Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 389 (2007) 484–511
Theoretical Computer Science
www.elsevier.com/locate/tcs

A formal semantics for protocol narrations☆

Sébastien Briais[a], Uwe Nestmann[b,*]

[a] *School of Computer and Communication Sciences, EPFL, Switzerland*
[b] *EECS, Technical University of Berlin, Germany*

Abstract

Protocol narrations are a widely-used informal means to describe, in an idealistic manner, the functioning of cryptographic protocols as a single intended sequence of cryptographic message exchanges among the protocol's participants. Protocol narrations have also been informally "turned into" a number of formal protocol descriptions, e.g., using the spi-calculus. In this paper, we propose a direct formal operational semantics for protocol narrations that fixes a particular and, as we argue, well-motivated interpretation on how the involved protocol participants are supposed to execute. Based on this semantics, we explain and formally justify a natural and precise translation of narrations into spi-calculus. An optimised translation has been implemented in OCaml, and we report on case studies that we have carried out using the tool.

Keywords: Cryptographic protocols; Protocol narrations; Spi-calculus

0. Introduction

The setting. In the cryptographic protocol literature, protocols are usually expressed as *narrations* (see for example [14,25]). A protocol narration is a simple sequence of message exchanges between the different participating principals and can be interpreted as the intended trace of the ideal execution of the protocol. The protocol in Table 1 is a typical example of this style. Two principals A and B are both connected to the server S with whom they share the secret keys k_{AS} and k_{BS}, respectively. The protocol tells the story where A wants to establish a secret connection (a shared key k_{AB}) with B via the common server S: first, A should contact S, then S forwards the key k_{AB} to B. Finally, A uses this key to exchange secret data with B.

While much of the literature is concerned with stating and proving a security property of protocols like this one, we are more interested in the bare operational content of the description technique of narrations.

Our own motivation for the interest in a formal semantics for narrations is that we had implemented a "straightforward" translator [19] from protocol narrations into the spi-calculus, itself a pi-calculus extended with encryption primitives [3]. We then wanted to formally prove our translator correct but faced the problem that there was no formal intended semantics to compare to. This lacking semantics is what we provide within this paper. Indeed,

☆ A short version of this paper appeared in the Proceedings of *Trustworthy Global Computing 2005*, LNCS 3705, pages 163–181.

* Corresponding author.
E-mail addresses: Sebastien.Briais@epfl.ch (S. Briais), Uwe.Nestmann@TU-Berlin.DE (U. Nestmann).

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.09.005

Table 1
Wide-Mouthed Frog protocol

$A \rightsquigarrow S \ : \ (A\,.\,\{(B\,.\,k_{AB})\}_{k_{AS}})$
$S \rightsquigarrow B \ : \ \{(A\,.\,(B\,.\,k_{AB}))\}_{k_{BS}}$
$A \rightsquigarrow B \ : \ \{m\}_{k_{AB}}$

it turns out that the attempt to properly formalise narrations brings one already much closer to spi-like executable descriptions, but there are a number of insightful observations along the way, on which we report here as well.

The challenge. Despite being rather intuitive, the description technique of protocol narrations contains lots of implicit concepts. Looking for a formal semantics, these need to be rendered explicit. For example, Abadi [1] pointed out that "informal protocol narrations" need to be complemented with explanations of some either implicitly assumed facts or additional information to remove ambiguities. He raised four tasks that need to be pursued:

(1) One should make explicit what is known (public, private) before a protocol run, and what is to be generated freshly during a protocol run.
(2) One should make explicit which checks the individual principals are expected to carry out on the reception of messages.
(3) Principals act concurrently, in contrast to the apparently sequential idealised execution of a run according to a narration.
(4) Concurrency occurs also at the level of different protocol sessions, which may happen to be executed simultaneously while sharing the principals across.

(Interestingly, Abadi used these requirements to motivate the use of the spi-calculus as a description technique for "formal protocol narrations".)

The first item above should be clear: data is missing otherwise. To this aim, narrations usually come with a bit of explanation in natural language on the spirit of the protocol and on the assumptions made. Essentially, these assumptions consist of expliciting the *pieces of data* known in advance by the agents[1] and those that are to be *freshly generated* during the course of a protocol run.

The second item above results from the too high level of abstraction of message exchanges, noted as $A \rightsquigarrow B : M$. There are a number of problems connected to the fact that message M is usually transmitted from A to B by passing through an asynchronous insecure network where a potential intruder can interfere [18]. Thus, once B receives a message, it may be just the expected one according to the protocol, but it may also be an intended message received at the wrong moment and, worse, it may be an unintended message forged by some malicious attacker. So, B needs to perform some informative checks. But precisely which ones? For example, when B receives M it must first check in how far, at this very moment, it "understands" M (with respect to possible encryptions). Then, if B acquires new knowledge by this analysis, it must ensure that this new knowledge is consistent with its previously acquired knowledge. Some careful analysis is due, requiring a suitable representation of knowledge.

The third item above looks innocent at first, but once the non-atomic passage of messages through the network is properly taken into account, some surprising effects arise due to parts of *later* message exchanges (referring to the order of exchanges in a narration) possibly occurring before *earlier* message exchanges have completed or even started.

The fourth item above is again intuitively straightforward, but the description technique of narrations completely ignores the problem.

Our approach. In this paper, we present solutions to the first three items, leaving the fourth for future work (see Section 8). Concerning the first item, we simply add a declaration part to narrations (Section 1). Here, we are no different from competing approaches (see the paragraph on related work). On item two, we propose (in Section 2) to compile exchanges of the form $A \rightsquigarrow B : M$ into three separate syntactic parts, corresponding to:

(i) A *asynchronously* sends M towards B,
(ii) B receives some message (intended to be M), and
(iii) finally B checks that the message it just received indeed has the expected properties (associated with M, from the point of view of B).

[1] We use the terms principal and agent interchangeably.

Table 2
Protocol narrations

M, N	::=	$a \mid A \mid \{M\}_N \mid (M\,.\,N) \mid \mathsf{pub}(M) \mid \mathsf{priv}(M) \mid \mathsf{H}(M)$	(messages $\boldsymbol{M}$)
T	::=	$A \rightsquigarrow B : M$	(exchanges)
L	::=	$\epsilon \mid T\,;\,L$	(narrations)
D	::=	A **knows** $M \mid A$ **generates** $n \mid$ **private** k	(declarations)
P	::=	$D\,;\,P \mid L$	(protocol narrations $\boldsymbol{D}$)

With respect to the required checks, our approach is to automatically generate the maximum of checks derivable from the static information of protocol narrations. We call the resulting refined notion of narrations *executable*, because it will allow us to formalise an operational semantics of narrations, which would not be possible with an atomic, or synchronous, interpretation of message exchanges.

Concerning the third item, we profit from the above decomposition of message transmission and introduce a natural structural equivalence relation on executable narrations that may bring any of the (con)currently enabled actions to top level. On this basis, we provide a labelled transition semantics (Section 4).

Finally, we rewrite the executable narrations within the spi-calculus, which is then only a minor, albeit insightful, remaining step (Section 5). We then establish a straightforward formal operational correspondence between the two semantics.

Tool support. We have implemented the previous developments in OCaml (see Section 6). Due to the overly big size of the generated formulae, we studied possible simplification strategies. To this end, we have implemented naïve ideas such as removing duplicated atoms, or removing atoms like $[\,E : \boldsymbol{M}\,]$ when E is a message or when it appears as a subexpression of the remaining formula. We also perform some rewriting inside formulae, which according to our experience gives good results in practise.

Impact. Our paper targets at two different audiences.

To the cryptographic protocol audience, we offer a high-level bridge to the low-level (process calculus motivated) semantics of protocol narrations. However, it is our primary intention to accomplish this undertaking such that a reader does not need to be proficient in spi-calculus or its relatives. Thus, we propose – for an only slightly refined narration syntax – a formal semantics in which we cast in high-level narration terms the behaviour of a corresponding low-level spi-calculus semantics. Analysis techniques can now be built on top of this direct semantics.

To the process calculus audience, mainly as a by-product, we offer a gentle systematic way to comprehend and formally justify spi-calculus representations corresponding to protocol narrations. In particular, the uniform generation of "checks-on-reception" was lacking in earlier translations.

Related work and future work are deferred to the concluding section (Section 8).

1. Extending protocol narrations

Like in the competing approaches on the representation of protocol narrations, we extend the narrations with a header that declares the initial knowledge of each agent, the names generated by them and also the names that are assumed to be initially only known by the system (this last point permits to simulate a first pass where shared keys are securely distributed among some agents).

Hence, an extended protocol narration is composed of two parts: a sequence of *declarations* followed by the *narration* itself. The agents are picked among a countably infinite set $\boldsymbol{A}$ of *agent names* ranged over by $A, B, C, \ldots, S, \ldots$ and the messages are built upon a countably infinite set $\boldsymbol{N}$ of *names* ranged over by $a, b, c, \ldots, k, l, m, n, \ldots$. For the sake of simplicity, we assume that $\boldsymbol{A} \cap \boldsymbol{N} = \emptyset$.

We implicitly assume that all the agents involved in the protocol know each other; this can be generalised by explicit declarations. The syntax of messages and protocol narrations is given in Table 2.

Among the messages, any name a, agent name A, encryption $\{M\}_N$ (using N as encryption key) or pair $(M\,.\,N)$ can itself be used as a *shared key*. In contrast, a message of the form $\mathsf{pub}(M)$ or $\mathsf{priv}(M)$ can only be used as an *asymmetric key*. The inverse key M^{-1} of a message M is defined as below:

Table 3
Wide-Mouthed Frog protocol, with formal declarations

private k_{AS} ; A **knows** k_{AS} ; S **knows** k_{AS} ;
private k_{BS} ; B **knows** k_{BS} ; S **knows** k_{BS} ;
A **generates** k_{AB} ; A **knows** m
$A \rightsquigarrow S : (A\,.\,\{(B\,.\,k_{AB})\}_{k_{AS}})$;
$S \rightsquigarrow B : \{(A\,.\,(B\,.\,k_{AB}))\}_{k_{BS}}$
$A \rightsquigarrow B : \{m\}_{k_{AB}}$; ϵ

$$M^{-1} \stackrel{\text{def}}{=} \begin{cases} \mathsf{pub}(M') & \text{if } M = \mathsf{priv}(M') \\ \mathsf{priv}(M') & \text{if } M = \mathsf{pub}(M') \\ M & \text{otherwise.} \end{cases}$$

Here, we adopt the point of view that – under the assumption that some participant already "knows" two messages M' and M – this participant has the power to "verify" whether M' is in fact (equal to) the inverse M^{-1} of M. Evidently, participants are not capable to "compute" the private key $(\mathsf{pub}(M))^{-1}$.

The meaning of **private** k is that k is a name which is initially only available for the agents involved in the protocol. Typically, it is useful to simulate that an agent A and a server S initially share a secret key k_{AS}. The meaning of A **knows** M is simply that, initially, agent A knows the message M. Finally the meaning of A **generates** n is that A will generate a fresh name n (typically a nonce). For the sake of clarity, we enforce fresh generated names to be declared explicitly. Table 3 shows the Wide-Mouthed Frog protocol using our framework.

It often happens in cryptographic protocols that a secret is shared by several participants. For this reason, we propose to introduce as a macro the construct

$$A_1, \ldots, A_n \textbf{ share } k$$

which is intended to mean that the agents $A_1, \ldots, A_n$ share the secret name k. This macro is simply expanded into:

private k ; A_1 **knows** k ; ... ; A_n **knows** k.

To ease the writing of formal declarations, one can also imagine to introduce the shortcut $A_1, \ldots, A_n$ **knows** M to mean A_1 **knows** M ; ... ; A_n **knows** M.

2. Compiling protocol narrations

Target syntax. As motivated in the introduction, *executable narrations* (set $\boldsymbol{X}$, as defined in Table 4) are to be more explicit about the behaviour of individual agents. Instead of atomic exchanges of the form $A \rightsquigarrow B : M$ as used in the standard narrations of Table 2, we observe four more fine-grained basic actions (nonterminal I in Table 4): emission $A{:}B!E$ of a message expression E (evaluating to M, see below), reception $B{:}?x$ of a message and binding it to a variable x (see below), check $B{:}\phi$ for the validity of formula ϕ from the point of view of principal B, and scoping νk, which is reminiscent of the spi-calculus and represents the creation and scope of private names. Scoping is decoupled from principals, allowing us to use a single construct for names that are **private** and **generated** according to the declarations of Section 1.

In interacting systems, when an agent receives a message, it binds it to a fresh variable for reference in subsequent processing. For this purpose, we introduce a well-founded totally ordered countably infinite set $x, y, z, \ldots$ of *variables* $\boldsymbol{V}$ that we assume to be disjoint from $\boldsymbol{A} \cup \boldsymbol{N}$. An agent can operate in different ways on a message: (1) as with the previous standard narrations, it can concatenate two messages, encrypt one message with another (the key), compute the hash code of a message or take the public/private part of a message; (2) it can project a message onto its parts using $\pi_1(E)$ or $\pi_2(E)$ (if E "represents" a pair of two messages) or decrypt it using $\mathrm{D}_F(E)$ (if it knows the inverse key "represented by" F that was used to encrypt the message "represented by" E). Since an agent does not only handle messages but also variables, we introduce the notion of message *expressions* ($\boldsymbol{E}$), including the above further operations. The process of finding out whether some expression indeed "represents" some particular message, is formalised using the *evaluation function* in Table 5. The definitions are straightforward and offer no surprises.

