A Formal Semantics for Protocol Narrations

Sébastien Briais* and Uwe Nestmann

School of Computer and Communication Sciences, EPFL, Switzerland

Abstract. Protocol narrations are an 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.

0 Introduction

The setting. In the cryptographic protocol literature, protocols are usually expressed as *narrations* (see for example [CJ97,MvOV96]). 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. The two principals A and B are both connected to the server S who shares secret keys k_{AS} and k_{BS} with each of them. The protocol tells the story (narration) where A wants to establish a secret connection (a shared key) with B via the common server S: first, A should contact S, S generates the fresh key intended for A and B and passes it on to A; then, A contacts B directly, at the same time delivering the fresh key. The name t is used as a time-stamp required to prevent from replay attacks; earlier version of the protocol were flawed in this respect.

$$
\begin{array}{l}
A \rightsquigarrow S : (A \,.\, B) \\
S \rightsquigarrow A : \{((B \,.\, (k_{AB} \,.\, t)) \,.\, \{(A \,.\, (k_{AB} \,.\, t))\}_{k_{BS}})\}_{k_{AS}} \\
A \rightsquigarrow B : \{(A \,.\, (k_{AB} \,.\, t))\}_{k_{BS}}
\end{array}
$$

Table 1. Denning-Sacco protocol

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 [Gen03] from protocol narrations into the spi-calculus, which is a pi-calculus extended with encryption

* Supported by the Swiss National Science Foundation, grant No. 21-65180.1

primitives [AG99]. Then, we were looking for a way to formally prove our translator correct and had the problem that there was no formal intended semantics to compare to. This lacking semantics is what we provide within this paper. Indeed, it turns out that the attempt to properly formalize 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 [Aba00] 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 idealized 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 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 [DY83]. Thus, once B receives some 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 even 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

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

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 5). Concerning the first item, we simply add a declaration part to narrations (§1). Here, we are no different from competing approaches (see the paragraph on Related work). On item two, we propose to compile exchanges of the form $A{\rightsquigarrow}B{:}M$ into three separate syntactic parts (§2):

(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).

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 formalize an operational semantics of narrations, which was not 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 allows us to bring any of the (con-)currently enabled actions to top-level. On this basis, we provide a labeled transition semantics (§3).

Finally, we rewrite executable narrations within the spi-calculus, which is then only a minor, albeit insightful, remaining step (§4). We then establish a straightforward formal operational correspondence between the two semantics.

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 behavior 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. In this sense, our formal description can also be seen as an abstract formulation of our compiler [Gen03].

Related work and future work are deferred to the concluding section (§5).

$$
\begin{array}{rll}
M, N ::= a \mid A \mid \{M\}_N \mid (M\,.\,N) & & \text{(messages } \boldsymbol{M}\text{)} \\
T ::= A{\rightsquigarrow}B{:}M & & \text{(exchanges)} \\
L ::= \epsilon \mid T; L & & \text{(narrations)} \\
D ::= A \textbf{ knows } M \mid A \textbf{ generates } n \mid \textbf{private } k & & \text{(declarations)} \\
P ::= D; P \mid L & & \text{(protocol narrations } \boldsymbol{D}\text{)}
\end{array}
$$

Table 2. Protocol narrations

1 Extending protocol narrations

Like in the competing approaches on the representation of protocol narrations, we extend 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 for example a first pass where shared keys have been 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 sake of simplicity, we assume that $\boldsymbol{A} \cap \boldsymbol{N} = \emptyset$.

For simplicity, we consider here only the possibility to concatenate messages (denoted by $(M\,.\,N)$ for M and N) or to encrypt them under a shared-key cryptosystem ($\{M\}_N$ is the encryption of message M with shared-key N). It is straightforward to extend the following formal development to a richer message language (using ideas of [Bri04,BBN04]). We implicitly assume that all agents involved in the protocol know each other; this can be generalized by explicit declarations. The syntax of messages and protocol narrations is given in Table 2.

