

# A Calculus of Challenges and Responses <sup>\*</sup>

Michael Backes<sup>\*\*</sup>, Agostino Cortesi<sup>\*\*\*</sup>, Riccardo Focardi<sup>\*\*\*</sup>, Matteo Maffei<sup>\*\*</sup>

**Abstract.** This paper presents a novel approach for concisely abstracting authentication protocols and for subsequently analyzing those abstractions in a sound manner, i.e., deriving authentication guarantees for protocol abstractions suffices for proving these guarantees for the actual protocols. The abstractions are formalized in a novel calculus which constitutes a higher-level abstraction of the  $\rho$ -spi calculus and is specifically tailored towards reasoning about challenge-response mechanisms within authentication protocols. Furthermore, it allows for expressing protocols without having to include details on the specific structure of exchanged messages. This in particular entails that many authentication protocols share a common abstraction so that a single validation of this abstraction already gives rise to security guarantees for all these protocols. Moreover, such an abstract validation can be automatically performed using static analysis techniques based on an effect system proposed in this paper. Finally, extensions to abstractions of additional protocol classes immediately enjoy a soundness theorem provided that these extensions satisfy certain explicit, easily checkable conditions.

## 1 Introduction

Language-based security has proved to be a salient technique for formally analyzing security protocols, dating back to Abadi’s seminal work on secrecy by typing [1]. Since then, a variety of language-based techniques have been developed and successfully applied to analyzing an increasingly large class of security protocols (a far from being comprehensive list includes [17, 2, 13, 4, 12, 5, 10, 14, 8]). The ability to reason about security at the language level often allowed for concisely clarifying why certain message components are included in a protocol, how their entirety suffices for establishing desired security guarantees, and for identifying ambiguities in protocol messages that could be exploited by an adversary to mount a successful attack against the protocol.

Authentication protocols are known to require very subtle reasoning about encrypting and decrypting various related messages in the presence of a powerful adversary, with each of these encryptions being meant to contribute some part of the overall authentication guarantee. Thus they are strongly vulnerable to attacks originating by a certain degree of ambiguity in the protocol message, e.g., man-in-the-middle attacks or attacks where certain encryptions are re-used by the adversary in another protocol execution where they suddenly get a different semantics. This turns authentication protocols into particularly suitable targets of language-based security, and several current language-based static analysis techniques [5, 14, 8] for tackling this problem have been proposed, e.g., based on protocols narrations formalized in the spi-calculus [3], or in variants thereof such as Lysa [4] and the  $\rho$ -spi calculus [10]. Roughly speaking, these techniques typically rely on some static patterns, defined on the syntax of the calculus, which suffice for showing that the run-time protocol execution respects certain intended challenge-response schemes which in turn imply the desired authentication guarantees. For instance, the type system by Gordon and Jeffrey [8, 14] relies on some type information provided by the user, which suffices for identifying nonces and formalizing in which extent their exchange contributes to achieve authentication. The generality of the analysis is sometimes paid by sophisticated type definitions that have to be defined manually and thus require a certain degree

---

<sup>\*</sup> Work partially supported by MIUR Project COFIN 2004013015 “Abstract Interpretation: Design and Applications” (AIDA) and MIUR Project 2005015785 “Logical Foundations of Distributed Systems and Mobile Code”.

<sup>\*\*</sup> Saarland University, Saarbrücken, Germany, {backes,maffei}@cs.uni-sb.de

<sup>\*\*\*</sup> Università Ca’ Foscari di Venezia, Italy, {cortesi,focardi}@dsi.unive.it

of expertise from the programmers. The type system by Bugliesi, Focardi and Maffei [8] relies on some dynamic information attached to ciphertexts, in the form of tags, which univocally determines the role of messages in the authentication task. A great advantage is that tag and type definitions are inferred in polynomial time [16]. This analysis is general enough to cover several existing protocols but its scope is strictly constraint by the set of tagged ciphertexts.

This paper tackles the problem of analyzing authentication protocols from a novel and conceptually more abstract perspective. Starting from protocol narrations expressed in a variation of the  $\rho$ -spi calculus, we abstract from the specific structure of messages by solely focussing on the challenge-response components that are inherent in the protocols. The resulting more abstract protocols are formalized in a new process calculus, the *CR calculus*, which is specifically tailored to reasoning about challenge-response mechanisms within authentication protocols. The CR calculus hence grants a conceptually more abstract view of authentication protocols, enables more abstract and general proofs, and enjoys several nice properties, most of which constitute a significant advantage over existing approaches.

