \mathsf{Owner}, y : \mathsf{Key} \ldots\}_{k_{AT}}$ and checking the nonce $n$, $A$ authenticates $y$ as a fresh session key owned by $B$. The reception of $\{D_1, \ldots, D_m\}_y$ authenticates $B$. See section 4.4.

**(Authenticate Verif)** After receiving $\{A : \mathsf{Id}, n : \mathsf{Verif} \ldots\}_{k_{AB}}$ and checking the nonce $n$, $A$ authenticates $B$, since the verifier requested by $B$ is $A$. See Section 4.1.

**(Claimant)** By encrypting $\{A : \mathsf{Id}, x : \mathsf{Claim}, \ldots\}_{k_{AB}}$, $A$ specifies herself as claimant and starts an authentication session with $B$. See Section 4.1.

**(Verifier)** By encrypting $\{B : \mathsf{Id}, x : \mathsf{Verif}, \ldots\}_k$, $A$ specifies $B$ as verifier and starts an authentication session with $B$. See Sections 4.1,4.2 and 4.3.

**(Owner)** By encrypting $\{D_1, \ldots, D_m\}_y$, $A$ confirms to have received the fresh session key $y$ in $\{B : \mathsf{Id}, n : \mathsf{Owner}, y : \mathsf{Key}, \ldots\}_k$ and starts an authentication session with $B$. See Section 4.4.

**(Ttp Forward & Check)** By encrypting $\{A : \mathsf{Id}, x : \mathsf{Claim}, \ldots\}_{k_{BT}}$, $T$ informs $B$ that $A$ is recently willing to authenticate herself with him. See Section 4.2

**(Ttp Forward)** By encrypting $\{A : \mathsf{Id}, x : \mathsf{Claim}, \ldots\}_{k_{BT}}$, $T$ informs $B$ that $A$ is willing to authenticate herself with him. Checking the freshness of the request is demanded to $B$. See Section 4.3.

**(Ttp Distribute)** By encrypting $\{A : \mathsf{Id}, x : \mathsf{Owner}, k_s : \mathsf{Key}, \ldots\}_{k_{BT}}$, $T$ distributes the fresh session key $k_s$ to $B$, declaring that it is shared with $A$. See Section 4.4.

In this paper we have applied our framework of [9,10] to the analysis of various authentication protocols taken from literature. The rules applied for verifying each protocol are summarized in Table 12. 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 principles we have considered here. However, the present principles appear general enough to validate many existing protocols and help understanding the underlying authentication mechanisms. Furthermore, the framework is scalable as new conditions may be added, if needed, to validate new protocols. We are currently extending our approach to deal with a wider range of authentication protocols, e.g., protocols based on public-key encryption and other kinds of nonce-challenges [18,19]. We are basing such an extension on a type and effect system. This should allow us to formally compare our approach with the type and effect systems proposed by Gordon and Jeffrey in [16,17].

**Acknoledgements** We thank the anonymous referees for their very helpful comments and suggestions.

## References

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pages 33–44, Portland, Oregon, January 2002. ACM Press.

[3] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theor. Comput. Sci.*, 298(3):387–415, 2003.

[4] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[5] R. Anderson and R. Needham. Programming satan's computer. In Jan van Leeuwen, editor, *Computer Science Today — Recent Trends and Developments*, volume 1000 of *LNCS*, pages 426–440, 1995.

[6] 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.

[7] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 01*, volume 2076, pages 667–681. LNCS 2076, Springer Verlag, 2001.

[8] C. Boyd. Hidden assumptions in cryptographic protocols. *IEEE Proceedings, Part E*, pages 433–436, November 1990.

[9] 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.

[10] M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of entity authentication protocols. In *Proceedings of European Symposium on Programming (ESOP 2004)*, March 2004.

[11] U. Carlsen. Optimal privacy and authentication on a portable communications system. *Operating Systems Review*, 28(3):16–23, July 1994.

[12] 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.

[13] D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.

[14] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[15] 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.

[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. *Information Technology-Security Tecniques-Entity Authentication Mechanisms, Part 2:Entity Authentication using Simmetric Tecniques*. 1993.

[22] G. 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.

[23] 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.

[24] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using mur$\phi$. 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. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[27] L. C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.

[28] M.D. Schroeder R.M. Needham. Using encryption for authentication in large networks of computers. *ACM Communication*, 21(12):993–999, 1978.

[29] Thomas Y. C. Woo and Simon S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[30] T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(3):39–51, 1992.