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 piece of data 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 Denning-Sacco protocol using our framework.

$$
\begin{array}{l}
\textbf{private } k_{AS}; \textbf{private } k_{BS}; \\
A \textbf{ knows } k_{AS}; A \textbf{ knows } t; B \textbf{ knows } k_{BS}; B \textbf{ knows } t; \\
S \textbf{ knows } k_{AS}; S \textbf{ knows } k_{BS}; S \textbf{ knows } t; S \textbf{ generates } k_{AB}; \\
A{\rightsquigarrow}S{:}(A\,.\,B); \\
S{\rightsquigarrow}A{:}\{((B\,.\,(k_{AB}\,.\,t))\,.\,\{(A\,.\,(k_{AB}\,.\,t))\}_{k_{BS}})\}_{k_{AS}}; \\
A{\rightsquigarrow}B{:}\{(A\,.\,(k_{AB}\,.\,t))\}_{k_{BS}}; \epsilon
\end{array}
$$

Table 3. Denning-Sacco protocol, with formal declarations

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:

$$\textbf{private } k; A_1 \textbf{ knows } k; \ldots; A_n \textbf{ knows } k$$

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** $M; \ldots; A_n$ **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 behavior 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 $A{:}B!E$, reception $B{:}?x$, and checking $B{:}\phi$, which are explicitly attached to some principal, 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 §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 or encrypt one message with another (the key); (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 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, will be formalized later on using the *evaluation function* in Table 10.

$$\begin{array}{rcll}
E, F & ::= & a \mid A \mid \{E\}_F \mid (E\,.\,F) & \text{(expressions } \boldsymbol{E}\text{)} \\
 & & \mid x \mid \mathrm{D}_F(E) \mid \pi_1(E) \mid \pi_2(E) & \\
\phi, \psi & ::= & tt \mid [\,E{=}F\,] \mid [\,E{:}\,\boldsymbol{M}\,] \mid \phi \wedge \phi & \text{(formulae } \boldsymbol{F}\text{)} \\
I & ::= & \nu k \mid A{:}B!E \mid A{:}?x \mid A{:}\phi & \text{(simple action)} \\
X & ::= & \epsilon \mid I; X & \text{(executable narrations } \boldsymbol{X}\text{)}
\end{array}$$

Table 4. Syntax of executable narrations

Formulae ϕ on received messages are described by (conjunctions of) two 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.

Table 4 lists the syntax of expressions, formulae and executable narrations. In the following, we will omit the trailing $;\epsilon$ 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 behavior 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.

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 generalize this representation of principal's 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 5.*

$$\textsc{syn-pair}\ \frac{(M,E) \in \mathcal{S}(K) \qquad (N,F) \in \mathcal{S}(K)}{((M\,.\,N),(E\,.\,F)) \in \mathcal{S}(K)}$$

$$\textsc{syn-enc}\ \frac{(M,E) \in \mathcal{S}(K) \qquad (N,F) \in \mathcal{S}(K)}{(\{M\}_N,\{E\}_F) \in \mathcal{S}(K)}$$

Table 5. Synthesis

$$\textsc{ana-ini}\ \frac{(M,E) \in K}{(M,E) \in \mathcal{A}_0(K)}$$

$$\textsc{ana-fst}\ \frac{((M\,.\,N),E) \in \mathcal{A}_n(K)}{(M,\pi_1(E)) \in \mathcal{A}_{n+1}(K)} \qquad \textsc{ana-snd}\ \frac{((M\,.\,N),E) \in \mathcal{A}_n(K)}{(N,\pi_2(E)) \in \mathcal{A}_{n+1}(K)}$$

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

$$\textsc{ana-dec-rec}\ \frac{(\{M\}_N,E) \in \mathcal{A}_n(K) \qquad (N,F) \notin \mathcal{S}(\mathcal{A}_n(K))}{(\{M\}_N,E) \in \mathcal{A}_{n+1}(K)}$$

$$\textsc{ana-nam-rec}\ \frac{(M,E) \in \mathcal{A}_n(K) \qquad M \in \boldsymbol{N} \cup \boldsymbol{A}}{(M,E) \in \mathcal{A}_{n+1}(K)}$$

Table 6. Analysis

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 6.*

Our definition of analysis refines the usual approach reminiscent of Paulson [Pau98]. Instead of directly defining a "flat" analysis set, we had to define a finitely stratified hierarchy $(\mathcal{A}_n(K))_{n\in\mathbb{N}}$. 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 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 6 (which amounts to admitting $\mathcal{A}_n(K) \subseteq \mathcal{A}_{n+1}(K)$). If we now analyze 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], which counters the goal of providing 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., de-

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

constructible) 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.