- Foremost, we prove a soundness theorem stating that abstractions preserve authentication properties, i.e., proving authentication for an abstraction of a protocol in the CR calculus suffices for obtaining an authentication proof for the actual protocol in the  $\rho$ -spi calculus.
- The abstraction induces a uniform representation of various different authentication protocols as it does not have to reflect details on the specific structure of exchanged messages. The way the abstraction is defined furthermore ensures that many authentication protocols share a common abstraction so that a single validation of this abstraction gives rise to security guarantees for all these protocols.
- The analysis is modular and compositional since each principal involved in the protocol is independently validated and the parallel composition of successfully validated protocols yields a secure protocol again. This fits very well the analysis of multi-protocol systems.
- The analysis can be automated by means of a static analysis technique based on the effect system proposed in this paper. We stress that the abstraction from  $\rho$ -spi to CR calculus protocol descriptions and the effect system used for verifying the safety of the latter are completely independent. This independence constitutes a key feature of our approach since it entails that refining the abstraction does not affect the soundness of the analysis and, conversely, refining the effect system presented in this paper or even the use of a different analysis technique does not affect the soundness of the abstraction.
- The abstraction is extensible in that extensions to additional protocol classes immediately enjoy a soundness theorem provided that these extensions satisfy certain explicit, easily checkable conditions. This is a key difference with respect to other type-based approaches [2, 8, 13, 14], where an extension of the analysis requires re-proving the soundness and the safety of the analysis from scratch.

**Outline.** Section 2 reviews the  $\rho$ -spi calculus and introduces a small novel dialect thereof. Section 3 presents the new CR calculus, which is specifically tailored to expressing challenge-response protocols by abstracting away from the specific structure of messages. Section 4 introduces the abstraction of  $\rho$ -spi protocol descriptions into the CR calculus. Section 5 proposes an effect system for checking the safety of protocols expressed in the CR calculus. Section 6 concludes and outlines future work.

## 2 Review of the $\rho$ -spi Calculus, and a Novel Dialect Thereof

The  $\rho$ -spi calculus [8, 10] is derived from the spi calculus [3] and inherits many of the features of *Lysa* [4], a dialect of the spi calculus specifically tailored to the analysis of authentication protocols. The  $\rho$ -spi calculus

**Table 1** (Our Dialect of) the  $\rho$ -spi Calculus**Notation:**  $M$  and  $N$  denote terms with no encryption and no tags.

Names	Terms	Processes
$a ::= n, m$ (Msgs)	$C ::= a$ (Names)	$P, Q ::= \text{new}(n).P$ (Res)
$I, J, A, B, E$ (Identities)	$x, y, z$ (Variables)	$\text{in}(C).P$ (In)
<b>Keys</b>	$\text{Tag}(C)$ (Tagged Term)	$\text{out}(C).P$ (Out)
$k ::= k_{IJ}$ (Shared Keys)	$(C_1, C_2)$ (Pair)	$\text{begin}_N(A, I, M_1; M_2).P$ (Begin)
$k_I^+, k_I^-$ (Asymmetric Keys)	$\{\!  C \!\}_k$ (Encryption)	$\text{end}_N(A, J, M_1; M_2).P$ (End)
		$A \triangleright P$ (Principal)
		$P Q$ (Par)
		$!P$ (Repl)
		$\mathbf{0}$ (Stop)

differs from both calculi in several respects: it incorporates the notion of tagged message exchange from [9], associates principal identities to processes and syntactically binds keys to their owners.

In this paper, we consider a novel dialect of  $\rho$ -spi in which encryptions and decryptions are performed on-the-fly when sending and receiving messages, respectively. This dialect in particular links protocol specifications more tightly to their informal “graphical” descriptions, which only depict sent and received messages without giving a precise semantics on how messages are parsed and constructed.