Table 4
Syntax of executable narrations

E, F	::=	$a \mid A \mid \{E\}_F \mid (E\,.\,F) \mid \mathsf{pub}(\mathrm{E}) \mid \mathsf{priv}(\mathrm{E}) \mid \mathsf{H}(\mathrm{E})$	(expressions $\boldsymbol{E}$)
	$\mid$	$x \mid \mathrm{D}_F(E) \mid \pi_1(E) \mid \pi_2(E)$	
ϕ, ψ	::=	$tt \mid [E = F] \mid [E : \boldsymbol{M}] \mid \mathrm{inv}(E, F) \mid \phi \wedge \phi$	(formulae $\boldsymbol{F}$)
I	::=	$\nu k \mid A{:}B!E \mid A{:}?x \mid A{:}\phi$	(simple action)
X	::=	$\epsilon \mid I\,;\,X$	(executable narrations $\boldsymbol{X}$)

Table 5
Evaluation of expressions (can fail, in particular if $\mathrm{v}(E) \neq \emptyset$) and formulae

Definition of $[\![\cdot]\!] : \boldsymbol{E} \to \{\bot\} \cup \boldsymbol{M}$			
$[\![E]\!]$	$\stackrel{\text{def}}{=}$	E	if $E \in \boldsymbol{N} \cup \boldsymbol{A}$
$[\![(E\,.\,F)]\!]$	$\stackrel{\text{def}}{=}$	$(M\,.\,N)$	if $[\![E]\!] = M \in \boldsymbol{M}$ and $[\![F]\!] = N \in \boldsymbol{M}$
$[\![\pi_1(E)]\!]$	$\stackrel{\text{def}}{=}$	M	if $[\![E]\!] = (M\,.\,N) \in \boldsymbol{M}$
$[\![\pi_2(E)]\!]$	$\stackrel{\text{def}}{=}$	N	if $[\![E]\!] = (M\,.\,N) \in \boldsymbol{M}$
$[\![\{E\}_F]\!]$	$\stackrel{\text{def}}{=}$	$\{M\}_N$	if $[\![E]\!] = M \in \boldsymbol{M}$ and $[\![F]\!] = N \in \boldsymbol{M}$
$[\![\mathrm{D}_F(E)]\!]$	$\stackrel{\text{def}}{=}$	M	if $[\![E]\!] = \{M\}_N \in \boldsymbol{M}$ and $[\![F]\!] = N^{-1} \in \boldsymbol{M}$
$[\![\mathsf{H}(E)]\!]$	$\stackrel{\text{def}}{=}$	$\mathsf{H}(M)$	if $[\![E]\!] = M \in \boldsymbol{M}$
$[\![\mathsf{pub}(E)]\!]$	$\stackrel{\text{def}}{=}$	$\mathsf{pub}(M)$	if $[\![E]\!] = M \in \boldsymbol{M}$
$[\![\mathsf{priv}(E)]\!]$	$\stackrel{\text{def}}{=}$	$\mathsf{priv}(M)$	if $[\![E]\!] = M \in \boldsymbol{M}$
$[\![E]\!]$	$\stackrel{\text{def}}{=}$	$\bot$	in all other cases
Definition of $[\![\cdot]\!] : \boldsymbol{F} \to \{\mathbf{true}, \mathbf{false}\}$			
$[\![tt]\!]$	$\stackrel{\text{def}}{=}$	**true**	
$[\![\phi \wedge \psi]\!]$	$\stackrel{\text{def}}{=}$	$[\![\phi]\!]$ and $[\![\psi]\!]$	
$[\![[E = F]]\!]$	$\stackrel{\text{def}}{=}$	**true**	if $[\![E]\!] = [\![F]\!] = M \in \boldsymbol{M}$
$[\![[E : \boldsymbol{M}]]\!]$	$\stackrel{\text{def}}{=}$	**true**	if $[\![E]\!] = M \in \boldsymbol{M}$
$[\![\mathrm{inv}(E, F)]\!]$	$\stackrel{\text{def}}{=}$	**true**	if $[\![E]\!] = [\![F]\!]^{-1} = M \in \boldsymbol{M}$
$[\![\phi]\!]$	$\stackrel{\text{def}}{=}$	**false**	in all other cases

Formulae ϕ on received messages are described by (conjunctions of) three kinds of checks: *equality tests* $[E = F]$ on expressions denote the comparison of two bit-streams of E and F; *well-formedness tests* $[E : \boldsymbol{M}]$ denote the verification of whether the projections and decryptions contained in E are likely to succeed; *inversion tests* $\mathrm{inv}(E, F)$ denote the verification that E and F evaluate to inverse messages. The evaluation function of Table 5 is straightforwardly extended to formulae; note that, according to it, $[E : \boldsymbol{M}]$ is just a macro for $[E = E]$. Similarly, $\mathrm{inv}(E, F)$ can be encoded (for example) as $[\mathrm{D}_F(\{(E\,.\,F)\}_E) : \boldsymbol{M}]$ (see also Section 6).

Table 4 lists the syntax of expressions, formulae and executable narrations. In the following, we will omit the trailing; ϵ of a non-empty executable narration. Moreover, we overload the operator; to also concatenate narrations.

Definition 1. Let $M \in \boldsymbol{M}$, $E \in \boldsymbol{E}$, $\phi \in \boldsymbol{F}$, $x \in \boldsymbol{V}$. We let $\mathrm{n}(M)$, $\mathrm{n}(E)$, and $\mathrm{n}(\phi)$ denote the set of names occurring in M, E, and ϕ, respectively. Similarly, we let $\mathrm{v}(E)$ and $\mathrm{v}(\phi)$ denote the set of variables occurring in E and ϕ. $E\{^M/_x\}$ and $\phi\{^M/_x\}$ denote the substitution of M for x in E and ϕ, respectively.

Knowledge representation. As motivated in the introduction, the central point of the actual behaviour of protocols is to find out which checks are to be performed. We further motivated that such checks need to be based on (1) the narration code, which statically spells out the intended message to be received, and (2) the current knowledge at the moment of reception, which imposes constraints on how much the recipient can dynamically learn from the received message and on what other information the newly acquired knowledge must be consistent with.

Table 6
Synthesis

$$\text{SYN-PAIR}\ \frac{(M, E) \in \mathcal{S}(K) \qquad (N, F) \in \mathcal{S}(K)}{((M \,.\, N), (E \,.\, F)) \in \mathcal{S}(K)}$$

$$\text{SYN-ENC}\ \frac{(M, E) \in \mathcal{S}(K) \qquad (N, F) \in \mathcal{S}(K)}{(\{M\}_N, \{E\}_F) \in \mathcal{S}(K)} \qquad \text{SYN-HASH}\ \frac{(M, E) \in \mathcal{S}(K)}{(\mathsf{H}(M), \mathsf{H}(E)) \in \mathcal{S}(K)}$$

$$\text{SYN-PRIV}\ \frac{(M, E) \in \mathcal{S}(K)}{(\mathsf{priv}(M), \mathsf{priv}(E)) \in \mathcal{S}(K)} \qquad \text{SYN-PUB}\ \frac{(M, E) \in \mathcal{S}(K)}{(\mathsf{pub}(M), \mathsf{pub}(E)) \in \mathcal{S}(K)}$$

Table 7
Analysis

$$\text{ANA-INI}\ \frac{(M, E) \in K}{(M, E) \in \mathcal{A}_0(K)}$$

$$\text{ANA-FST}\ \frac{((M \,.\, N), E) \in \mathcal{A}_n(K)}{(M, \pi_1(E)) \in \mathcal{A}_{n+1}(K)} \qquad \text{ANA-SND}\ \frac{((M \,.\, N), E) \in \mathcal{A}_n(K)}{(N, \pi_2(E)) \in \mathcal{A}_{n+1}(K)}$$

$$\text{ANA-DEC}\ \frac{(\{M\}_N, E) \in \mathcal{A}_n(K) \qquad (N^{-1}, F) \in \mathcal{S}(\mathcal{A}_n(K))}{(M, \mathrm{D}_F(E)) \in \mathcal{A}_{n+1}(K)}$$

$$\text{ANA-DEC-REC}\ \frac{(\{M\}_N, E) \in \mathcal{A}_n(K) \qquad (N^{-1}, F) \notin \mathcal{S}(\mathcal{A}_n(K))}{(\{M\}_N, E) \in \mathcal{A}_{n+1}(K)}$$

$$\text{ANA-NAM-REC}\ \frac{(M, E) \in \mathcal{A}_n(K) \qquad M \in \boldsymbol{N} \cup \boldsymbol{A}}{(M, E) \in \mathcal{A}_{n+1}(K)}$$

$$\text{ANA-PUB}\ \frac{(\mathsf{pub}(M), E) \in \mathcal{A}_n(K)}{(\mathsf{pub}(M), E) \in \mathcal{A}_{n+1}(K)} \qquad \text{ANA-PRIV}\ \frac{(\mathsf{priv}(M), E) \in \mathcal{A}_n(K)}{(\mathsf{priv}(M), E) \in \mathcal{A}_{n+1}(K)}$$

$$\text{ANA-HASH}\ \frac{(\mathsf{H}(M), E) \in \mathcal{A}_n(K)}{(\mathsf{H}(M), E) \in \mathcal{A}_{n+1}(K)}$$

Instead of accumulating only the dynamically acquired messages (stored in variables x) we propose to tightly connect the (according to the narration) statically intended messages M with the dynamically received actual messages x. For this, we simply use pairs (M, x). Since consistency checks will then (have to) operate on such pairs, we need to generalise this representation of principal knowledge to finite subsets of $\boldsymbol{M} \times \boldsymbol{E}$. The underlying idea is that a pair (M, E) means that the expression E is supposed to be equal (or: has to evaluate) to M.

The following definition introduces knowledge sets, and also some traditionally employed operations on them: *synthesis* reflects the closure of knowledge sets using message constructors; *analysis* reflects the exhaustive recursive decomposition of knowledge pairs as enabled by the currently available knowledge.

Definition 2 (*Knowledge*). *Knowledge sets* $K \in \boldsymbol{K}$ are finite subsets of $\boldsymbol{M} \times \boldsymbol{E}$.

The set of names occurring in K is denoted by $\mathrm{n}(K)$.

The *synthesis* $\mathcal{S}(K)$ of K is the smallest subset of $\boldsymbol{M} \times \boldsymbol{E}$ containing K and satisfying the SYN-rules in Table 6.

The *analysis* $\mathcal{A}(K)$ of K is $\bigcup_{n\in\mathbb{N}} \mathcal{A}_n(K)$ where the sets $\mathcal{A}_i(K)$ are the smallest sets satisfying the ANA-rules in Table 7.

Our definition of analysis refines the usual approach reminiscent of Paulson [27]. Instead of directly defining a "flat" analysis set, we had to define a finitely stratified hierarchy $(\mathcal{A}_n(K))_{n\in\mathbb{N}}$. (See Appendix A for a detailed comparison of the two approaches.) Essentially, the index n of an analysis set $\mathcal{A}_n(K)$ approximates the number of proper deconstruction steps that were needed in order to derive its knowledge items (see the rules ANA-INI, ANA-FST,

ANA-SND, and ANA-DEC). In contrast to the standard approach, corresponding to $\mathcal{A}_n(K) \subseteq \mathcal{A}_{n+1}(K)$, here only certain items – not all of them – may be propagated from analysis level n to $n+1$ without proper deconstruction step.

As the following example shows, with the notion of knowledge of this paper the simple rule $\mathcal{A}_n(K) \subseteq \mathcal{A}_{n+1}(K)$ would allow us to possibly analyse the same message several times, in different ways, which would indeed be harmful. Assume that we remove the rules ANA-DEC-REC and ANA-NAM-REC as well as the indices of analysis sets in Table 7 (which amounts to admitting $\mathcal{A}_n(K) \subseteq \mathcal{A}_{n+1}(K)$). If we now analyse the knowledge set $K = \{(k, k), (\{k\}_k, x)\}$ according to this "standard" approach then we would first get the pair $(k, \mathrm{D}_k(x))$, then the pair $(k, \mathrm{D}_{\mathrm{D}_k(x)}(x))$, then $(k, \mathrm{D}_{\mathrm{D}_{\mathrm{D}_k(x)}(x)}(x))$, etc. The resulting analysis set $\mathcal{A}(K)$ would be of infinite size, and thus not even be a knowledge set[2], thus prohibiting a finite representation of the knowledge of participants.

Instead, we control the propagation from analysis level n to $n+1$ by the rules ANA-NAM-REC and ANA-DEC-REC. Knowledge items (M, E) can only be propagated to the next level of the analysis if M is not analysable (i.e., deconstructable) with the knowledge of the same level: either M is a pure name (possibly an agent name) or M can *not* be decrypted with knowledge from the same analysis level. Note that when computing the sequence $(\mathcal{A}_n(K))_{n\in\mathbb{N}}$, the rules ANA-FST, ANA-SND and ANA-DEC strictly decrease the size of the messages, so they can only be applied a finite number of times. Thus, it is obvious that the sequence $(\mathcal{A}_n(K))_{n\in\mathbb{N}}$ converges and thus $\mathcal{A}(K)$ is finite.

Example 3. Consider $K_0 = \{(A, A), (B, B), (S, S), (k_{AS}, k_{AS}), (k_{BS}, k_{BS})\}$.

Let $K = K_0 \cup \left\{((A\,.\,\{(B\,.\,k_{AB})\}_{k_{AS}}), x_0)\right\}$.

$$\text{We have } \mathcal{A}(K) = K \cup \begin{Bmatrix} (\quad A & , & \pi_1(x_0) &) \\ (\ \{(B\,.\,k_{AB})\}_{k_{AS}} & , & \pi_2(x_0) &) \\ (\quad (B\,.\,k_{AB}) & , & \mathrm{D}_{k_{AS}}(\pi_2(x_0)) &) \\ (\quad B & , & \pi_1(\mathrm{D}_{k_{AS}}(\pi_2(x_0))) &) \\ (\quad k_{AB} & , & \pi_2(\mathrm{D}_{k_{AS}}(\pi_2(x_0))) &) \end{Bmatrix}.$$

Generating checks. The above knowledge representation allows us to generate the checks required on message reception in a justified manner. Recall that these checks must verify (1) in how far the expectations of the recipient on the received message (as expressed statically in the narration) are matched according to the recipient's current knowledge, and (2) in how far the gained knowledge is consistent with its previously acquired knowledge.

Thus, obviously necessary checks are due to the *type* of messages: if an expression shall correspond to a pair then it better allows for projections; if an expression shall correspond to an encrypted message, then it better allows for decryption with the appropriate key – but only if it is known by the receiver.

Less obviously required checks result from the following observation: a message (identifier) M may occur more than once in a protocol narration. Thus, it may happen that, in some knowledge set, M is related to two different expressions E_1 and E_2, via (M, E_1) and (M, E_2). As M was precisely used in protocol narrations to indicate the *very same* message, such a knowledge set can only be considered consistent if E_1 and E_2 indeed evaluate to the same message. In the context of asymmetric keys, it can also happen that, in some knowledge set, we find a combination of (M_1, E_1) and (M_2, E_2) where $M_1 = {M_2}^{-1}$. In this case, also the corresponding $E_1 = {E_2}^{-1}$ should be satisfied.