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 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 piece of data, then it better allows for decryption with the appropriate (corresponding) key—but only if it is known.

Less obviously required checks result from the following observation. Since a message (identifier) M may occur more than once in a protocol narration it may happen that, in some knowledge set, M is related to two different expressions E_1 and E_2. As M was precisely used in protocol narrations to indicate the *very same* piece of data, such a knowledge set can only be considered consistent if E_1 and E_2 indeed evaluate to the same message. 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 formalize these requirements, we generate consistency formulae.

Definition 3 (Consistency formula) *Let K be a knowledge set. Its* consistency formula $\Phi(K)$ *is defined as follows:*

$$\begin{aligned}\Phi(K) \stackrel{\text{def}}{=} \quad & \bigwedge_{((M\,.\,N),E)\in K} \ ([\pi_1(E):\boldsymbol{M}] \wedge [\pi_2(E):\boldsymbol{M}]) \\ \wedge \ & \bigwedge_{(\{M\}_N,E)\in K \,\wedge\, (N,F)\in\mathcal{S}(K)} \ [\mathrm{D}_F(E):\boldsymbol{M}] \\ \wedge \ & \bigwedge_{(M,E_i)\in K \,\wedge\, (M,E_j)\in K \,\wedge\, E_i\neq E_j} \ [E_i = E_j]\end{aligned}$$

The third conjunction clause actually may include some of the checks produced in the other conjunction clauses. Since our main goal was to capture all possible checks in a uniform manner, we accept this redundancy.

Usually, 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 3 remembers the duplication.

Definition 4 (Irreducibles) *Let K be a knowledge set. We define the set of* irreducibles $\mathcal{I}(K)$ *as follows:*

$$\mathcal{I}(K) \stackrel{\text{def}}{=} \mathcal{A}(K) \setminus \left(\begin{array}{c} \{((M\,.\,N), E) \in \mathcal{A}(K)\} \\ \cup\, \{(\{M\}_N, E) \in \mathcal{A}(K) \mid \exists F \in \boldsymbol{E} : (N, F) \in \mathcal{S}(\mathcal{A}(K))\} \end{array} \right)$$

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 1 *To see all the previous definitions in action, we consider the initial knowledge $K_0 = \{(A, A), (B, B), (S, S), (k_{AS}, k_{AS}), (t, t)\}$ of agent A of the Denning-Sacco protocol given Table 3.*

(For the sake of readability, we sometimes write $M \bullet E$ for (M, E).)

We now consider $K \stackrel{\text{def}}{=} K_0 \cup \{(\{((B\,.\,(k_{AB}\,.\,t))\,.\,\{(A\,.\,(k_{AB}\,.\,t))\}_{k_{BS}})\}_{k_{AS}}, x_1)\}$.

$$\text{We have } \mathcal{A}(K) = K \cup \left\{ \begin{array}{r} ((B\,.\,(k_{AB}\,.\,t))\,.\,\{(A\,.\,(k_{AB}\,.\,t))\}_{k_{BS}}) \bullet \mathrm{D}_{k_{AS}}(x_1) \\ (B\,.\,(k_{AB}\,.\,t)) \bullet \pi_1(\mathrm{D}_{k_{AS}}(x_1)) \\ \{(A\,.\,(k_{AB}\,.\,t))\}_{k_{BS}} \bullet \pi_2(\mathrm{D}_{k_{AS}}(x_1)) \\ B \bullet \pi_1(\pi_1(\mathrm{D}_{k_{AS}}(x_1))) \\ (k_{AB}\,.\,t) \bullet \pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1))) \\ k_{AB} \bullet \pi_1(\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1)))) \\ t \bullet \pi_2(\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1)))) \end{array} \right\}.$$