## 2.1 Syntax of our Dialect of the $\rho$ -spi Calculus

The formal syntax of our dialect of the  $\rho$ -spi calculus is depicted in Table 1. We presuppose a countable set  $\mathcal{N}$  of *names* partitioned into two distinct categories: *msgs* and *identities*. The set of identities  $ID$ , ranged over by  $I$  and  $J$ , are further partitioned into *trusted principals*  $ID_P$ , ranged over by  $A$  and  $B$ , and *enemies*  $ID_E$ , ranged over by  $E$ . *Keys* are partitioned into symmetric keys  $k_{IJ}$ , shared between  $I$  and  $J$ , and asymmetric keys  $k_I^+, k_I^-$  representing corresponding public and private keys belonging to  $I$ . Notice that keys are never transmitted on the network, thus modelling long-term encryption keys. We also presuppose a set  $\mathcal{T}$  of tags. Terms can be tagged using a  $\text{Tag} \in \mathcal{T}$ , thus determining their role in the authentication task. Moreover, terms can be composed in pairs or encrypted using keys.<sup>1</sup> The special name  $\varepsilon$ , used to denote the empty message, will be always omitted, e.g., the primitive  $\text{begin}_\varepsilon(A, B, M_1; \varepsilon)$  will be written as  $\text{begin}(A, B, M_1; )$ .

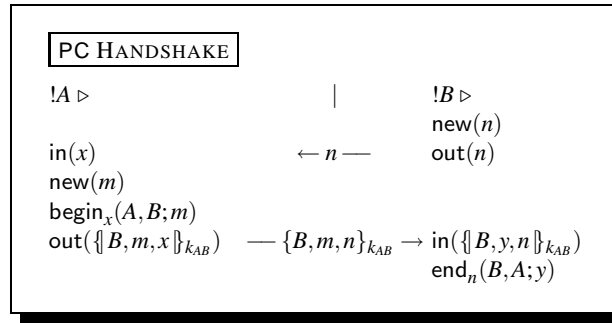
*Processes* (or *protocols*), ranged over by  $P$  and  $Q$ , behave as follows:  $\text{new}(n).P$  generates a fresh name  $n$  local to  $P$ . We presuppose a unique anonymous public channel, the network, from/to which all principals, including intruders, read and send messages. Similarly to *Lysa*, our input primitive may atomically test part of the read message, by employing pattern-matching. If the input message matches the input term, then the variables occurring in the term are bound to the remaining sub-part of the message; otherwise the message is not read at all. This mechanism is also used to decrypt received messages on-the-fly, cf. more details below, and thus constitutes an important novelty compared to the  $\rho$ -spi calculus; of course, in order to immediately match a message encrypted with asymmetric cryptography, the correct decryption key has to be specified in the term. For giving the intuition of the semantics, which is formally defined in Section 2.2, the process ‘ $\text{in}(n).P$ ’ tries to read a specific nonce  $n$  from the network and, if such a name can be read, no binding occurs and  $P$  is executed. This is useful, e.g., to check protocols where nonce  $n$  is sent encrypted as challenge and received back in clear as response. As another example, consider ‘ $\text{in}(\{\!| x \!\}_{k_A^-}).P$ ’. This process reads any

<sup>1</sup> For the sake of readability, in the rest of the paper we omit brackets: for instance, the nested pair  $((a, b), k)$  is simplified in  $a, b, k$ .

message encrypted with  $A$ 's public key  $k_A^+$ , i.e., messages of the form  $\{G\}_{k_A^+}$ , decrypts them on the fly, and binds all the free occurrences of  $x$  to  $G$  in process  $P$ .

Note that we distinguish between the *static* cryptographic terms  $C$  of the calculus, and the actual sent and received *messages*  $G$ . Encryption for terms is noted  $\{C\}_k$  while encrypted messages are noted  $\{G\}_k$ . This is crucial in the calculus semantics to distinguish between the simple reception and the reception-with-decryption of an encrypted message. For instance,  $\text{in}(x).P$  and  $\text{in}(\{y\}_{k_{AB}}).Q$  can both read message  $\{G\}_{k_{AB}}$  but the first process just reads it, noted  $\text{in}(\{G\}_{k_{AB}})$ , while the second one reads and decrypts it, noted  $\text{in}(\{G\}_{k_{AB}})$ . In particular,  $x$  is bound to the whole message  $\{G\}_{k_{AB}}$  while  $y$  is bound to the plaintext  $G$ . The  $\text{begin}_N(A, B, M_1; M_2)$  and  $\text{end}_N(B, A, M_1; M_2)$  primitives are used to check the *correspondence assertions* [18] in a nonce handshake between  $A$  and  $B$  based on nonce  $N$ . The former primitive declares that  $A$  is starting a protocol session with  $B$ , while the latter declares that  $B$  is ending a protocol session in which he believes to have correctly authenticated  $A$ . Messages  $M_1$  and  $M_2$ , when specified, represent messages exchanged respectively from  $B$  to  $A$  and from  $A$  to  $B$  during the protocol session. Finally,  $A \triangleright P$  represents principal  $A$  executing process  $P$ ; process  $P|Q$  is the parallel composition of  $P$  and  $Q$ ; process  $!P$  indicates an arbitrary number of parallel instances of  $P$ , and  $\mathbf{0}$  is the null process that does nothing. We will often omit  $\mathbf{0}$  from protocol specifications. Finally, we remark that the  $\rho$ -spi calculus comes with a notion of well-formedness checking that (i) identity declarations do not nest; (ii) the first identity in  $\text{begin}$  and  $\text{end}$  assertions refers to the principal running the process; and (iii) principals only use their own keys. In the following, we shall always consider well-formed processes. We refer to Appendix A for more details.