Let us assume, as it is customary, that agents dispose of some meaningful initial knowledge (usually of the form (M, M) with M representing some initially known key or participant name). Then, the consistency check for repeated occurrences of data implicitly may take care of testing, e.g., whether some received datum was sent by the expected agent.

To formalise these requirements, we generate consistency formulae.

Definition 4 (*Consistency Formula*). Let K be a knowledge set. Its *consistency formula* $\Phi(K)$ is defined as follows:

$$\begin{aligned} \Phi(K) \stackrel{\text{def}}{=} \quad & \textstyle\bigwedge_{(M,E)\in K} \ [\,E : \boldsymbol{M}\,] \\ \wedge \quad & \textstyle\bigwedge_{(M,E_i)\in K \,\wedge\, (M,E_j)\in\mathcal{S}(K) \,\wedge\, E_i\neq E_j} \ [\,E_i = E_j\,] \\ \wedge \quad & \textstyle\bigwedge_{(M,E_i)\in K \,\wedge\, (M^{-1},E_j)\in\mathcal{S}(K)} \ \mathrm{inv}(E_i, E_j). \end{aligned}$$

[2] In contrast, the "standard" analysis of the corresponding (i.e., projected onto the static component) knowledge set $K_1 = \{k, \{k\}_k\}$ yields $\mathcal{A}(K_1) = \{k, \{k\}_k\}$.

The first conjunction clause checks that all the expressions can be evaluated. The second conjunction clause checks that if there are several ways to build a particular message M, then all the corresponding expressions evaluate to the same entity. (Note that the omission of the subclause $E_i \neq E_j$ would make the first clause redundant; we just kept it for clarity of the respective concepts.) The third conjunction clause checks that if it was possible to generate a message M and its inverse M^{-1}, then the corresponding expressions must also be mutually inverse. Note that this clause represents the principle of a rather paranoid lack of confidence by a protocol principal in its peers in that it includes all imaginable malicious situations. On the downside, it may sometimes create lots of tests that may not be informative or intuitive in all contexts (see Example 5). More refined, and less paranoid, principles to generate tests are of course possible and can be considered as variations of the above third clause.

When generating the above consistency formula, we compare pairs taken from K with pairs taken from $\mathcal{S}(K)$. The following example shows why it does not suffice to compare just the pairs in K. On the other hand, we should not compare any possible combination of pairs taken from $\mathcal{S}(K)$, because this would yield an infinite formula.

Example 5. If $K = \{(m, x), (\mathsf{H}(m), y)\}$, we have that

$$\Phi(K) = [x : \boldsymbol{M}] \wedge [y : \boldsymbol{M}] \wedge [\mathsf{H}(x) = y] \wedge \mathrm{inv}(x, x) \wedge \mathrm{inv}(y, y) \wedge \mathrm{inv}(\mathsf{H}(x), y).$$

Observe that, if the consistency formula did not consider the pairs taken from $\mathcal{S}(K)$, then the test $[\mathsf{H}(x) = y]$ would not be present.

Reducing knowledge sets. Knowledge sets can often be simplified without loss of information by reducing complex elements to their parts. In our case, we can further simplify due to the occurrence of duplicated elements; there is no loss of information once the consistency formula of Definition 4 remembers the duplication.

Definition 6 (*Irreducibles*). Let K be a knowledge set.

The set of *irreducibles* $\mathcal{I}(K)$ is defined by

$$\mathcal{I}(K) \stackrel{\text{def}}{=} \mathrm{irr}(\mathcal{A}(K)),$$

where

$$\begin{aligned}
\mathrm{irr}(K) = \quad & \{(M, E) \in K \mid M \in \boldsymbol{N} \cup \boldsymbol{A}\} \\
\cup \quad & \{(\{M\}_N, E) \in K \mid \neg(\exists F_1, F_2 : (M, F_1) \in \mathcal{S}(K) \wedge (N, F_2) \in \mathcal{S}(K))\} \\
\cup \quad & \{((M \,.\, N), E) \in K \mid \neg(\exists F_1, F_2 : (M, F_1) \in \mathcal{S}(K) \wedge (N, F_2) \in \mathcal{S}(K))\} \\
\cup \quad & \{(\mathsf{H}(M), E) \in K \mid \neg(\exists F : (M, F) \in \mathcal{S}(K))\} \\
\cup \quad & \{(\mathsf{priv}(M), E) \in K \mid \neg(\exists F : (M, F) \in \mathcal{S}(K))\} \\
\cup \quad & \{(\mathsf{pub}(M), E) \in K \mid \neg(\exists F : (M, F) \in \mathcal{S}(K))\}.
\end{aligned}$$

Let $\sim$ denote the equivalence relation on $\boldsymbol{M} \times \boldsymbol{E}$ induced by

$$(M, E) \sim (N, F) \iff M = N.$$

We let $\mathrm{rep}(K)$ denote the result of deterministically selecting[3] one representative element for each equivalence class induced by $\sim$ on K.

Example 7. We continue Example 3. We have:

$$\begin{aligned}
\mathcal{I}(K) &= K_0 \cup \left\{(A, \pi_1(x_0)), (B, \pi_1(\mathrm{D}_{k_{AS}}(\pi_2(x_0)))), (k_{AB}, \pi_2(\mathrm{D}_{k_{AS}}(\pi_2(x_0))))\right\} \\
\mathrm{rep}(\mathcal{I}(K)) &= K_0 \cup \left\{(k_{AB}, \pi_2(\mathrm{D}_{k_{AS}}(\pi_2(x_0))))\right\}.
\end{aligned}$$

Here, we assume that the function $\mathrm{rep}(\cdot)$ selected (A, A) instead of $(A, \pi_1(x_0))$, and (B, B) instead of $(B, \pi_1(\mathrm{D}_{k_{AS}}(\pi_2(x_0))))$. Moreover, we have that

$$(\{(A \,.\, (B \,.\, k_{AB}))\}_{k_{BS}}, \{(A \,.\, (B \,.\, \pi_2(\mathrm{D}_{k_{AS}}(\pi_2(x_0)))))\}_{k_{BS}}) \in \mathcal{S}(\mathrm{rep}(\mathcal{I}(K))).$$

[3] Choose an arbitrary well-founded total order on expressions and select the smallest expression according to this order.

The following lemma states a number of simple sanity properties.

Lemma 8. *Let K be a knowledge set. We have:*

(1) $\mathcal{S}(\text{irr}(K)) \subseteq \mathcal{S}(K)$
(2) $\mathcal{S}(\text{rep}(K)) \subseteq \mathcal{S}(K)$
(3) $\forall (M, E) \in \mathcal{S}(K) : \exists F : (M, F) \in \mathcal{S}(\text{irr}(K))$
(4) $\forall (M, E) \in \mathcal{S}(K) : \exists F : (M, F) \in \mathcal{S}(\text{rep}(K))$
(5) $\forall (M, E), (N, F) \in \mathcal{S}(\text{rep}(\text{irr}(K))) : M = N \Rightarrow E = F$.

Proof. (1) Because $\text{irr}(K) \subseteq K$ and $\mathcal{S}(\cdot)$ is monotonic.
(2) Because $\text{rep}(K) \subseteq K$ and $\mathcal{S}(\cdot)$ is monotonic.
(3) By induction on the structure of M.
(4) By induction on the derivation of $(M, E) \in \mathcal{S}(K)$.
(5) By induction on the structure of M. ■

The following proposition studies the relation between evaluation consistency formulae of a knowledge set K and some operations on the latter.

Proposition 9. *Let K be a closed knowledge set (where no variables appear) and $\phi = \Phi(K)$. If $[\![\phi]\!] = \textbf{true}$ then*

(1) $\forall (M, E) \in \mathcal{S}(K) : \forall (M, F) \in \mathcal{S}(\text{irr}(K)) : [\![E = F]\!] = \textbf{true}$
(2) $\forall (M, E) \in \mathcal{S}(K) : \forall (M, F) \in \mathcal{S}(\text{rep}(K)) : [\![E = F]\!] = \textbf{true}$
(3) $[\![\Phi(\text{irr}(K))]\!] = \textbf{true}$
(4) $[\![\Phi(\text{rep}(K))]\!] = \textbf{true}$.

Proof. (1) By induction on the structure of M.
(2) By induction on the derivations $(M, E) \in \mathcal{S}(K)$ and $(M, F) \in \mathcal{S}(K)$.
(3) Obvious, because $\text{irr}(K) \subseteq K$.
(4) Obvious, because $\text{rep}(K) \subseteq K$. ■

The compilation. We now have set up all the required ingredients to compile an extended protocol narration into an executable protocol narration. Technically, while traversing the syntax of a given narration, the translation function keeps a record of global information on the used variables and hidden names, as well as local (i.e., participant-dependent) information on their knowledge on generated names.

Definition 10 (*Compilation*). The translation $\mathcal{X}[\![\cdot]\!]^{(\upsilon,\varpi,\kappa,\nu)} : \boldsymbol{D} \to \boldsymbol{X}$ is defined inductively in Table 8, where $\upsilon \subset \boldsymbol{V}$ (current set of used variables), $\varpi \subset \boldsymbol{N}$ (current set of private names), $\kappa : \boldsymbol{A} \to \boldsymbol{K}$ (partial mapping from agents to their current knowledge), and $\nu : \boldsymbol{A} \to \boldsymbol{N}$ (partial mapping from agents to their current set of generated names).

Let $P \in \boldsymbol{D}$ be a protocol narration. Let $\boldsymbol{A}_P$ denote the set of agent names appearing in P. Then, $\mathcal{X}[\![P]\!]^{(\emptyset,\emptyset,\kappa_P,\emptyset)}$ denotes the *compilation of* P, where the *initial knowledge* κ_P is defined by $\kappa_P(A) := \{(B, B) \mid B \in \boldsymbol{A}_P\}$ for all $A \in \boldsymbol{A}_P$.

P is called *well-formed* iff its compilation is defined.

For simplicity, the compilation assumes that all agents initially know each other, as expressed in the initial knowledge set κ_P. Checks-on-reception are deduced from the individual knowledge set of a receiver. To avoid to perform the same checks again and again, the compilation keeps the knowledge sets of κ in reduced form, i.e., $\kappa(A) = \text{rep}(\mathcal{I}(\kappa(A)))$. To update $\text{f} \in \{\kappa, \upsilon\}$, we note $\text{f}[x \leftarrow y]$ with $\text{f}[x \leftarrow y](x) = y$ and $\text{f}[x \leftarrow y](z) = \text{f}(z)$ for $z \neq x$.

The compilation of **private** k and A **generates** n checks in both cases that the local (or generated) name is fresh, but differs with respect to the addition of the fresh name to the knowledge sets of agents: whereas A **generates** n increases the knowledge of A, the name k of **private** k is not added to any knowledge; this task is deferred to explicit A **knows** k clauses for the intended A.

Table 8
Definition of $\mathcal{X}[\![\cdot]\!]$

$$
\begin{array}{rcl}
\mathcal{X}[\![\epsilon]\!]^{(\upsilon,\varpi,\kappa,\nu)} & \stackrel{\text{def}}{=} & \epsilon \\
\mathcal{X}[\![A \textbf{ knows } M \ ; \ P]\!]^{(\upsilon,\varpi,\kappa,\nu)} & \stackrel{\text{def}}{=} & \mathcal{X}[\![P]\!]^{(\upsilon,\varpi,\kappa',\nu)} \qquad \text{if } \mathrm{n}(M) \cap \bigcup_{A\in\boldsymbol{A}} \nu(A) = \emptyset \\
& & \text{where} \quad K'_A \stackrel{\text{def}}{=} \kappa(A) \cup \{(M, M)\} \\
& & \text{and} \quad \kappa' \stackrel{\text{def}}{=} \kappa\left[A \leftarrow \mathrm{rep}(\mathcal{I}(K'_A))\right] \\
\mathcal{X}[\![\textbf{private } k \ ; \ P]\!]^{(\upsilon,\varpi,\kappa,\nu)} & \stackrel{\text{def}}{=} & \nu k \ ; \ \mathcal{X}[\![P]\!]^{(\upsilon,\varpi\cup\{k\},\kappa,\nu)} \\
& & \qquad \text{if } k \notin \varpi \cup \bigcup_{A\in\boldsymbol{A}} (\mathrm{n}(\kappa(A)) \cup \nu(A)) \\
\mathcal{X}[\![A \textbf{ generates } n \ ; \ P]\!]^{(\upsilon,\varpi,\kappa,\nu)} & \stackrel{\text{def}}{=} & \nu n \ ; \ \mathcal{X}[\![P]\!]^{(\upsilon,\varpi,\kappa',\nu')} \\
& & \qquad \text{if } n \notin \varpi \cup \bigcup_{A\in\boldsymbol{A}} (\mathrm{n}(\kappa(A)) \cup \nu(A)) \\
& & \text{where} \quad K'_A \stackrel{\text{def}}{=} \kappa(A) \cup \{(n, n)\} \\
& & \text{and} \quad \kappa' \stackrel{\text{def}}{=} \kappa\left[A \leftarrow \mathrm{rep}(\mathcal{I}(K'_A))\right] \\
& & \text{and} \quad \nu' \stackrel{\text{def}}{=} \nu\left[A \leftarrow \nu(A) \cup \{n\}\right] \\
\mathcal{X}[\![A \rightsquigarrow B : M \ ; \ P]\!]^{(\upsilon,\varpi,\kappa,\nu)} & \stackrel{\text{def}}{=} & A{:}B!E \ ; \ B{:}?x \ ; \ B{:}\phi \ ; \ \mathcal{X}[\![P]\!]^{(\upsilon\cup\{x\},\varpi,\kappa',\nu)} \\
& & \qquad \text{if } A \neq B \text{ and } (M, E) \in \mathcal{S}(\kappa(A)) \\
& & \text{where} \quad x \stackrel{\text{def}}{=} \min(\boldsymbol{V} \setminus \upsilon) \\
& & \text{and} \quad K'_B \stackrel{\text{def}}{=} \kappa(B) \cup \{(M, x)\} \\
& & \text{and} \quad \kappa' \stackrel{\text{def}}{=} \kappa\left[B \leftarrow \mathrm{rep}(\mathcal{I}(K'_B))\right] \\
& & \text{and} \quad \phi \stackrel{\text{def}}{=} \Phi(\mathcal{A}(K'_B))
\end{array}
$$

The compilation of $A \rightsquigarrow B : M$ checks that M can be synthesized by A, picks a new variable x and adds the pair (M, x) to the knowledge of B.[4] The consistency formula $\Phi(\mathcal{A}(K'_B))$ of the analysis of this updated knowledge K'_B defines the checks ϕ to be performed by B at runtime. Note that this must be done on the non-reduced version. In fact, it is precisely the consistency check that allows us then to continue with the knowledge in reduced form.