$$\begin{array}{rl} \Phi(\mathcal{A}(K)) = & [\mathrm{D}_{k_{AS}}(x_1) : \boldsymbol{M}] \\ & \wedge\ [\pi_1(\mathrm{D}_{k_{AS}}(x_1)) : \boldsymbol{M}] \wedge [\pi_2(\mathrm{D}_{k_{AS}}(x_1)) : \boldsymbol{M}] \\ & \wedge\ [\pi_1(\pi_1(\mathrm{D}_{k_{AS}}(x_1))) : \boldsymbol{M}] \wedge [\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1))) : \boldsymbol{M}] \\ & \wedge\ [\pi_1(\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1)))) : \boldsymbol{M}] \wedge [\pi_2(\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1)))) : \boldsymbol{M}] \\ & \wedge\ [\pi_1(\pi_1(\mathrm{D}_{k_{AS}}(x_1))) = B] \wedge [\pi_2(\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1)))) = t] \end{array}$$

$$\text{And finally, } \mathrm{rep}(\mathcal{I}(K)) = K_0 \cup \left\{ \begin{array}{r} \{(A\,.\,(k_{AB}\,.\,t))\}_{k_{BS}} \bullet \pi_2(\mathrm{D}_{k_{AS}}(x_1)) \\ k_{AB} \bullet \pi_1(\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1)))) \end{array} \right\}.$$

Here, $\mathrm{rep}(\mathcal{I}(K))$ *includes (t, t) instead of $(t, \pi_2(\pi_2(\pi_1(\mathrm{D}_{k_{AS}}(x_1)))))$ and (B, B) instead of $(B, \pi_1(\pi_1(\mathrm{D}_{k_{AS}}(x_1))))$.*

The compilation. We now have set up all the required ingredients to compile an extended protocol narration into an executable protocol narration.

Definition 5 (Compilation) *The translation $\mathcal{X}[\![\cdot]\!]^{(\upsilon, \varpi, \kappa, \nu)} : \boldsymbol{D} \to \boldsymbol{X}$ is defined inductively in Table 7, 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).*

[3] Choose a well-founded total order for expressions and select the smallest expression.

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., $\kappa(A) = \mathrm{rep}(\mathcal{I}(\kappa(A)))$. To update $\mathrm{f} \in \{\kappa, \upsilon\}$, we note $\mathrm{f}\,[x \leftarrow y]$ with $\mathrm{f}\,[x \leftarrow y]\,(x) = y$ and $\mathrm{f}\,[x \leftarrow y]\,(z) = \mathrm{f}(z)$ 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.

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 just the consistency check that allows us then to continue with the knowledge in reduced form.

Example 2 *Let DS be the Denning-Sacco protocol presented Table 3.*

We have $\kappa_{DS} : \boldsymbol{A} \to \boldsymbol{K}$

$$\begin{aligned} A &\mapsto \{(A,A),(B,B),(S,S)\} \\ B &\mapsto \{(A,A),(B,B),(S,S)\} \\ S &\mapsto \{(A,A),(B,B),(S,S)\} \end{aligned}$$

DS is well-formed and its compilation is

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

where $\phi_0 \stackrel{\text{def}}{=}$

$$\begin{aligned} &[\pi_1(x_0){:}\boldsymbol{M}] \wedge [\pi_2(x_0){:}\boldsymbol{M}] \\ \wedge\ &[\pi_1(x_0){=}A] \wedge [\pi_2(x_0){=}B] \end{aligned}$$

$\phi_1 \stackrel{\text{def}}{=} \Phi(\mathcal{A}(K))$ *where K has been defined in Example 1*

$\phi_2 \stackrel{\text{def}}{=}$

$$\begin{aligned} &[\mathrm{D}_{k_{BS}}(x_2){:}\boldsymbol{M}] \\ \wedge\ &[\pi_1(\mathrm{D}_{k_{BS}}(x_2)){:}\boldsymbol{M}] \wedge [\pi_2(\mathrm{D}_{k_{BS}}(x_2)){:}\boldsymbol{M}] \\ \wedge\ &[\pi_1(\pi_2(\mathrm{D}_{k_{BS}}(x_2))){:}\boldsymbol{M}] \wedge [\pi_2(\pi_2(\mathrm{D}_{k_{BS}}(x_2))){:}\boldsymbol{M}] \\ \wedge\ &[\pi_1(\mathrm{D}_{k_{BS}}(x_2)){=}A] \wedge [\pi_2(\pi_2(\mathrm{D}_{k_{BS}}(x_2))){=}t] \end{aligned}$$