*Example 1.* To illustrate the calculus, let us consider a simple challenge-response protocol in which a nonce is sent in clear and received back encrypted with a long-term shared key. Following [8, 10], we call this kind of handshake PC, meaning *Plaintext* challenge and *Ciphertext* response:



Notice the use of variables  $x$  and  $y$  to receive the nonce and the message, respectively. Recall that names are checked for equality, when specified in a term. So, for example,  $\text{in}(\{B, y, n\}_{k_{AB}})$  receives and decrypts the message only if the key is  $k_{AB}$  and the first and third components are  $B$  and  $n$ , respectively. The idea of the protocol is that only  $A$ , who shares the long term key  $k_{AB}$  with  $B$ , is able to send the correct reply. The identity label  $B$  is used for specifying the intended receiver of the ciphertext, thus preventing reflection attacks. Through  $\text{begin}_x(A, B; m)$ , principal  $A$  declares the beginning of the protocol based on nonce  $x$  for authenticating  $m$ . On the other side,  $B$  ends up the protocol after checking that  $n$  has been encrypted with  $k_{AB}$  and, through  $\text{end}_n(B, A; y)$ , he declares to have authenticated, based on nonce  $n$ , “ $A$  sending message  $y$  to  $B$ ”. Notice that the above protocol specification models an unbounded number of sessions between  $A$  and  $B$ , thanks to the replication operator.  $\square$

## 2.2 Operational Semantics of our Dialect of the $\rho$ -spi Calculus

We define the operational semantics of our dialect of  $\rho$ -spi in terms of *traces*, following [6]. A trace is a possible sequence of *actions* performed by a process. Each process primitive has an associated action and we denote with  $Act$  the set of all possible actions. The dynamics of the calculus is formalized by means of a transition relation between *configurations*, i.e., pairs  $\langle s, P \rangle$ , where  $s \in Act^*$  is a trace and  $P$  is a closed process. Each transition  $\langle s, P \rangle \rightarrow \langle s :: \alpha, P' \rangle$  simulates one computation step in  $P$  and records the corresponding action  $\alpha$  in the trace. We denote by  $\rightarrow^*$  a finite non-empty sequence of computation steps.

Principals do not directly synchronize with each other. Instead, they may receive from the unique channel an arbitrary message known by the environment, which models the Dolev-Yao intruder [11]: the knowledge of the environment is formalized by a set of deduction rules stating that the environment knows all the messages sent on the network, every name which is not restricted in the trace, all public keys, and every shared and private key with  $E$  in the subscript, e.g.,  $k_{EI}$  and  $k_E^-$ . Moreover, the environment can tag and untag messages, construct or destruct pairs, and encrypt/decrypt messages if the appropriate key is known. Due to space constraints, the formal definition of the transition relation and the deductive system formalizing the knowledge of the environment is postponed to Appendix B.

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

The notion of safety extends the standard *correspondence* property of [15, 18] by distinguishing between received and sent messages and pointing out the nonce which the handshake is based on.

**Definition 2 (Safety).** *A trace  $s$  is safe iff whenever  $s = s_1 :: \text{end}_n(B, A, G_1; G_2) :: s_2$ , then we have  $s_1 = s'_1 :: \text{begin}_n(A, B, G_1; G_2) :: s''_1$ , such that  $s'_1 :: s''_1 :: s_2$  is safe. A process  $P$  is safe if  $s$  is safe for all  $s \in T(P)$ .*

A trace is thus safe if every  $\text{end}_n(B, A, G_1; G_2)$  is preceded by a distinct  $\text{begin}_n(A, B, G_1; G_2)$ . Intuitively, this guarantees that whenever  $B$  authenticates  $A$  receiving  $G_1$  and sending  $G_2$ , then  $A$  has received  $G_1$  and has sent  $G_2$  in a handshake with  $B$  based on nonce  $n$ .