Finally, note that our concept of *well-formedness* of a protocol narration corresponds to the notions of *executability* in [12]).

Example 11. Let WMF be the Wide-Mouthed Frog protocol presented Table 3.

$$
\begin{array}{rcl}
\text{We have } \kappa_{\mathsf{WMF}} : \boldsymbol{A} & \to & \boldsymbol{K} \\
A & \mapsto & \{(A, A), (B, B), (S, S)\} \\
B & \mapsto & \{(A, A), (B, B), (S, S)\} \\
S & \mapsto & \{(A, A), (B, B), (S, S)\}
\end{array}
$$

WMF is well-formed and its compilation is

$$
\begin{array}{l}
\mathcal{X}[\![\mathsf{WMF}]\!]^{(\emptyset,\emptyset,\kappa_{\mathsf{WMF}},\emptyset)} = \\
\qquad \nu k_{AS} \ ; \ \nu k_{BS} \ ; \ \nu k_{AB} \ ; \\
\qquad A{:}S!(A \,.\, \{(B \,.\, k_{AB})\}_{k_{AS}}) \ ; \qquad\qquad S{:}?x_0 \ ; \quad S{:}\phi_0 \ ; \\
\qquad S{:}B!\{(A \,.\, (B \,.\, \pi_2(\mathrm{D}_{k_{AS}}(\pi_2(x_0)))))\}_{k_{BS}} \ ; \quad B{:}?x_1 \ ; \quad B{:}\phi_1 \ ; \\
\qquad A{:}B!\{m\}_{k_{AB}} \ ; \qquad\qquad B{:}?x_2 \ ; \quad B{:}\phi_2
\end{array}
$$

where ϕ_0, ϕ_1 and ϕ_2 are given below.

[4] Usually, narrations are defined such that the sender A is supposed to statically know the precise name B of the intended receiver. In a dynamic scenario, the compilation would need to check that B is synthesizable by A.

A **knows** m ; A **knows** k_A ; A **knows** $\mathsf{pub}(k_B)$
B **knows** m ; B **knows** k_B ; B **knows** $\mathsf{pub}(k_A)$;
A **generates** n_1 ;
B **generates** n_2 ;
$A \rightsquigarrow B : \{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1)))\}_{\mathsf{priv}(k_A)}$
$B \rightsquigarrow A : \{(\{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1)))\}_{\mathsf{priv}(k_A)} \,.\, \mathsf{H}(n_2))\}_{\mathsf{priv}(k_B)}$
$A \rightsquigarrow B : n_1$
$B \rightsquigarrow A : n_2$

Fig. 1. The Exchange Subprotocol of the ASW protocol (simplified).

$$\{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1)))\}_{\mathsf{priv}(k_A)}$$

$$\{(\{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1)))\}_{\mathsf{priv}(k_A)} \,.\, \mathsf{H}(n_2))\}_{\mathsf{priv}(k_B)}$$

$$n_1$$

$$n_2$$

Fig. 2. Contract form at the end of the exchange.

$$\begin{aligned}
\phi_0 \;\approx\; & [\, A = \pi_1(x_0)\,] \wedge [\, B = \pi_1(\mathrm{D}_{k_{AS}}(\pi_2(x_0)))\,] \\
& \wedge\; \mathrm{inv}(\pi_2(\mathrm{D}_{k_{AS}}(\pi_2(x_0))), \pi_2(\mathrm{D}_{k_{AS}}(\pi_2(x_0)))) \\
\phi_1 \;\approx\; & [\, A = \pi_1(\mathrm{D}_{k_{BS}}(x_1))\,] \wedge [\, B = \pi_1(\pi_2(\mathrm{D}_{k_{BS}}(x_1)))\,] \\
& \wedge\; \mathrm{inv}(\pi_2(\pi_2(\mathrm{D}_{k_{BS}}(x_1))), \pi_2(\pi_2(\mathrm{D}_{k_{BS}}(x_1)))) \\
\phi_2 \;\approx\; & \mathrm{inv}(\mathrm{D}_{\pi_2(\pi_2(\mathrm{D}_{k_{BS}}(x_1)))}(x_2), \mathrm{D}_{\pi_2(\pi_2(\mathrm{D}_{k_{BS}}(x_1)))}(x_2)).
\end{aligned}$$

We refer the reader to Section 6.2 where this protocol is studied with our tool.

3. A detailed example: The ASW protocol

3.1. The protocol

The ASW protocol is an optimistic fair-exchange protocol for contract signing, proposed by Asokan, Shoup and Waidner in [4]. Fig. 1 shows the slightly simplified version of the Exchange Subprotocol of ASW (that we will simply refer afterwards as the ASW protocol) that Caleiro, Viganò and Basin have used in [13] to illustrate that a *direct interpretation* of protocol narrations would be too naïve. A direct interpretation simply lists all the *external* actions each participant should commit, but does not explicit the internal checks and does not verify that these external actions are actually feasible. This protocol also shows that it is not sufficient to check received messages just once, immediately after their reception, because the receiving participant might only later on gain further knowledge that would enable it to analyse the structure of the just-received message more deeply.

The goal of the ASW protocol is to establish a valid contract between the two participants A and B. The protocol proceeds in two rounds.

First, the two participants send their respective so-called *public commitments* $\mathsf{H}(n_1)$ or $\mathsf{H}(n_2)$ with the contract text m they have agreed upon; n_1 and n_2 being nonces generated by the two participants and called their respective *secret commitments* to the contract. For this first round, the respective messages are digitally signed with the participants' private keys. As usual, the signature can be verified by using the corresponding public key.

Then, in the second round, the participants exchange their respective secret commitments so that they can check the public commitment they have received in round one by hashing this value.

At the end of this exchange, both participants have a valid contract of the form indicated in Fig. 2.

In this protocol, the participants should in some sense backtrack their analysis once they have received the message of the second round. Indeed, when B first receives $\mathsf{H}(n_1)$, it cannot check that this corresponds to the hashing of the nonce n_1 since n_1 is not yet part of B's knowledge. However, once B receives n_1 in the second round, it is able to

$$\nu n_1 \ ; \ \nu n_2 \ ; \tag{1}$$
$$A{:}B!E_1 \ ; \ B{:}?x_1 \ ; \ B{:}\phi_1 \ ; \tag{2}$$
$$B{:}A!E_2 \ ; \ A{:}?x_2 \ ; \ A{:}\phi_2 \ ; \tag{3}$$
$$A{:}B!E_3 \ ; \ B{:}?x_3 \ ; \ B{:}\phi_3 \ ; \tag{4}$$
$$B{:}A!E_4 \ ; \ A{:}?x_4 \ ; \ A{:}\phi_4 \tag{5}$$

Fig. 3. Executable narration compiled from ASW protocol.

check that the hashing of n_1 is effectively equal to the message that it has supposed to be $\mathsf{H}(n_1)$ in the first round; this check should occur before B sends its own public commitment to A.

3.2. Compilation of the ASW protocol

We now study this protocol in our setting. We first define some shortcuts:

$$M_1 \stackrel{\text{def}}{=} \{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1)))\}_{\mathsf{priv}(k_A)}$$
$$M_2 \stackrel{\text{def}}{=} \{(\{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1)))\}_{\mathsf{priv}(k_A)} \,.\, \mathsf{H}(n_2))\}_{\mathsf{priv}(k_B)}.$$

Computing the initial knowledge. The initial knowledge set of participants A and B is, by definition,

$$\kappa_0^A \stackrel{\text{def}}{=} \{(A, A), (B, B)\}$$
$$\kappa_0^B \stackrel{\text{def}}{=} \{(A, A), (B, B)\} \,.$$

Compilation of the declarations. When compiling the declarations, the compilation process checks that n_1 and n_2 are distinct names not used in the other pieces of information declared to be known at the beginning of the protocol. After computation, we obtain the following knowledge sets

$$k_1^A \stackrel{\text{def}}{=} \{(A, A), (B, B), (m, m), (n_1, n_1), (k_A, k_A), (\mathsf{pub}(k_B), \mathsf{pub}(k_B))\}$$
$$k_1^B \stackrel{\text{def}}{=} \{(A, A), (B, B), (m, m), (n_2, n_2), (k_B, k_B), (\mathsf{pub}(k_A), \mathsf{pub}(k_A))\} \,.$$

The line 1 of Fig. 3 corresponds to the compilation of the declarations.

First message. When compiling the first message exchange, the compilation process

(1) checks that A can synthesize message M_1 by looking for an expression E_1 such that $(M_1, E_1) \in \mathcal{S}(k_1^A)$. Here, the unique candidate is expression $E_1 \stackrel{\text{def}}{=} \{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1)))\}_{\mathsf{priv}(k_A)}$.
(2) takes a new variable x_1 to be bound to the message that participant B will receive; according to the statically defined information contained in the narration, this message is expected to be M_1.
(3) computes the consistency formula ϕ_1 of the analysis of the knowledge set resulting from the addition of (M_1, x_1) to k_1^B, and computes a reduced form of $k_1^B \cup \{(M_1, x_1)\}$.

Here we have

$$\begin{aligned}
&\mathcal{A}(k_1^B \cup \{(M_1, x_1)\}) \\
&\quad = k_1^B \\
&\quad \cup \{(M_1, x_1)\} \\
&\quad \cup \left\{(((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1))), \mathrm{D}_{\mathsf{pub}(k_A)}(x_1))\right\} \\
&\quad \cup \left\{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)), \pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\right\} \\
&\quad \cup \left\{((m \,.\, \mathsf{H}(n_1)), \pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\right\} \\
&\quad \cup \left\{(\mathsf{pub}(k_A), \pi_1(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\right\} \\
&\quad \cup \left\{(\mathsf{pub}(k_B), \pi_2(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\right\} \\
&\quad \cup \left\{(m, \pi_1(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\right\} \\
&\quad \cup \left\{(\mathsf{H}(n_1), \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\right\} \,.
\end{aligned}$$

After some simplifications (see 6.1), the consistency formula appears to be equivalent to

$$\begin{aligned}\phi_1 \stackrel{\text{def}}{=} & \operatorname{inv}(x_1, x_1)\\ & \wedge \operatorname{inv}(\pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))), \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\\ & \wedge [\mathsf{pub}(k_B) = \pi_2(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))]\\ & \wedge [\mathsf{pub}(k_A) = \pi_1(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))]\\ & \wedge [m = \pi_1(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))].\end{aligned}$$

And a possible candidate for $\mathrm{rep}(\mathcal{I}(k_1^B \cup (M_1, x_1)))$ is

$$\begin{aligned}k_2^B \stackrel{\text{def}}{=} & \; k_1^B\\ & \cup \{(M_1, x_1)\}\\ & \cup \left\{(\mathsf{H}(n_1), \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\right\}.\end{aligned}$$

Note that (M_1, x_1) is not removed because B has no way to digitally sign a message with the *private* key of A.

(4) generates line 2 of Fig. 3.

Second message. The compilation of the second message exchange is similar to the first message but with role of A and B swapped. So the compilation process

(1) checks that B can synthesize the message M_2 by looking for an expression E_2 such that $(M_2, E_2) \in \mathcal{S}(k_2^B)$.

The candidate is expression $E_2 \stackrel{\text{def}}{=} \{(x_1 \,.\, \mathsf{H}(n_2))\}_{\mathsf{priv}(k_B)}$.

(2) takes a new variable x_2 to be bound to the message that participant A will receive; according to the statically defined information contained in the narration, this message is expected to be M_2.

(3) computes the consistency formula ϕ_2 of the analysis of the knowledge set resulting of the addition of (M_2, x_2) to k_1^A and computes a reduced form of $k_1^A \cup \{(M_2, x_2)\}$.

Here we have

$$\begin{aligned}&\mathcal{A}(k_1^A \cup \{(M_2, x_2)\})\\ &\quad = k_1^A\\ &\quad \cup \{(M_2, x_2)\}\\ &\quad \cup \left\{((M_1 \,.\, \mathsf{H}(n_2)), \mathrm{D}_{\mathsf{pub}(k_B)}(x_2))\right\}\\ &\quad \cup \left\{(M_1, \pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))\right\}\\ &\quad \cup \left\{(\mathsf{H}(n_2), \pi_2(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))\right\}\\ &\quad \cup \left\{(((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(n_1))), \mathrm{D}_{\mathsf{pub}(k_A)}(\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2))))\right\}\\ &\quad \cup \left\{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)), \pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))))\right\}\\ &\quad \cup \left\{((m \,.\, \mathsf{H}(n_1)), \pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))))\right\}\\ &\quad \cup \left\{(\mathsf{pub}(k_A), \pi_1(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2))))))\right\}\\ &\quad \cup \left\{(\mathsf{pub}(k_B), \pi_2(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2))))))\right\}\\ &\quad \cup \left\{(m, \pi_1(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2))))))\right\}\\ &\quad \cup \left\{(\mathsf{H}(n_1), \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2))))))\right\}.\end{aligned}$$

After some simplifications, the consistency formula appears to be equivalent to

$$\begin{aligned}\phi_2 \stackrel{\text{def}}{=} & \operatorname{inv}(x_2, x_2)\\ & \wedge \operatorname{inv}(\pi_2(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)), \pi_2(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))\\ & \wedge [\pi_1(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)) = M_1].\end{aligned}$$

And a possible candidate for $\mathrm{rep}(\mathcal{I}(k_1^A \cup (M_2, x_2)))$ is

$$k_2^A \stackrel{\text{def}}{=} k_1^A \cup \{(M_2, x_2)\} \cup \left\{(\mathsf{H}(n_2), \pi_2(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))\right\}.$$

(4) generates line 3 of Fig. 3.