[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.

$$
\begin{aligned}
\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 \\
&\quad \text{where } K'_A \stackrel{\text{def}}{=} \kappa(A) \cup \{(M,M)\} \\
&\quad \text{and} \quad \kappa' \stackrel{\text{def}}{=} \kappa\,[A \leftarrow \mathrm{rep}(\mathcal{I}(K'_A))] \\
\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)) \\
&\quad \text{where } K'_A \stackrel{\text{def}}{=} \kappa(A) \cup \{(n,n)\} \\
&\quad \text{and} \quad \kappa' \stackrel{\text{def}}{=} \kappa\,[A \leftarrow \mathrm{rep}(\mathcal{I}(K'_A))] \\
&\quad \text{and} \quad \nu' \stackrel{\text{def}}{=} \nu\,[A \leftarrow \nu(A) \cup \{n\}] \\
\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)) \\
&\quad \text{where} \quad x \stackrel{\text{def}}{=} \min(\boldsymbol{V} \setminus \upsilon) \\
&\quad \text{and} \quad K'_B \stackrel{\text{def}}{=} \kappa(B) \cup \{(M,x)\} \\
&\quad \text{and} \quad \kappa' \stackrel{\text{def}}{=} \kappa\,[B \leftarrow \mathrm{rep}(\mathcal{I}(K'_B))] \\
&\quad \text{and} \quad \phi \stackrel{\text{def}}{=} \Phi(\mathcal{A}(K'_B))
\end{aligned}
$$

Table 7. Definition of $\mathcal{X}[\![\cdot]\!]^{\cdot}$

In the last example, the obtained formulae apparently contain some redundant checks. As usual, two formulae ϕ and ψ may be considered equivalent if for all substitutions $\sigma : \boldsymbol{V} \to \boldsymbol{M}$, we have $[\![\phi\sigma]\!] = [\![\psi\sigma]\!]$. Then, for example, ϕ_2 is equivalent to ϕ'_2 where:

$$\phi'_2 \stackrel{\text{def}}{=} [\,\pi_1(\mathrm{D}_{k_{BS}}(x_2)) = A\,] \wedge [\,\pi_2(\pi_2(\mathrm{D}_{k_{BS}}(x_2))) = t\,]$$

Formulae like ϕ_2 are to be generated automatically, and it seems mandatory to also provide automatic simplification to remove redundant checks. However, in general, it is not obvious to define an intuitive notion of *minimality* for formulae. For example, $\phi = [\,E_1 = F_1\,] \wedge [\,E_2 = F_2\,]$ and $\psi = [\,(E_1\,.\,E_2) = (F_1\,.\,F_2)\,]$ are equivalent; which one should be qualified as simpler? An even more interesting case is $\phi = [\,E = F\,] \wedge [\,G : \boldsymbol{M}\,]$ and $\psi = [\,\pi_1((E\,.\,G)) = F\,]$, which indicates that there is a trade-off between the bare number of conjoints and their size.

3 Executing protocol narrations

In this section, we propose an operational semantics for narrations. It proceeds in a traditional syntax-directed manner by analyzing the current top-level construct

$$\epsilon\{^{M}/_{x}\}_{@A} \stackrel{\text{def}}{=} \epsilon$$

$$(A'{:}B!E; X)\{^{M}/_{x}\}_{@A} \stackrel{\text{def}}{=} \begin{cases} A'{:}B!E; X\{^{M}/_{x}\}_{@A} & \text{if } A' \neq A \\ A{:}B!E\{^{M}/_{x}\}; X\{^{M}/_{x}\}_{@A} & \text{otherwise} \end{cases}$$