## 3 CR Calculus

In the previous section, we presented a calculus for specifying and reasoning on cryptographic authentication protocols. Now, we want to abstract away from the specific structure of messages, and in particular from symbolic cryptography, and to reason on authentication protocols just in terms of challenges and responses. To illustrate, let us consider again the protocol of Example 1.  $B$  challenges  $A$  to encrypt a freshly generated nonce  $n$ , together with identifier  $B$  and message  $m$ , using a shared key. Intuitively, the first message is a public challenge and the second one is a private response, directed to  $B$ . Notice that the identifier  $B$  is important to disambiguate the sender of the message: By receiving  $\{m, n\}_{k_{AB}}$ ,  $B$  would not be able to tell whether the message was sent by  $A$  or by  $B$  itself, given that there is nothing indicating the “direction” of the message. In the protocol above, instead,  $A$  and  $B$  agree that the identifier in the message represents the intender receiver, which can be directly checked by the receiver before concluding the protocol.

An abstract view of the protocol is depicted in Table 2. The idea is that cryptographic challenges or responses are abstracted into two special messages, namely  $\text{Chal}_N^{\ell, \ell'}(B, A, M_1)$  and  $\text{Resp}_N^{\ell, \ell'}(A, B, M_2)$ . The former may be read as “challenge sent by  $B$  to  $A$  for authenticating message  $M_1$  with nonce  $N$ ”. Similarly, the

**Table 2** Simple CR protocol

A	B
	$\leftarrow \text{Chal}_n^{\text{Pub,Priv}}(B, A) \text{ ---}$
$\text{--- Resp}_n^{\text{Pub,Priv}}(A, B, m) \rightarrow$	

latter may be read as "response sent by  $A$  to  $B$  for authenticating message  $M_2$  with nonce  $N$ ". Labels  $\ell, \ell' \in \{\text{Pub}, \text{Taint}, \text{Int}, \text{Priv}\}$  specify the security level of the challenge and the response, respectively, with the following intuitive meaning:

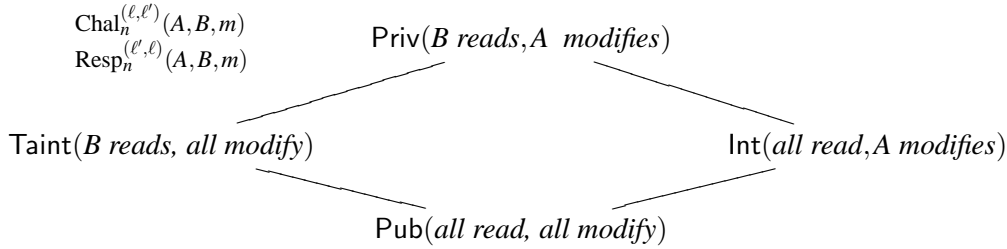
**Pub** : readable and modifiable by everyone; cleartexts fall in this class;

**Taint** : modifiable by everyone but readable only by the intender receiver; public key cryptography is an example of this kind;

**Int** : readable by everyone but modifiable only by the sender; for example, digital signatures;

**Priv** : readable only by the intender receiver and modifiable only by the sender; this is achievable through, e.g., "authenticated symmetric encryption" as soon as the "direction" is made explicit as previously explained.

In the example above, the labels **Pub**, **Priv** specify that the challenge is public, i.e., sent as plaintext, and the response is private, i.e., encrypted with a symmetric key. The possibility of reading or modifying challenges/responses induces the following partial order  $\leq$  on security labels,



Messages with  $\ell \leq \text{Taint}$  may be modified by everyone, while messages with  $\ell \leq \text{Int}$  may be read by everyone. Moreover, only  $B$  can read messages with  $\ell \geq \text{Taint}$  and only  $A$  can generate messages with  $\ell \geq \text{Int}$ . There are two ways in which a principal  $B$  can authenticate himself: (i) by sending a response which only  $B$  can generate, i.e., **Int** or **Priv** ; and (ii) by replying to a challenge which only  $B$  can read, i.e., **Taint** or **Priv** . (cf. Remark 2.)