Third message. For the third message, the compilation process

(1) checks that n_1 is synthesizable by A. The expression n_1 is a candidate to build the message n_1 (the pair (n_1, n_1) has been added to the knowledge set of A during the compilation of the declarations).
(2) takes a new variable x_3 to be bound to the message that participant B will receive; according to the statically defined information contained in the narration, this message is expected to be n_1.
(3) computes the consistency formula ϕ_3 of the analysis of the knowledge set resulting of the addition of (n_1, x_3) to k_2^B and computes a reduced form of $k_2^B \cup \{(n_1, x_3)\}$
Here we have

$$\mathcal{A}(k_2^B \cup \{(n_1, x_3)\}) = k_2^B \cup \{(n_1, x_3)\}.$$

After some simplifications (taking into account that at this point of the executable narration, the formula ϕ_1 should have been satisfied), the consistency formula appears to be equivalent to

$$\phi_3 \stackrel{\text{def}}{=} \mathrm{inv}(x_3, x_3) \wedge [\mathsf{H}(x_3) = \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))].$$

And a possible candidate for $\mathrm{rep}(\mathcal{I}(k_2^B \cup (n_1, x_3)))$ is

$$k_3^B \stackrel{\text{def}}{=} \left(k_2^B \setminus \left\{(\mathsf{H}(n_1), \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\right\}\right) \cup \{(n_1, x_3)\}.$$

Note that the pair $(\mathsf{H}(n_1), \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))$ has been removed from the knowledge set of B because now B knows n_1 and thus can synthesize himself $\mathsf{H}(n_1)$.
(4) generates line 4 of Fig. 3.

Fourth message. Finally, for the fourth message, the compilation process

(1) checks that n_2 is synthesizable by B. The expression n_2 is a candidate to build the message n_2.
(2) takes a new variable x_4 to be bound to the message that participant a will receive; according to the statically defined information contained in the narration, this message is expected to be n_2.
(3) computes the consistency formula ϕ_4 of the analysis of the knowledge set resulting of the addition of (n_2, x_4) to k_2^A and computes a reduced form of $k_2^A \cup \{(n_2, x_4)\}$.
Here we have

$$\mathcal{A}(k_2^A \cup \{(n_2, x_4)\}) = k_2^A \cup \{(n_2, x_4)\}.$$

After some simplifications (taking into account that at this point of the executable narration, the formula ϕ_2 should have been satisfied), the consistency formula appears to be equivalent to

$$\phi_4 \stackrel{\text{def}}{=} \mathrm{inv}(x_4, x_4) \wedge [\mathsf{H}(x_4) = \pi_2(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2))].$$

And a possible candidate for $\mathrm{rep}(\mathcal{I}(k_2^A \cup (n_2, x_4)))$ is

$$k_3^A \stackrel{\text{def}}{=} (k_2^A \setminus \left\{(\mathsf{H}(n_2), \pi_2(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))\right\} \cup \{(n_2, x_4)\}.$$

Note that the pair $(\mathsf{H}(n_2), \pi_2(\mathrm{D}_{\mathsf{pub}(k_B)}(x_2)))$ has been removed from the knowledge set of A because now A knows n_2 and thus is able to synthesize himself $\mathsf{H}(n_2)$.

(4) generates line 5 of Fig. 3.

3.3. *The* ASW *protocol and the pattern-matching spi-calculus*

If we just look at participant B, the spi-calculus term we can derive from the executable narration of Fig. 3 is (see also Section 5)

$$
\begin{aligned}
&(\nu n_2)\ B(x_1).\\
&\qquad \mathrm{inv}(x_1, x_1) \wedge \mathrm{inv}(\pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))), \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1))))\\
&\qquad \wedge [\,\mathsf{pub}(k_A) = \pi_1(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\qquad \wedge [\,\mathsf{pub}(k_B) = \pi_2(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\qquad \wedge [\,m = \pi_1(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\quad \overline{A}\langle\{(x_1 \,.\, \mathsf{H}(n_2))\}_{\mathsf{priv}(k_B)}\rangle.\\
&\quad B(x_3).\\
&\qquad \mathrm{inv}(x_3, x_3)\\
&\qquad \wedge [\,\mathsf{H}(x_3) = \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\quad \overline{A}\langle n_2\rangle.\,\mathbf{0}\,.
\end{aligned}
$$

But it is not clear to us how such a process can be expressed in the pattern-matching spi-calculus in the spirit of what is defined in [22], even if we ignore the third clause of the consistency formula (Definition 4) and thus adopt a less paranoid approach.

Indeed, in this case, the spi-calculus process would be

$$
\begin{aligned}
&(\nu n_2)\ B(x_1).\\
&\qquad [\,\mathsf{pub}(k_A) = \pi_1(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\qquad \wedge [\,\mathsf{pub}(k_B) = \pi_2(\pi_1(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\qquad \wedge [\,m = \pi_1(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\quad \overline{A}\langle\{(x_1 \,.\, \mathsf{H}(n_2))\}_{\mathsf{priv}(k_B)}\rangle.\\
&\quad B(x_3).\\
&\qquad [\,\mathsf{H}(x_3) = \pi_2(\pi_2(\mathrm{D}_{\mathsf{pub}(k_A)}(x_1)))\,]\\
&\quad \overline{A}\langle n_2\rangle.\,\mathbf{0}\,.
\end{aligned}
$$

A possible term in the pattern-matching spi-calculus would be

$$
\begin{aligned}
&\mathit{new}\ n_2; \mathit{inp}\ B\ \{x \bullet \{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(x)))\}_{\mathsf{priv}(k_A)}\};\\
&\qquad \mathit{out}\ A\ \{(\{((\mathsf{pub}(k_A) \,.\, \mathsf{pub}(k_B)) \,.\, (m \,.\, \mathsf{H}(x_1)))\}_{\mathsf{priv}(k_A)} \,.\, \mathsf{H}(n_2))\}_{\mathsf{priv}(k_B)};\\
&\qquad \mathit{inp}\ B\ \{\bullet x\};\\
&\qquad \mathit{out}\ A\ n_2; \mathbf{0}\,.
\end{aligned}
$$

But unfortunately, the first pattern is not *implementable* in the sense of [22]. Indeed, being able to write $\{x \bullet \cdots \mathsf{H}(x) \cdots\}$ in a pattern position would intuitively mean that it is possible to inverse the supposed one-way function $\mathsf{H}(\cdot) : \boldsymbol{M} \to \boldsymbol{M}$ and thus get a value x from its hashing $\mathsf{H}(x)$.

4. Executing protocol narrations

In this section, we propose an operational semantics for narrations. It proceeds in a traditional syntax-directed manner by analysing the current top-level construct in order to see what to execute next. Since narrations contain some implicit concurrency among principals, we introduce a structural reordering relation to shuffle concurrently

Table 9
Substitution

$$
\begin{array}{rcl}
\epsilon\{^{M}/_{x}\}_{@A} & \stackrel{\text{def}}{=} & \epsilon \\
(A'{:}B!E\ ;\ X)\{^{M}/_{x}\}_{@A} & \stackrel{\text{def}}{=} & \begin{cases} A'{:}B!E\ ;\ X\{^{M}/_{x}\}_{@A} & \text{if } A' \neq A \\ A{:}B!E\{^{M}/_{x}\}\ ;\ X\{^{M}/_{x}\}_{@A} & \text{otherwise} \end{cases} \\
(A'{:}?y\ ;\ X)\{^{M}/_{x}\}_{@A} & \stackrel{\text{def}}{=} & \begin{cases} A'{:}?y\ ;\ X\{^{M}/_{x}\}_{@A} & \text{if } A' \neq A \\ A{:}?y\ ;\ X\{^{M}/_{x}\}_{@A} & \text{if } A = A' \text{ and } y \neq x \\ A{:}?x\ ;\ X & \text{otherwise} \end{cases} \\
(A'{:}\phi\ ;\ X)\{^{M}/_{x}\}_{@A} & \stackrel{\text{def}}{=} & \begin{cases} A'{:}\phi\ ;\ X\{^{M}/_{x}\}_{@A} & \text{if } A' \neq A \\ A{:}\phi\{^{M}/_{x}\}\ ;\ X\{^{M}/_{x}\}_{@A} & \text{otherwise} \end{cases} \\
(\nu n\ ;\ X)\{^{M}/_{x}\}_{@A} & \stackrel{\text{def}}{=} & \nu n\ ;\ X\{^{M}/_{x}\}_{@A}
\end{array}
$$

enabled actions to the top level. The actual execution of steps further needs to take care of the evaluation of messages to be sent, and also to prevent from name clashes that are possible due to the presence of binders.

Binders and α-conversion. Our language of executable narrations contains two sorts of binders: one for names and one for variables.

The first binder is introduced by the construction νn. If $X = \nu n\ ;\ X'$, then n is bound in X (i.e. the free occurrences of n in X' refers to this binder). As the identity of n is not important, we identify X with $\nu n'\ ;\ X'\{^{n'}/_{n}\}$ where n' is a name that is not free in X and $X'\{^{n'}/_{n}\}$ is X' where all the free occurrences of n has been replaced with n'. X and $\nu n'\ ;\ X'\{^{n'}/_{n}\}$ are called α-equivalent. In the following, we identify α-equivalent executable narrations. Now, for an executable narration X, we can define the usual *bound names* $\mathrm{bn}(X)$, *free names* $\mathrm{fn}(X)$ of X and, moreover, if $n, n' \in \boldsymbol{N}$, $X\{^{n'}/_{n}\}$, the substitution of n' for n in X.

The second binder is the one introduced by the construction $A{:}?x$. If $X = A{:}?x\ ;\ X'$, then x is bound in the actions of X' concerning A: indeed, if further in the executable narration, B refers to x, the x is not the same as the one used by A. Since variables will typically be substituted with messages, we do not need α-conversion on variables but we need to define a new kind of *local substitution*: if X is an executable narration, $x \in \boldsymbol{V}$, $M \in \boldsymbol{M}$ with $\mathrm{n}(M) \cap \mathrm{bn}(X) = \emptyset$ (which can be assured by choosing a suitable α-equivalent version of X), and $A \in \boldsymbol{A}$, we define in Table 9 the substitution $X\{^{M}/_{x}\}_{@A}$ of M for x in X on A.

Reordering. Protocol narrations are sequences of actions. However, the sequential character is not always causally motivated. Instead, the order of two consecutive actions carried out by *different* principals can always be swapped, because – after our split of message exchanges in the compilation process of Section 2 – they are *independent*. The same holds for the consecutive occurrence of an action and a scope, unless the scope's name occurs in the action. Formally, we manifest the swapping of independent actions in a structural congruence relation.

Definition 12. The *reordering* $\cong \subseteq \boldsymbol{X} \times \boldsymbol{X}$ is the least equivalence relation satisfying the rules given in Table 10, and closed under contexts of the form $X\ ;\ [\cdot]\ ;\ X'$.

We define $\cong_{\alpha}$ to be the union of $\cong$ and α-equivalence.

Given a particular message exchange $A \rightsquigarrow B\ :\ M$, it may possibly seem surprising at first that the reordering relation allows the respective reception action $B{:}?x$ to occur *before* its associated emission action $A{:}B!M$. Clearly, the received message cannot be the intended one. Such a behaviour must be dealt with carefully, e.g., by rejecting unintended messages, but its existence cannot be avoided; it is a matter of fact in concurrent systems that exchange messages asynchronously.

Labelled transitions. We define a straightforward labelled semantics of executable narrations, in style influenced by the spi-calculus, in Table 11.

Our semantics relates two executable narrations with a transition $\xrightarrow{A:\beta}$ where $A \in \boldsymbol{A}$ and β is either an input action $?M$ where $M \in \boldsymbol{M}$ or a bound output action $(\nu\tilde{n})\ B!M$ where $\tilde{n}$ is a (possibly empty) list of pairwise distinct names $n_1 \cdots n_k$ (that are bound in the remainder), $B \in \boldsymbol{A}$ and $M \in \boldsymbol{M}$. If $k = 0$ (i.e. $\tilde{n}$ is empty), we will simply write $B!M$. Note that there is no internal action in our formal semantics of narrations. We might also have introduced a rule

Table 10
Reordering

$$\cong\text{-S-S}\ \frac{A \neq C}{A{:}B!E\ ;\ C{:}D!F \cong C{:}D!F\ ;\ A{:}B!E} \qquad \cong\text{-S-C}\ \frac{A \neq C}{A{:}B!E\ ;\ C{:}\phi \cong C{:}\phi\ ;\ A{:}B!E}$$

$$\cong\text{-S-R}\ \frac{A \neq C}{A{:}B!E\ ;\ C{:}?x \cong C{:}?x\ ;\ A{:}B!E} \qquad \cong\text{-R-C}\ \frac{A \neq C}{A{:}?x\ ;\ C{:}\phi \cong C{:}\phi\ ;\ A{:}?x}$$

$$\cong\text{-R-R}\ \frac{A \neq C}{A{:}?x\ ;\ C{:}?y \cong C{:}?y\ ;\ A{:}?x} \qquad \cong\text{-C-C}\ \frac{A \neq C}{A{:}\phi\ ;\ C{:}\psi \cong C{:}\psi\ ;\ A{:}\phi}$$

$$\cong\text{-S-N}\ \frac{n \notin \mathrm{n}(E)}{A{:}B!E\ ;\ \nu n \cong \nu n\ ;\ A{:}B!E} \qquad \cong\text{-C-N}\ \frac{n \notin \mathrm{n}(\phi)}{A{:}\phi\ ;\ \nu n \cong \nu n\ ;\ A{:}\phi}$$

$$\cong\text{-R-N}\ \frac{}{A{:}?x\ ;\ \nu n \cong \nu n\ ;\ A{:}?x} \qquad \cong\text{-N-N}\ \frac{}{\nu n\ ;\ \nu m \cong \nu m\ ;\ \nu n}$$

Table 11
Labelled semantics of executable narrations

$$\textsc{Send}\ \frac{[\![E]\!] = M \in \boldsymbol{M}}{A{:}B!E\ ;\ X \xrightarrow{A:B!M} X} \qquad \textsc{Receive}\ \frac{}{A{:}?x\ ;\ X \xrightarrow{A:?M} X\{^{M}/_{x}\}_{@A}}\ M \in \boldsymbol{M}$$

$$\textsc{Check}\ \frac{X \xrightarrow{A:\beta} X'}{A{:}\phi\ ;\ X \xrightarrow{A:\beta} X'}\ [\![\phi]\!] = \mathbf{true}$$

$$\textsc{Open}\ \frac{X \xrightarrow{A:(\nu\tilde{n})\ B!M} X'}{\nu z\ ;\ X \xrightarrow{A:(\nu z\tilde{n})\ B!M} X'}\ z \in \mathrm{n}(M) \setminus \{\tilde{n}\}$$

$$\textsc{Rearrange}\ \frac{X \cong_{\alpha} X' \qquad X' \xrightarrow{A:\beta} X''}{X \xrightarrow{A:\beta} X''}$$

$$\textsc{Com}\ \frac{X \xrightarrow{A:(\nu\tilde{n})\ B!M} X' \qquad X' \xrightarrow{B:?M} X''}{X \xrightarrow{\tau} \nu\tilde{n}\ ;\ X''}$$

but we tend to insist on the fact that every communication necessarily passes through the network, while such a rule COM would allow to avoid this.