$$(A'{:}?y; X)\{^{M}/_{x}\}_{@A} \stackrel{\text{def}}{=} \begin{cases} A'{:}?y; X\{^{M}/_{x}\}_{@A} & \text{if } A' \neq A \\ A{:}?y; X\{^{M}/_{x}\}_{@A} & \text{if } A = A' \text{ and } y \neq x \\ A{:}?x; X & \text{otherwise} \end{cases}$$

$$(A'{:}\phi; X)\{^{M}/_{x}\}_{@A} \stackrel{\text{def}}{=} \begin{cases} A'{:}\phi; X\{^{M}/_{x}\}_{@A} & \text{if } A' \neq A \\ A{:}\phi\{^{M}/_{x}\}; X\{^{M}/_{x}\}_{@A} & \text{otherwise} \end{cases}$$

$$(\nu n; X)\{^{M}/_{x}\}_{@A} \stackrel{\text{def}}{=} \nu n; X\{^{M}/_{x}\}_{@A}$$

Table 8. Substitution

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 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 sort 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 8 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.

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

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

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

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

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

Table 9. Reordering

Definition 6 *The* reordering $\cong \subseteq \boldsymbol{X} \times \boldsymbol{X}$ *is the least equivalence relation satisfying the rules given in Table 9, 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 behavior 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.

Evaluation of expressions and formulae. Table 10 shows how to evaluate expressions and formulae. The definitions are straightforward and offer no surprises, except for allowing the observation that $[\,E{:}\boldsymbol{M}\,]$ is just a macro for $[\,E{=}E\,]$.

Labeled transitions. We define a straightforward labeled semantics of executable narrations, in style influenced by semantics for 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 like

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

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.

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 $[\![F]\!] = N \in \boldsymbol{M}$
$[\![E]\!] \stackrel{\text{def}}{=} \bot$	in all other cases
Definition of $[\![\cdot]\!] : \boldsymbol{F} \to \{\mathbf{true}, \mathbf{false}\}$	
$[\![tt]\!] \stackrel{\text{def}}{=} \mathbf{true}$	
$[\![\phi \wedge \psi]\!] \stackrel{\text{def}}{=} [\![\phi]\!]$ and $[\![\psi]\!]$	
$[\![[\,E = F\,]]\!] \stackrel{\text{def}}{=} \mathbf{true}$	if $[\![E]\!] = [\![F]\!] = M \in \boldsymbol{M}$
$[\![[\,E : \boldsymbol{M}\,]]\!] \stackrel{\text{def}}{=} \mathbf{true}$	if $[\![E]\!] = M \in \boldsymbol{M}$
$[\![\phi]\!] \stackrel{\text{def}}{=} \mathbf{false}$	in all other cases

Table 10. Evaluation of expressions (can fail, in particular if $\mathrm{v}(E) \neq \emptyset$) and formulae

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

$$\textsc{Check}\ \frac{X \xrightarrow{A:\beta} X'}{A{:}\phi; X \xrightarrow{A:\beta} X'}\ [\![\phi]\!] = \mathbf{true} \qquad \textsc{Open}\ \frac{X \xrightarrow{A:(\nu\tilde{n})\,B!M} X'}{\nu z; X \xrightarrow{A:(\nu z\tilde{n})\,B!M} X'}\ z \in \mathrm{n}(M) \setminus \{\tilde{n}\}$$

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

Table 11. Labeled semantics of executable narrations

4 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:

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

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

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

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

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

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

Table 12. Labeled semantics of spi-calculus

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 labeled 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

Communication can only occur on agent names. Moreover, since it is syntactically not possible to hide an agent name from outside, we do not consider internal communications. 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 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 §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 ($X\!\upharpoonright_A$ of Definition 7) and explicitly represented using the parallel composition operator of the spi-calculus. Any intended sequential occurrence of actions, namely those actions that are associated to 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 restricted around the parallel composition.

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

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

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

Definition 7 (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:*

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

   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.*

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

Theorem 1. *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 $\mu = \mathcal{T}[\![A{:}\beta]\!]$.*

5 Conclusions

Contributions. In summary, we stepwise enhance protocol narrations in order to build up enough structure such that a well-motivated formalization of their 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.

In summary, 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 §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 §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.