For example,  $\text{Resp}_n^{\text{Pub}, \text{Int}}(A, B, m)$  represents an integer response to a public challenge  $\text{Chal}_n^{\text{Pub}, \text{Int}}(B, A)$ , which might be concretized through symbolic cryptography as in protocol a (cf. Table 3):  $A$  proves her identity by signing the nonce together with 'B' and message  $m$ . As another example, consider

$\text{Resp}_n^{\text{Taint}, \text{Taint}}(A, B, m_2)$  representing a tainted response to a tainted challenge  $\text{Chal}_n^{\text{Taint}, \text{Taint}}(B, A, m_1)$ , which might be concretized as in protocol b:  $A$  proves her identity by decrypting the tainted challenge using her private key. Attention should be paid when the security label of the challenge and the response is the same and symmetric key cryptography is used. For example,  $\text{Resp}_n^{\text{Priv}, \text{Priv}}(A, B, m_2)$  with challenge  $\text{Chal}_n^{\text{Priv}, \text{Priv}}(B, A, m_1)$  should never be concretized using messages that might be confused. For example, consider the worst case in which challenge and response are the same, as in protocol c. Here the enemy can trivially attack the protocol by replaying the received challenge back to  $B$ , authenticating as  $A$ . This is why the CR calculus explicitly distinguishes between challenges and responses and the abstraction from  $\rho$ -spi to

**Table 3** Protocol concretizations

(protocol a)		(protocol b)		(protocol c)	
A	B	A	B	A	B
$\leftarrow n$		$\leftarrow \{A, n, m_1\}_{k_A^+}$		$\leftarrow \{B, m, n\}_{k_{AB}}$	
$\rightarrow \{B, m, n\}_{k_A^-}$		$\rightarrow \{B, n, m_2\}_{k_B^+}$		$\rightarrow \{B, m, n\}_{k_{AB}}$	

**Table 4** CR Calculus Names and Terms

**Notation:**  $\ell \in \{\text{Pub}, \text{Taint}, \text{Priv}, \text{Int}\}$ .  $M^\#$  and  $N^\#$  denote terms with no challenges and responses.

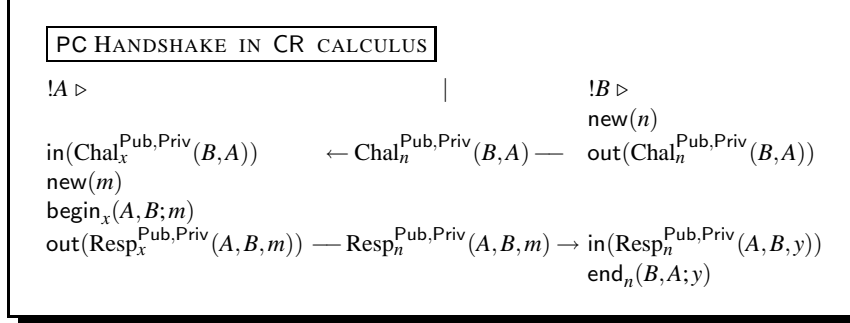
CR Names		CR Terms	
$a^\# ::= n^\#, m^\#$	(Msgs)	$C^\# ::= a^\#$	(Names)
$I^\#, J^\#, A^\#, B^\#, E^\#$	(Identities)	$x^\#, y^\#, z^\#$	(Variables)
$\top$	(Any)	$(C_1^\#, C_2^\#)$	(Pair)
		$\text{Chal}_{N^\#}^{(\ell, \ell)}(I^\#, J^\#, M^\#)$	(Chal)
		$\text{Resp}_{N^\#}^{(\ell, \ell)}(I^\#, J^\#, M^\#)$	(Resp)

the CR calculus guarantees that they remain distinguishable even when concretized through symbolic cryptography (see Section 4).

### 3.1 Syntax

The calculus of Challenges and Responses, called CR calculus, has the same syntax as our dialect of  $\rho$ -spi calculus apart from names and terms, reported in Table 4. CR names, noted  $a^\#$ , correspond to the names in  $\rho$ -spi plus  $\top$ , used for abstracting an arbitrary  $\rho$ -spi term. Terms, instead, have neither tags nor symbolic cryptography, but include challenges and responses. CR processes, ranged over by  $P^\#$ , have the same syntax as  $\rho$ -spi processes but use the CR names and terms described above.

*Example 2.* We show the CR calculus specification corresponding to Example 1. We omit the  $\bullet^\#$  notation for the sake of readability.