5. Rewriting protocol narrations ... into spi-calculus

The spi-calculus is a process calculus that was designed in order to study cryptographic protocols. In this section, we show that executable narrations closely correspond to terms in a quite restricted fragment of the spi-calculus.

Syntax. We use a finite spi-calculus without choice, generated as P by:

$$P \quad ::= \quad \mathbf{0} \mid E(x).P \mid \overline{E}\langle F\rangle.P \mid P \mid Q \mid (\nu n)\ P \mid \phi P.$$

It is a subcalculus of the spi-calculus that we used in [9]. Similar in spirit to the Applied pi-calculus [2], it offers the advantage of making the language of expressions and formulae orthogonal to the process calculus itself. Note also that since our rewriting process does not use replication nor choice, we have deliberately chosen not to include these primitives at all.

We use the same syntactic categories (names, agent names) as for narrations.

Table 12
Labelled semantics of spi-calculus

$$\textsc{Input}\ \frac{[\![E]\!] = A \in \boldsymbol{A} \qquad M \in \boldsymbol{M}}{E(x).P \xrightarrow{A\,M} P\{^{M}/_{x}\}} \qquad \textsc{Output}\ \frac{[\![E]\!] = A \in \boldsymbol{A} \qquad [\![F]\!] = M \in \boldsymbol{M}}{\overline{E}\langle F\rangle.P \xrightarrow{\overline{A}\,M} P}$$

$$\textsc{Open}\ \frac{P \xrightarrow{(\nu\tilde{n})\,\overline{A}\,M} P'}{(\nu z)\,P \xrightarrow{(\nu z\tilde{n})\,\overline{A}\,M} P'}\ z \in \mathrm{n}(M) \setminus \tilde{n} \qquad \textsc{Res}\ \frac{P \xrightarrow{\mu} P'}{(\nu n)\,P \xrightarrow{\mu} (\nu n)\,P'}\ n \notin \mathrm{fn}(\mu)$$

$$\textsc{Guard}\ \frac{P \xrightarrow{\mu} P'}{\phi P \xrightarrow{\mu} P'}\ [\![\phi]\!] = \mathbf{true} \qquad \textsc{Par}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\textsc{Struct}\ \frac{P \equiv P' \qquad P' \xrightarrow{\mu} P'' \qquad P'' \equiv Q''}{P \xrightarrow{\mu} Q''}$$

In process $E(x).P$, the variable x is bound in P and in the process $(\nu n)\,P$, the name n is bound in P. For a process P, we denote its set of free names $\mathrm{fn}(P)$, bound names $\mathrm{bn}(P)$, free variables $\mathrm{fv}(P)$ and bound variables $\mathrm{bv}(P)$.

Semantics. Table 12 shows a labelled semantics for the spi-calculus. It relies on the definition of structural congruence $\equiv$ defined as the least congruence satisfying:

- $\forall P, Q, R : (P \mid Q) \mid R \equiv P \mid (Q \mid R)$
- $\forall P, Q : P \mid Q \equiv Q \mid P$
- $\forall P : P \mid \mathbf{0} \equiv P$
- $\forall P, Q, n : (\nu n)\,P \mid Q \equiv (\nu n)\,(P \mid Q)$ if $n \notin \mathrm{fn}(Q)$
- $\forall P, Q : P \equiv Q$ if P and Q are α-equivalent.

According to the rules Input and Output, communication can only occur on agent names. This limitation of the "standard" semantics of the spi-calculus is just for convenience: our rewriting process will always only put agent names in place of channels. Also, since it is syntactically impossible to hide an agent name from the outside, we do not even consider internal communication. Transitions are thus of two kinds: either an input action $A\,M$ or a bound output action $(\nu\tilde{n})\,\overline{A}\,M$ where in both the cases $A \in \boldsymbol{A}$ and $M \in \boldsymbol{M}$, $\tilde{n}$ being a (possibly empty) list of pairwise distinct names that are binding occurrences in M.

Executable narrations in spi-calculus. As the reader might have noticed, the executable narrations as of Section 2 and the spi-calculus above are similar. Thus, we may now provide a straightforward translation of executable narrations into the spi-calculus and easily show that the semantics is preserved. The main idea is that the implicit concurrency structure of narrations as encoded with explicit agent names is projected out ($X\restriction_A$ of Definition 13) and explicitly represented using the parallel composition operator of the spi-calculus. Any intended sequential occurrence of actions, namely those actions that are associated with the same agent, is preserved by using the prefix operator of the spi-calculus. The private names are then simply put as a top-level restriction around the parallel composition.

Definition 13 (*Translation*). Let $X \in \boldsymbol{X}$ be an executable narration.

(1) $\mathcal{A}(X)$ (Table 13) defines the set of agents acting in X.
(2) $R(X)$ (Table 13) defines the set of fresh restricted names of X.
(3) $X\restriction_A$ (Table 13) defines the spi projection of X on $A \in \boldsymbol{A}$.
(4) The translation $\mathcal{T}[\![X]\!]$ of X into spi-calculus is defined by:

$$\mathcal{T}[\![X]\!] \stackrel{\text{def}}{=} (\nu n)_{\,n \in R(X)} \prod_{A \in \mathcal{A}(X)} X\restriction_A,$$

where $(\nu n)_{\,n \in I}$ and $\prod_{n \in I}$ denote n-ary restriction and composition.
(5) $\mathcal{T}[\![A{:}?M]\!] \stackrel{\text{def}}{=} A\,M$ and $\mathcal{T}[\![A{:}(\nu\tilde{n})\,B!M]\!] \stackrel{\text{def}}{=} (\nu\tilde{n})\,\overline{B}\,M$ map transition labels.

Table 13
Definition of $\mathcal{A}(\cdot)$, $R(\cdot)$, and $\cdot\restriction$.

$$
\begin{array}{rcl}
\mathcal{A}(\epsilon) & \stackrel{\text{def}}{=} & \emptyset \\
\mathcal{A}(A{:}B!E\ ;\ X) & \stackrel{\text{def}}{=} & \{A\} \cup \mathcal{A}(X) \\
\mathcal{A}(A{:}?x\ ;\ X) & \stackrel{\text{def}}{=} & \{A\} \cup \mathcal{A}(X) \\
\mathcal{A}(A{:}\phi\ ;\ X) & \stackrel{\text{def}}{=} & \{A\} \cup \mathcal{A}(X) \\
\mathcal{A}(\nu n\ ;\ X) & \stackrel{\text{def}}{=} & \mathcal{A}(X)
\end{array}
\qquad
\begin{array}{rcl}
R(\epsilon) & \stackrel{\text{def}}{=} & \emptyset \\
R(A{:}B!E\ ;\ X) & \stackrel{\text{def}}{=} & R(X) \\
R(A{:}?x\ ;\ X) & \stackrel{\text{def}}{=} & R(X) \\
R(A{:}\phi\ ;\ X) & \stackrel{\text{def}}{=} & R(X) \\
R(\nu n\ ;\ X) & \stackrel{\text{def}}{=} & \{n\} \cup R(X)
\end{array}
$$

$$
\begin{array}{rcl}
\epsilon\restriction_A & \stackrel{\text{def}}{=} & \mathbf{0} \\
(A'{:}B!E\ ;\ X)\restriction_A & \stackrel{\text{def}}{=} & \begin{cases} \overline{B}\langle E\rangle.X\restriction_A & \text{if } A' = A \\ X\restriction_A & \text{otherwise} \end{cases} \\
(A'{:}?x\ ;\ X)\restriction_A & \stackrel{\text{def}}{=} & \begin{cases} A(x).X\restriction_A & \text{if } A' = A \\ X\restriction_A & \text{otherwise} \end{cases} \\
(A'{:}\phi\ ;\ X)\restriction_A & \stackrel{\text{def}}{=} & \begin{cases} \phi X\restriction_A & \text{if } A' = A \\ X\restriction_A & \text{otherwise} \end{cases} \\
(\nu n\ ;\ X)\restriction_A & \stackrel{\text{def}}{=} & X\restriction_A
\end{array}
$$

Table 14
Correspondence of abstract and `spyer` syntax for expressions and formulae

Expressions

this paper $\rightsquigarrow$	`spyer`
a	`a`
x	`x`
A	`A`
$\{E\}_F$	`enc(E,F)`
$\mathrm{D}_F(E)$	`dec(E,F)`
$(E\,.\,F)$	`<E,F>`

this paper $\rightsquigarrow$	`spyer`
$\pi_1(E)$	`fst(E)`
$\pi_2(E)$	`snd(E)`
$\mathsf{pub}(E)$	`pub(E)`
$\mathsf{priv}(E)$	`priv(E)`
$\mathsf{H}(E)$	`hash(E)`

Formulae

this paper $\rightsquigarrow$	`spyer`
$[\,E{:}\boldsymbol{M}\,]$	`wff(E)`
$[\,E = F\,]$	`[E=F]`
$\mathrm{inv}(E, F)$	`inv(E,F)`
$F \wedge G$	`F /\G`

The following theorem concludes that the operational semantics of executable narrations and their spi-calculus translations precisely coincide up to $\equiv$.

Theorem 14. *Let $X \in \boldsymbol{X}$ be an executable narration.*

(1) *If $X \xrightarrow{A:\beta} X'$ then $\mathcal{T}[\![X]\!] \xrightarrow{\mathcal{T}[\![A:\beta]\!]} P'$ with $P' \equiv \mathcal{T}[\![X']\!]$.*

(2) *If $\mathcal{T}[\![X]\!] \xrightarrow{\mu} P'$ then there exists $A \in \boldsymbol{A}$, X' and β such that $X \xrightarrow{A:\beta} X'$, $P' \equiv \mathcal{T}[\![X']\!]$ and $\mu = \mathcal{T}[\![A{:}\beta]\!]$.*

Proof. By transition induction. Straightforward due to the tailor-made definition of the target calculus in the light of the translation mapping. ■

6. Spyer

`spyer` is a tool, developed in OCaml, that implements the previous formal developments. A source distribution of `spyer` can be found online [11]; an early version was developed by Gensoul [19].

`spyer` takes as an input file an extended protocol narration (using also the syntactic sugar described at the end of Section 1) and outputs an executable protocol narration and/or a network of spi-calculus processes. The latter can then be used as input for our bisimulation checker `sbc` that implements the symbolic bisimulation described in [9]. We have briefly summarised in Table 14 the correspondence between the abstract syntax of expressions and formulae used in this paper and the expressions and formulae used by `spyer`.

Before commenting some examples adapted from [14], we explain how consistency formulae, which can quickly become huge, may be simplified.

6.1. Simplifying formula

The various subformulae generated by the consistency formula of Definition 4 contain lots of redundant information.

Example 15. For example, if $K = \{(A, A), (B, B), ((A \,.\, B), x), (A, \pi_1(x)), (B, \pi_2(x))\}$, then

$$\begin{aligned}
\Phi(K) = \quad & [\,A : \boldsymbol{M}\,] \wedge [\,B : \boldsymbol{M}\,] \wedge [\,x : \boldsymbol{M}\,] \\
\wedge \quad & [\,\pi_1(x) : \boldsymbol{M}\,] \wedge [\,\pi_2(x) : \boldsymbol{M}\,] \\
\wedge \quad & [\,A = \pi_1(x)\,] \wedge [\,B = \pi_2(x)\,] \\
\wedge \quad & [\,x = (A \,.\, B)\,] \wedge [\,x = (\pi_1(x) \,.\, B)\,] \\
\wedge \quad & [\,x = (\pi_1(x) \,.\, \pi_2(x))\,] \wedge [\,x = (A \,.\, \pi_2(x))\,] \\
\wedge \quad & \mathrm{inv}(A, A) \wedge \mathrm{inv}(A, \pi_1(x)) \\
\wedge \quad & \mathrm{inv}(B, B) \wedge \mathrm{inv}(B, \pi_2(x)) \\
\wedge \quad & \mathrm{inv}(x, x) \wedge \mathrm{inv}(x, (A \,.\, B)) \wedge \mathrm{inv}(x, (A \,.\, \pi_2(x))) \\
\wedge \quad & \mathrm{inv}(x, (\pi_1(x) \,.\, \pi_2(x))) \wedge \mathrm{inv}(x, (\pi_1(x) \,.\, B)).
\end{aligned}$$

Actually, we should also add the symmetric tests since they are syntactically different and the Definition 4 ignores the symmetry of $[\,\cdot = \cdot\,]$ and $\mathrm{inv}(\cdot, \cdot)$.

To avoid this combinatorial explosion, we devise some mostly straightforward rules to simplify formulae. Before stating them, we define formula equivalence.

Definition 16 (*Formula Equivalence*). Two formulae ϕ and ψ are equivalent – written $\phi \approx \psi$ – if and only if for all (closing) substitutions $\sigma : \boldsymbol{V} \to \boldsymbol{M}$, we have $[\![\phi\sigma]\!] = [\![\psi\sigma]\!]$.

Since substitution corresponds to message reception, two formulae are thus equivalent if they evaluate in the same way in every execution.

In the following enumeration of equivalence laws, with $\phi_1 \wedge \phi_2 \approx \phi_2 \wedge \phi_1$ and $(\phi_1 \wedge \phi_2) \wedge \phi_3 \approx \phi_1 \wedge (\phi_2 \wedge \phi_3)$, we consider formulae up to commutativity and associativity of the conjunction operator.

The first set of laws states the symmetry and transitivity of the equality test.

- $[\,E = F\,] \wedge \phi \approx [\,F = E\,] \wedge \phi$
- $[\,E = F\,] \wedge [\,F = G\,] \wedge \phi \approx [\,E = F\,] \wedge [\,E = G\,] \wedge \phi$.

The second set of laws simplifies well-formedness tests or inversion tests.