Tool support. We have implemented the previous developments in OCaml, including the syntactic sugar mentioned at the end of Section 1. Due to the big size of the formulae generated, we have studied possible simplifications for them. To this end, we have implemented naive 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 apparently gives good result in practice.

We have also investigated extension of the work of this paper towards richer message languages (i.e. with public/private key, hashing). It appears that simplifying formulae becomes even more of a necessity.

Related work. Sumii et. al. [STY05] propose a formal semantics of narrations by translation into spi-calculus. The paper is written in Japanese, so it remains unclear to us how they treat the problem of checks-on-reception. In any case, our intention was to provide a formal semantics that does *not require* an underlying (too) general process calculus, so our approach is still substantially different.

The work of Bodei et. al. [BBD$^+$03] 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 process calculus LySA. In the above paper (the only that we are aware of), the system underlying their "systematic expansion" is not unambiguously explained, while our expansion is fully automatic and generates a maximum number of checks. Finally, their approach aims at static analysis techniques, while we ultimately target at dynamic analysis in the form of bisimulation checks [BBN04] in the full spi-calculus (LySA is channel-free).

In other related approaches, narrations are reformulated or translated using Casper [Low98], HLPSL2IF [BMV03], CAPSL [Mil], CASRUL [JRV00], or (s)pi-calculus [AG99,Bla03]. 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 formalize 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 [Low98] 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.

Future work. Here, we do not tackle the fourth task listed by Abadi [Aba00] on how to get to a formalization 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 non-trivially extend the narration notation rather than providing a suboptimal semantics to an inappropriate notation. Note that this confusion also rules out the naïve modeling of concurrent sessions by the bare unbounded replication within spi-calculus. Some inspiration from the work of Cremers and Mauw [CM05] may help us here.

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

References

[Aba00] M. Abadi. Security Protocols and their Properties. In *Foundations of Secure Computation*, pages 39–60. NATO ASI, IOS Press, 2000.

[AG99] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.

[BBD+03] C. Bodei, M. Buchholtz, P. Degano, F. Nielson and H. Nielson. Automatic validation of protocol narration. In *Proceedings of 16th IEEE Computer Security Foundations Workshop (CSFW 16)*, pages 126–140, 2003.

[BBN04] J. Borgström, S. Briais and U. Nestmann. Symbolic Bisimulation in the Spi Calculus. In *Proceedings of CONCUR 2004*, volume 3170 of *LNCS*, pages 161–176. Springer, Sept. 2004.

[Bla03] B. Blanchet. Automatic Verification of Cryptographic Protocols: A Logic Programming Approach. In *Proceedings of Principles and Practice of Declarative Programming (PPDP'03)*. ACM, 2003.

[BMV03] D. Basin, S. Mödersheim and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, Heidelberg, 2003.

[Bri04] S. Briais. Formal proofs about hedges using the Coq proof assistant, 2004. http://lamp.epfl.ch/~sbriais/spi/hedges/hedge.html.

[CJ97] J. A. Clark and J. L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, University of York, 1997.

[CM05] C. Cremers and S. Mauw. Operational Semantics of Security Protocols. In *Scenarios: Models, Algorithms and Tools (Dagstuhl 03371 Post-Seminar Proceedings)*, volume 3466 of *LNCS*, 2005.

[DY83] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, Mar. 1983.

[Gen03] C. Gensoul. *Spyer — un compilateur de protocoles cryptographiques*. Semester Project Report, EPFL, July 2003.

[JRV00] F. Jacquemard, M. Rusinowitch and L. Vigneron. Compiling and Verifying Security Protocols. In *Logic for Programming and Automated Reasoning*, volume 1955 of *LNCS*, pages 131–160. Springer-Verlag, November 2000.

[Low98] G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6:53–84, 1998.

[Mil] J. K. Millen. CAPSL: Common Authentication Protocol Specification Language. http://www.csl.sri.com/users/millen/capsl/.

[MvOV96] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Pau98] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[STY05] E. Sumii, H. Tatsuzawa and A. Yonezawa. Translating Security Protocols from Informal Notation into Spi Calculus. *IPSJ Transactions on Programming*, 45, 2005. Written in Japanese, abstract in English. To appear.