$A$  reads a public challenge on nonce  $x$  coming (apparently) from  $B$ . She generates a new message  $m$ , starts the protocol session, and sends to  $B$  a private reply containing  $m$  and based on the received nonce  $x$ . On the other side,  $B$  generates the nonce  $n$ , sends the public challenge to  $A$ , waits for the private reply containing the same nonce  $n$ , and commits. Authentication of  $A$  is guaranteed by the fact that only  $A$  can generate such a response and the nonce is fresh and used only once.  $\square$

As for the  $\rho$ -spi calculus, we have a notion of process well-formedness that rules out undesired process behaviors and enforces the scope of challenges and responses to the principals that can actually generate and receive them. In the following, we shall always consider well-formed processes. For more detail, please refer to Appendix A.

**Table 5** Abstraction

Terms	Processes
$\alpha(a) = a^\#$	$\alpha(\mathbf{0}) = \mathbf{0}$
$\alpha(x) = x^\#$	$\alpha(\text{new}(n).P) = \text{new}(\alpha(n)).\alpha(P)$
$\alpha(C_1, C_2) = (\alpha(C_1), \alpha(C_2))$	$\alpha(\text{in}(\bar{C}).P) = \text{in}(\alpha(C)).\alpha(P)$
$\alpha(\text{Tag}(C)) = \alpha(C)$	$\alpha(\text{out}(C).P) = \text{out}(\alpha(C)).\alpha(P)$
$\alpha(\llbracket C \rrbracket_k) = \begin{cases} \underline{f}(\llbracket C \rrbracket_k) & \text{if } \llbracket C \rrbracket_k \in \text{dom}(\underline{f}) \\ \alpha(C) & \text{otherwise} \end{cases}$	$\alpha(\text{begin}_N(A, I, M_1; M_2).P) = \text{begin}_{\alpha(N)}(\alpha(A), \alpha(I), \alpha(M_1); \alpha(M_2)).\alpha(P)$
Notice that:	$\alpha(\text{end}_N(A, I, M_1; M_2).P) = \text{end}_{\alpha(N)}(\alpha(A), \alpha(I), \alpha(M_1); \alpha(M_2)).\alpha(P)$
- $\underline{f} : \llbracket C \rrbracket_k \mapsto \text{Chal/Resp}_{N^\#}^{\ell, \ell'}(I^\#, J^\#, M^\#)$	$\alpha(A \triangleright P) = \alpha(A) \triangleright \alpha(P)$
- $\bar{C}$ replaces each encryption key with the corresponding decryption one.	$\alpha(P Q) = \alpha(P) \alpha(Q)$
- Given $C \in \text{dom}(\underline{f})$ and $\sigma : \text{Vars} \rightarrow \text{Names} \cup \text{Ciphertexts}$ , $\underline{f}(C\sigma) = \underline{f}(C)\sigma^\#$	$\alpha(!P) = !\alpha(P)$
- $\sigma^\#$ is defined as follows:	
(i) $\sigma^\# : \text{Abstract Vars} \rightarrow \text{Abstract Terms}$	
(ii) $x^\# \in \text{dom}(\sigma^\#) \Leftrightarrow x \in \text{dom}(\sigma)$	
(iii) $\sigma^\#(x^\#) = \begin{cases} a^\# & \text{if } \sigma(x) = a \\ \top & \text{otherwise} \end{cases}$	

### 3.2 Operational Semantics

As for  $\rho$ -spi, we define CR calculus operational semantics in terms of *traces*. Recall that principals do not directly synchronize with each other. Instead, they may receive from the unique channel an arbitrary message known by the environment, which models the Dolev-Yao intruder. Here, instead of symbolic representations of cryptographic terms, the intruder can manipulate challenges and responses but we assume that security labels are respected: the intruder can forge messages with labels less than or equal to  $\text{Taint}$ ; moreover, the intruder can read messages with labels less than or equal to  $\text{Int}$ . In the following, we write  $\rightarrow_a$  and  $\rightarrow_a^*$  to denote one-step and multi-step reduction of abstract processes, respectively. For more detail on the transition relation and the deductive system for the knowledge of the environment, please refer to Appendix B.

## 4 Abstraction from $\rho$ -spi to CR

In this section, we define the abstraction of protocol narrations from the  $\rho$ -spi into the CR calculus. The idea is to abstract away from the specific structure of messages, focusing instead on their role in the authentication task. More precisely, the abstraction tells whether a ciphertext is used as challenge or response. The abstraction is fully automated and, given a  $\rho$ -spi protocol narration, yields a CR protocol that is proved to be a faithful abstraction of the former.