- If $\phi = [\,E : \boldsymbol{M}\,] \wedge \phi'$ and E is an expression without deconstructors (i.e., that does not contain any occurrence of $\pi_1(\cdot)$, $\pi_2(\cdot)$ or $\mathrm{D}.(\cdot)$), then $\phi \approx \phi'$.
- If $\phi = [\,E : \boldsymbol{M}\,] \wedge \phi'$ and E appear as a subexpression of an expression appearing in ϕ', then $\phi \approx \phi'$.
- If $\phi = [\,\pi_1(E) : \boldsymbol{M}\,] \wedge \phi'$ and $\pi_1(E)$ or $\pi_2(E)$ appear as a subexpression of an expression appearing in ϕ', then $\phi \approx \phi'$.
- If $\phi = [\,\pi_2(E) : \boldsymbol{M}\,] \wedge \phi'$ and $\pi_1(E)$ or $\pi_2(E)$ appear as a subexpression of an expression appearing in ϕ', then $\phi \approx \phi'$.
- $\mathrm{inv}(M, M^{-1}) \wedge \phi \approx \phi$.

The third set of laws rewrites well-formedness tests or inversion tests in terms of equality tests.

- $\mathrm{inv}(E, F) \wedge \phi \approx [\,\mathrm{D}_F(\{G\}_E) : \boldsymbol{M}\,] \wedge \phi$ for all G that do not contain deconstructors or if it contains some, they are inside an exact occurrence of E or F. For example, $G = F$ or $G = E$ are valid choices for G.
- $[\,E : \boldsymbol{M}\,] \wedge \phi \approx [\,E = E\,] \wedge \phi$.

The following law states a substitutivity property of equality:

- $[E=F]\wedge\phi \approx [E=F]\wedge\phi'$, for all ϕ' which is ϕ where some occurrences of E has been replaced by F or conversely.

Finally, the last set of laws rewrites equality tests such that the resulting expressions contain fewer constructors.

- $[(E_1 . E_2)=(F_1 . F_2)]\wedge\phi \approx [E_1=F_1]\wedge[E_2=F_2]\wedge\phi$
- $[\{E_1\}_{E_2}=\{F_1\}_{F_2}]\wedge\phi \approx [E_1=F_1]\wedge[E_2=F_2]\wedge\phi$
- $[\mathsf{H}(E)=\mathsf{H}(F)]\wedge\phi \approx [E=F]\wedge\phi$
- $[\mathsf{pub}(E)=\mathsf{pub}(F)]\wedge\phi \approx [E=F]\wedge\phi$
- $[\mathsf{priv}(E)=\mathsf{priv}(F)]\wedge\phi \approx [E=F]\wedge\phi$
- $[(E_1 . E_2)=F]\wedge\phi \approx [E_1=\pi_1(F)]\wedge[E_2=\pi_2(F)]\wedge\phi$.

Example 17. With the above laws, the formula of Example 15 is provably equivalent to:

$$\psi = [A=\pi_1(x)]\wedge[B=\pi_2(x)].$$

These ideas are implemented in `spyer`. Moreover, it exploits the fact that the consistency formula is only used when adding a pair (M, x) to an already reduced knowledge set K. So, to avoid that the same checks are performed several times, it keeps in a formula only the atoms involving the variable x.

6.2. The Wide-Mouthed Frog protocol

First, we give the input file corresponding to the Wide-Mouthed Frog protocol that we have studied earlier in this article.

```
(* Wide Mouthed Frog protocol *)

(* initial knowledge *)
A,B,S know A B S
A,S share kAS
B,S share kBS
A generates kAB
A know m

(* protocol narration *)
A -> S: <A,enc(<B,kAB>,kAS)>
S -> B: enc(<A,B,kAB>,kBS)
A -> B: enc(m,kAB)
```

We then invoke `spyer` with the above file to obtain the following executable narration:

```
new kAS
new kBS
A: new kAB
A: S!<A,enc(<B,kAB>,kAS)>
S: ?0
S: inv(snd(dec(snd(0),kAS)),snd(dec(snd(0),kAS)))
   [B = fst(dec(snd(0),kAS))]
   [A = fst(0)]
S: B!enc(<A,<B,snd(dec(snd(0),kAS))>>,kBS)
B: ?1
B: inv(snd(snd(dec(1,kBS))),
       snd(snd(dec(1,kBS))))
   [B = fst(snd(dec(1,kBS)))]
   [A = fst(dec(1,kBS))]
```

```
A: B!enc(m,kAB)
B: ?2
B: inv(dec(2,snd(snd(dec(1,kBS)))),
       dec(2,snd(snd(dec(1,kBS)))))
```

Finally, this gives the following spi-calculus system:

```
agent A(agent_A, agent_B, agent_S, kAS, m) =
  (^kAB)
    'agent_S<<agent_A, enc(<agent_B, kAB>, kAS)>>.
      'agent_B<enc(m, kAB)>.0)

agent B(agent_A, agent_B, kBS) =
  agent_B(x_1).
    {[agent_B = fst(snd(dec(x_1, kBS)))]
  /\ [agent_A = fst(dec(x_1, kBS))]
  /\ wff(dec(enc(kBS, snd(snd(dec(x_1, kBS)))),
             snd(snd(dec(x_1, kBS)))))}
    agent_B(x_2).
      {wff(dec(enc(kBS,
                   dec(x_2, snd(snd(dec(x_1, kBS))))),
              dec(x_2, snd(snd(dec(x_1, kBS))))))}0

agent S(agent_A, agent_B, agent_S, kAS, kBS) =
  agent_S(x_0).
    {[agent_B = fst(dec(snd(x_0), kAS))]
  /\ [agent_A = fst(x_0)]
  /\ wff(dec(enc(kAS, snd(dec(snd(x_0), kAS))),
             snd(dec(snd(x_0), kAS))))}
    'agent_B<enc(<agent_A, <agent_B,
                           snd(dec(snd(x_0), kAS))>>,
                kBS)>.0

agent System(agent_A, agent_B, agent_S, m) =
  (^kAS, kBS)
    (A(agent_A, agent_B, agent_S, kAS, m)
   | B(agent_A, agent_B, kBS)
   | S(agent_A, agent_B, agent_S, kAS, kBS))
```

6.3. The Otway–Rees protocol

In the Otway–Rees protocol, lots of redundant information is shared by participants. For example, the message m appears in every single message exchange. The extended narration corresponding to the Otway–Rees protocol is:

```
    (* Otway Rees protocol *)

    (* initial knowledge *)
    A,B,S know A B S
    A,S share kAS
    B,S share kBS
    A know m
    A generates nA
    B generates nB
```

```
    S generates kAB

    (* protocol narration *)
    A -> B : <m,A,B,enc(<nA,m,A,B>,kAS)>
    B -> S : <m,A,B,enc(<nA,m,A,B>,kAS),enc(<nB,m,A,B>,kBS)>
    S -> B : <m,enc(<nA,kAB>,kAS),enc(<nB,kAB>,kBS)>
    B -> A : <m,enc(<nA,kAB>,kAS)>
```

The executable narration computed by `spyer` is then:

```
new kAS
new kBS
A: new nA
B: new nB
S: new kAB
A: B!<m,<A,<B,enc(<nA,<m,<A,B>>>,kAS)>>>
B: ?0
B: inv(snd(snd(snd(0))),snd(snd(snd(0))))
   inv(fst(0),fst(0))
   [B = fst(snd(snd(0)))]
   [A = fst(snd(0))]
B: S!<fst(0),<A,<B,<snd(snd(snd(0))),
                    enc(<nB,<fst(0),<A,B>>>,kBS)>>>>
S: ?1
S: inv(fst(dec(snd(snd(snd(snd(1)))),kBS)),
       fst(dec(snd(snd(snd(snd(1)))),kBS)))
   inv(fst(dec(fst(snd(snd(snd(1)))),kAS)),
       fst(dec(fst(snd(snd(snd(1)))),kAS)))
   inv(fst(1),fst(1))
   [fst(1) = fst(snd(dec(snd(snd(snd(snd(1)))),kBS)))]
   [fst(1) = fst(snd(dec(fst(snd(snd(snd(1)))),kAS)))]
   [<A,B> = snd(snd(dec(snd(snd(snd(snd(1)))),kBS)))]
   [<A,B> = snd(snd(dec(fst(snd(snd(snd(1)))),kAS))))]
   [B = fst(snd(snd(1)))]
   [A = fst(snd(1))]
S: B!<fst(1),
       <enc(<fst(dec(fst(snd(snd(snd(1)))),kAS)),kAB>,kAS),
        enc(<fst(dec(snd(snd(snd(snd(1)))),kBS)),kAB>,kBS)>>
B: ?2
B: inv(snd(dec(snd(snd(2)),kBS)),snd(dec(snd(snd(2)),kBS)))
   inv(fst(snd(2)),fst(snd(2)))
   [fst(2) = fst(0)]
   [nB = fst(dec(snd(snd(2)),kBS))]
B: A!<fst(2),fst(snd(2))>
A: ?3
A: inv(snd(dec(snd(3),kAS)),snd(dec(snd(3),kAS)))
   [nA = fst(dec(snd(3),kAS))]
   [m = fst(3)]
```

Actually, if the computation of the consistency formulae was following literally what we devised in Section 2, the generated formulae of S would contain more than 300,000 (!) equality tests.

7. Related work

We roughly and somewhat artificially divide the list of competing approaches into two classes. One class tries to tightly associate some formal semantics with (variants of) narrations themselves. Another class provides less tight associations, usually involving a different target formalism: here, we may distinguish approaches that informally reformulate narrations within a different formalism from those that offer mostly automated translations from narrations into target formalisms, but do not clearly justify the underlying translation principles (if exposed at all). Let us, for the purpose of structuring this section, use the terms "tight" and "lax" semantics to separate the two classes.

Sumii et al. [28] propose a formal semantics of narrations by translation into spi-calculus. The paper is written in Japanese, so it remains unclear to us how "tight" the approach really is, how they treat the problem of checks-on-reception, and also whether there is any formal or informal justification of the translation principles. In any case, our own intention was to provide a formal semantics that does *not require* the use of an underlying (and possibly too) general process calculus, so our approach is still substantially different.

Tight semantics

The work of Caleiro, Viganó and Basin [12,13] is quite similar in spirit and aim with our work. They defined a trace-based denotational semantics and gave a corresponding operational semantics with a variant of pattern-matching spi-calculus as target language. Some underlying ideas are quite similar to ours but we find our formalism of knowledge sets is more light-weight (although equally powerful) than their theory of view/opacity. The *view* that a principal has of a message M corresponds to how far it understands the message M with its current knowledge. A message is said to be *opaque* for a participant if the latter is not able to analyse at all the form of the message (this corresponds to a view equal to a special symbol γ_M). To relate to these definitions, one might say that our approach consists in considering that initially a received message M is opaque and is thus bound to a fresh variable x_M. Then, the analysis of the receiver's knowledge set resulting from the addition of (M, x_M) to its current knowledge set corresponds to computing the view that the receiver then has of this particular message. In addition, the receiver also updates the "view" that it has of other previously received message. Then, we can directly use the result of the analysis to say which checks have to occur after the reception of M. In contrast, Caleiro, Viganó and Basin had to introduce and refer to further concepts like the *facial pattern*, the *constructive form* and the *inner facial pattern* (and relate them with the concept of view) before being able to give an operational semantics. The main simplification in our setting arises from the joint treatment of messages and associated "views" as knowledge elements of the form (M, E).

Another way to give semantics to protocol narrations could exploit the widely-developed machinery of *strand spaces* [30], proposed by Thayer Fábrega, Herzog and Guttman. This formalism has proved to be a successful framework for reasoning on and verifying security properties of cryptographic protocols [21]. Strand spaces are graphs that represent the intended protocol behaviour of narrations by an explicit use of arrow notations: one type of arrow captures the sequential dependencies within individual participants, giving rise to *strands*; another type of arrow captures the flow of messages between strands. In contrast to mere narrations, strand spaces are not limited to represent just the intended behaviour, but also the behaviour of malicious attackers, represented as the so-called penetrator strands. Since strand spaces come with a formal semantics, in terms of the so-called *bundles*, these immediately also provide some semantics for narrations. A bundle can be understood as a causality-closed subgraph of a given strand space. As such, it represents the possible result of a valid executions of the strand space. However, there is no notion of *dynamic* execution that could be understood as a form of operational semantics. Thus, there is also no study of dynamic checks-on-reception, which is the main technical contribution of the current paper.

On the other hand, strand spaces have also been studied by Crazzolara and Winskel in comparison to other models of concurrency [15,16], notably including event structures Petri nets and the algebraic process language SPL; the latter is a simplified (since channel-free) spi-calculus that is enhanced with some form of pattern-matching (cf. also [22] for pattern-matching in a more standard spi-calculus, and our comments below). Since these models of concurrency – in particular the language SPL – are equipped with forms of operational semantics, one might think that their relation to strand spaces could provide an operational semantics of the latter "for free". This, however, is not the case. In [15], for any given SPL-process P in some particularly restricted form called *!-par* process, Crazzolara and Winskel show how to formally and closely relate the net behaviour $\mathrm{Net}(P)$ to the strand space behaviour $\mathrm{Tr}(P)$. In contrast, they do not offer any way to translate strand spaces back into SPL-terms, which would be required to inherit the desired operational semantics. In [16], the authors further refine the relation between SPL and strand spaces by extending the

latter with a notion of conflict to allow for better composition properties. Still, they offer no way to translate arbitrary strand spaces to SPL processes.

Lax semantics

The work of Bodei et al. [7,8] is also similar to ours, although still quite different. Like us, they present a refinement of protocol narrations, but the respective checks-on-reception appear only informally. Like us, they split message exchanges into three parts, albeit different to ours. A formal semantics is then only provided after "rewriting", again informally, refined narrations into terms of their channel-free process calculus LySA. To our knowledge, the above papers (short and long version) provide the best available information about the system underlying their "systematic expansion". Still, the expansion is not unambiguously explained, while our expansion is fully automatic, based on simple intuitive principles and generates a maximum number of checks according to these principles. Finally, their approach aims at static analysis techniques, while we ultimately target at dynamic analysis, e.g., as in the form of bisimulation checks [9] in the full spi-calculus.

In other related approaches, narrations are reformulated or translated using Casper [24], HLPSL2IF [5], CAPSL [26], CASRUL [23], or (s)pi-calculus [3,6]. They have in common that they do not easily help to understand how the gap between the rather informal narrations and the target formalism is bridged. A compiler can itself be interpreted as giving semantics to narrations, but usually the translation process is not well-explained or otherwise justified, in particular regarding the treatment of checks-on-reception. Moreover, our interest was to try to formalise the semantics at the level of narrations rather than by translation into some reasonably unrelated target formalism.

A subtle, but interesting difference between our work and Casper [24] is their modified message syntax using a construction `M % v`, meaning that the recipient of `M` should *not* try to decrypt `M`. We think this construct was added because of Casper's rather strict policy to *require*, unless the % is used, to be able to fully decrypt all messages (and possibly provide a warning in case this fails). Our (arguably more flexible) policy is instead to require agents to always *just try* to decrypt messages as far as their current knowledge permits, so we implicitly let agents accept messages even if they cannot (yet) fully decrypt them.

As we previously observed (cf. Section 3.3), the pattern-matching spi-calculus of [22] is not expressive enough to capture the checks-on-reception we require. To overcome its limitations (in the part of our work that deals with rewriting into spi-calculus), we could have used a variant similar in spirit to the one of [13]. However, we found it more orthogonal and extensible to express those checks by means of dedicated formulae. Moreover, our use of this version of the spi-calculus was driven by the wish to have our tool `spyer` generate spi-calculus code that is compatible with our symbolic bisimulation checker based on [9]. Finally, note that the aim of [22] was to offer a type system to study safety property of processes, but not to enable an automated way to turn protocol narrations into spi-calculus processes.

8. Conclusions

Contributions. In summary, we stepwise enhance protocol narrations in order to build up enough structure such that a well-motivated formalisation of their operational semantics becomes possible. The main technical contribution is the proposal of the automatic generation of "checks-on-reception", together with a suitable representation of the principals' knowledge on which the generation depends.

If one wants to reformulate informal protocol narrations within a calculus like the spi-calculus, then we propose the following method:

(1) Extend the narration, as shown in Section 1, by a declaration part making precise the origin of and initial knowledge about the involved data (names). This step requires human interaction, because ambiguities need to be resolved.
(2) Compile the resulting narration, as shown in Section 2, into an executable narration. This step can now be done automatically.
(3) Extract the implicit concurrency, as shown just above. Again, automatically.

It is worthwhile pointing out that our approach does not bother the protocol designer to come up with suitable or sufficient checks-on-reception, because they are generated automatically. Our approach does not even require the designer to actually look at these generated checks at all.

Future work. Here, we do not tackle the fourth task listed by Abadi [1] on how to get to a formalisation of concurrent sessions on the basis of protocol narrations. The main problem is that *principals* may play different *roles* in concurrent sessions such that the lookup of their respective keys needs to be dealt with dynamically. The usual convenient confusion of the two concepts of principal and role is no longer appropriate, so we propose to nontrivially extend the narration notation rather than providing a suboptimal semantics to an inappropriate notation. Note that this confusion also rules out the naïve modelling of concurrent sessions by the bare unbounded replication within spi-calculus. Some inspiration from the work of Cremers and Mauw [17] and the work done in the context of mixed strand spaces [29, 20] may help us here.

Furthermore, it should be possible to develop reasoning techniques for protocol narrations via an *environment-sensitive* extension of our semantics that could be used to define and study meaningful behavioural equivalences.

Acknowledgment

The first author was supported by the Swiss National Science Foundation, grant No. 21-65180.1.

Appendix. A note about synthesis, flat analysis, closure set and stratified analysis

In this appendix, we compare our "stratified" approach to compute analyses to the usual "flat" approach. We do so by means of projection, focusing on the message components of our knowledge sets. This comparison is mainly intended to help readers who are familiar with Paulson's inductive approach (see [27]) to better understand our formalism. In particular, we show that the stratified analysis of a message set yields a complete knowledge seed.

Definition 18. If $S \subseteq \boldsymbol{M}$, we define

(1) The *synthesis* $synth(S)$ of S is the smallest set satisfying

$$\text{SYN-INC}\ \frac{M \in S}{M \in synth(S)} \qquad \text{SYN-PAIR}\ \frac{M \in synth(S) \qquad N \in synth(S)}{(M\,.\,N) \in synth(S)}$$

$$\text{SYN-ENC}\ \frac{M \in synth(S) \qquad N \in synth(S)}{\{M\}_N \in synth(S)}$$

$$\text{SYN-PUB}\ \frac{M \in synth(S)}{\mathsf{pub}(M) \in synth(S)} \qquad \text{SYN-PRIV}\ \frac{M \in synth(S)}{\mathsf{priv}(M) \in synth(S)}$$

$$\text{SYN-HASH}\ \frac{M \in synth(S)}{\mathsf{H}(M) \in synth(S)}$$

(2) the *flat analysis* $analyz(S)$ of S is the smallest set satisfying

$$\text{ANA-INC}\ \frac{M \in S}{M \in analyz(S)} \qquad \text{ANA-FST}\ \frac{(M\,.\,N) \in analyz(S)}{M \in analyz(S)}$$

$$\text{ANA-SND}\ \frac{(M\,.\,N) \in analyz(S)}{N \in analyz(S)}$$

$$\text{ANA-DEC}\ \frac{\{M\}_N \in analyz(S) \qquad N^{-1} \in synth(S)}{M \in analyz(S)}$$

(3) the *closure set* $close(S)$ of S is the smallest superset of S closed under both the flat analysis and the synthesis.

(4) the stratified analysis $\mathcal{A}(S)$ of S is $\bigcup_{i=0}^{+\infty} \mathcal{A}_i(S)$ where

- $\mathcal{A}_0(S) = S$

- $\mathcal{A}_{i+1}(S) = analyz(\mathcal{A}_i(S))$.

Definition 19. Let $S \subseteq \boldsymbol{M}$.

- S is *closed under synthesis* if and only if $synth(S) \subseteq S$
- S is *closed under flat analysis* if and only if $analyz(S) \subseteq S$.

It is well-known that $synth(analyz(S)) \subsetneq close(S)$.
For example, if $S = \left\{(k\,.\,\{m\}_{\mathsf{priv}((k\,.\,k))})\right\}$ then $m \notin synth(analyz(S))$ whereas $m \in close(S)$.
The remainder of this section is devoted to show that $synth(\mathcal{A}(S)) = close(S)$.

$synth(\mathcal{A}(S)) \subseteq close(S)$. To show this inclusion, we need the following lemma:

Lemma 20. *Let $S, T \subseteq \boldsymbol{M}$ and assume that $S \subseteq T$. Then*

- $synth(S) \subseteq synth(T)$
- $analyz(S) \subseteq analyz(T)$.

Proof. See [10]. ■

Proof (*of the First Inclusion*). We first show that for all $i \in \mathbb{N}$, $\mathcal{A}_i(S) \subseteq close(S)$.

Indeed, for $i = 0$, we have $\mathcal{A}_0(S) = S$ which is a subset of $close(S)$ by definition.

Then, if we assume by induction that $\mathcal{A}_i(S) \subseteq close(S)$, then by Lemma 20, we have $analyz(\mathcal{A}_i(S)) \subseteq analyz(close(S))$, i.e. $\mathcal{A}_{i+1}(S) \subseteq analyz(close(S))$. But since $analyz(close(S)) \subseteq close(S)$, we have thus $\mathcal{A}_{i+1}(S) \subseteq close(S)$.

Hence, we have $\mathcal{A}(S) \subseteq close(S)$.

Thus, still by Lemma 20, we have $synth(\mathcal{A}(S)) \subseteq synth(close(S))$ and since $synth(close(S)) \subseteq close(S)$, we conclude that $synth(\mathcal{A}(S)) \subseteq close(S)$. ■

$close(S) \subseteq synth(\mathcal{A}(S))$. To show this inclusion, we need the following lemmae:

Lemma 21. *Let $S \subseteq \boldsymbol{M}$. Then $synth(synth(S)) = synth(S)$.*

Proof. See [10]. ■

Lemma 22. *Let $S \subseteq \boldsymbol{M}$. Then $\mathcal{A}(S)$ is closed under flat analysis, i.e.*

$$analyz(\mathcal{A}(S)) \subseteq \mathcal{A}(S).$$

Proof. See [10]. ■

Lemma 23. *Let $S \subseteq \boldsymbol{M}$. Then if S is closed under flat analysis then so is $synth(S)$, i.e.*

$$analyz(S) \subseteq S \implies analyz(synth(S)) \subseteq synth(S).$$

Proof. See [10]. ■

Proof (*of the Second Inclusion*). We can now show that $synth(\mathcal{A}(S))$ is a superset of S closed both under flat analysis and synthesis.

First, we have $S = \mathcal{A}_0(S) \subseteq \mathcal{A}(S)$, so $S \subseteq synth(\mathcal{A}(S))$ and thus $synth(\mathcal{A}(S))$ is a superset of S.

By Lemma 22, we have $analyz(\mathcal{A}(S)) \subseteq \mathcal{A}(S)$. So by Lemma 23, we have $analyz(synth(\mathcal{A}(S))) \subseteq synth(\mathcal{A}(S))$. Hence $synth(\mathcal{A}(S))$ is closed under flat analysis.

By Lemma 21, we have $synth(synth(\mathcal{A}(S))) = synth(\mathcal{A}(S))$ so, in particular, $synth(synth(\mathcal{A}(S))) \subseteq synth(\mathcal{A}(S))$. Hence $synth(\mathcal{A}(S))$ is closed under synthesis.

Since $close(S)$ is the *smallest* superset of S which is closed both under synthesis and flat analysis, we have $close(S) \subseteq synth(\mathcal{A}(S))$. ■

Conclusion. We have finally shown that $synth(\mathcal{A}(S)) = close(S)$.

References

[1] M. Abadi, Security protocols and their properties, in: Foundations of Secure Computation, in: NATO ASI, IOS Press, 2000, pp. 39–60.
[2] M. Abadi, C. Fournet, Mobile values, new names, and secure communication, in: Proceedings of POPL '01, ACM, 2001, pp. 104–115.
[3] M. Abadi, A.D. Gordon, A calculus for cryptographic protocols: The spi calculus, Information and Computation 148 (1) (1999) 1–70.
[4] N. Asokan, V. Shoup, M. Waidner, Asynchronous protocols for optimistic fair exchange, in: Proceedings of the IEEE Symposium on Research in Security and Privacy, 1998, pp. 86–99.
[5] D. Basin, S. Mödersheim, L. Viganò, An on-the-fly model-checker for security protocol analysis, in: Proceedings of ESORICS'03, in: LNCS, vol. 2808, Springer-Verlag, Heidelberg, 2003, pp. 253–270.
[6] B. Blanchet, Automatic verification of cryptographic protocols: A logic programming approach, in: Proceedings of Principles and Practice of Declarative Programming, PPDP'03, ACM, 2003.
[7] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, H. Nielson, Automatic validation of protocol narration, in: Proceedings of 16th IEEE Computer Security Foundations Workshop, CSFW 16, 2003, pp. 126–140.
[8] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, H. Nielson, Static validation of security protocols, Journal of Computer Security 13 (2005) 347–390.
[9] J. Borgström, S. Briais, U. Nestmann, Symbolic bisimulation in the spi calculus, in: Proceedings of CONCUR 2004, in: LNCS, vol. 3170, Springer, 2004, pp. 161–176.
[10] S. Briais, Formal proofs about hedges using the Coq proof assistant. http://lamp.epfl.ch/~sbriais/spi/hedges/hedge.html, 2004.
[11] S. Briais, `spyer`. http://lamp.epfl.ch/spyer, 2006.
[12] C. Caleiro, L. Viganò, D. Basin, Deconstructing alice and bob, in: Proceedings of the ICALP 2005 Workshop on Automated Reasoning for Security Protocol Analysis, ARSPA'05, in: ENTCS, vol. 135.1, 2005, pp. 3–22.
[13] C. Caleiro, L. Viganò, D. Basin, On the semantics of alice and bob specifications of security protocols, Theoretical Computer Science 367 (1) (2006) 88–122.
[14] J.A. Clark, J.L. Jacob, A survey of authentication protocol literature, Technical Report 1.0, University of York, 1997.
[15] F. Crazzolara, G. Winskel, Events in security protocols, in: ACM Conference on Computer and Communications Security, 2001, pp. 96–105.
[16] F. Crazzolara, G. Winskel, Composing strand spaces, in: M. Agrawal, A. Seth (Eds.), FSTTCS '02, in: LNCS, vol. 2556, Springer, 2002, pp. 97–108.
[17] C. Cremers, S. Mauw, Operational semantics of security protocols, in: Scenarios: Models, Algorithms and Tools (Dagstuhl 03371 Post-Seminar Proceedings), in: LNCS, vol. 3466, 2005.
[18] D. Dolev, A.C. Yao, On the security of public key protocols, IEEE Transactions on Information Theory IT-29 (12) (1983) 198–208.
[19] C. Gensoul, Spyer — un compilateur de protocoles cryptographiques, Semester Project Report, EPFL, July 2003.
[20] J.D. Guttman, F.J. Thayer Fábrega, Protocol independence through disjoint encryption, in: Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW'00, 2000, pp. 24–34.
[21] J.D. Guttman, F.J. Thayer Fábrega, Authentication tests and the structure of bundles, Theoretical Computer Science 283 (2) (2002) 333–380.
[22] C. Haack, A.S.A. Jeffrey, Pattern-matching spi-calculus, Information and Computation 204 (8) (2006) 1195–1263.
[23] F. Jacquemard, M. Rusinowitch, L. Vigneron, Compiling and verifying security protocols, in: Logic for Programming and Automated Reasoning, in: LNCS, vol. 1955, Springer-Verlag, 2000, pp. 131–160.
[24] G. Lowe, Casper: A compiler for the analysis of security protocols, Journal of Computer Security 6 (1998) 53–84.
[25] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.
[26] J.K. Millen, CAPSL: Common authentication protocol specification language. http://www.csl.sri.com/users/millen/capsl/.
[27] L.C. Paulson, The inductive approach to verifying cryptographic protocols, Journal of Computer Security 6 (1–2) (1998) 85–128.
[28] E. Sumii, H. Tatsuzawa, A. Yonezawa, Translating security protocols from informal notation into spi calculus, IPSJ Transactions on Programming 45 (SIG 12) (1–10) (2004) written in Japanese, abstract in English.
[29] F.J. Thayer Fábrega, J.C. Herzog, J.D. Guttman, Mixed strand spaces, in: Proceedings of the 12th IEEE Computer Security Foundations Workshop, CSFW 1999, 1999, pp. 72–82.
[30] F.J. Thayer Fábrega, J.C. Herzog, J.D. Guttman, Strand spaces: Proving security protocols correct, Journal of Computer Security 7 (1) (1999) 191–230.