### 4.1 Abstraction Function

The abstraction is defined on  $\rho$ -spi terms and processes and is depicted in Table 5. The abstraction of names and variables is straightforward and amounts to map them to their corresponding CR counterparts. The abstraction of a pair yields the pair composed of the abstractions of each component. Finally, tags are abstracted away.

The abstraction of encryptions performed by trusted principals is parameterized and ruled by a partial function  $f$  from encryptions to abstract challenge-response messages. This function works on syntactic terms. At run time, variables may be replaced by ground terms: the abstraction of the resulting encryption is

ruled by the closure of  $f$ , denoted by  $\underline{f}$ , which behaves as  $f$  apart from abstract variables that are replaced by either abstract names or  $\top$ , depending on whether the corresponding concrete variable is instantiated to a name or a different term, i.e., a ciphertext. This simulates the semantics of the CR calculus, which instantiates variables only to abstract names. (For more detail, please cf. Table 15 in Appendix B.)

---

**Table 6** Encryption Abstraction

---

A partial function  $f : \{\!\{ C \}\!\}_k \mapsto \text{Chal}/\text{Resp}_{N^\#}^{\ell, \ell'}(I^\#, J^\#, M^\#)$  is an *encryption abstraction* if the following conditions hold:

*Format* -  $\text{names}(C) \subseteq \mathcal{ID}$  and  $\text{names}(N^\#) = \text{names}(M^\#) = \emptyset$ ;  
-  $x \in \text{vars}(C)$  if and only if  $x^\# \in \text{vars}(N^\#) \cup \text{vars}(M^\#)$ ;  
-  $C$  does not contain encryptions

*Unique Abstraction* if  $\exists C, C' \in \text{dom}(f)$  s.t.  $C\sigma = C'\sigma'$  with  $\sigma, \sigma' : x \mapsto a \mid \{G\}_k$ , then  $C = C'$  (the closure  $\underline{f}$  of  $f$  is a partial function);

*Encryption* If  $\{\!\{ C \}\!\}_k \in \text{dom}(f)$  then

- $k = k_I^+$  implies  $f(\{\!\{ C \}\!\}_k) \in \{\text{Chal}_{N^\#}^{\ell, \ell'}(J^\#, I^\#, M^\#), \text{Resp}_{N^\#}^{\ell', \ell}(J^\#, I^\#, M^\#)\}$ , with  $l \leq \text{Taint}$ ; notice  $I^\#$  in the receiver place;
  - $k = k_I^-$  implies  $f(\{\!\{ C \}\!\}_k) \in \{\text{Chal}_{N^\#}^{\ell, \ell'}(I^\#, J^\#, M^\#), \text{Resp}_{N^\#}^{\ell', \ell}(I^\#, J^\#, M^\#)\}$ , with  $l \leq \text{Int}$ ; notice  $I^\#$  in the sender place;
  - $k = k_{IJ}$  implies  $f(\{\!\{ C \}\!\}_k) \in \{\text{Chal}/\text{Resp}_{N^\#}^{\ell, \ell'}(I^\#, J^\#, M^\#), \text{Chal}/\text{Resp}_{N^\#}^{\ell', \ell}(J^\#, I^\#, M^\#)\}$ ; notice that  $I^\#, J^\#$  correspond to key owners;
- 

Instead of fixing function  $f$ , we let the programmer define, and extend when needed, such a function. The abstraction is proved to be safe as far as  $f$  is an *encryption abstraction*, namely it respects the conditions reported in Table 6 and discussed below:

*Format* The only names occurring in  $C$  are identities;  $N^\#$  and  $M^\#$  are abstract terms only composed of the (abstraction of) variables occurring in  $C$ . It is important that abstractions preserve all of the encrypted variables so that, when decryption is performed, we can recover those (abstract) values from the corresponding challenge-response. For the sake of simplicity, the definition of encryption abstraction does not cover ciphertexts with nested encryptions and throughout this paper we only consider processes where nested encryptions do not occur. The extension to nested encryptions is addressed in Appendix E.

*Unique Abstraction* We require that, when  $f$  is closed by substitution of variables to names and ciphertexts, it remains a function. This means that given a ground run-time message  $G$ , it has a unique possible abstraction  $\underline{f}(G)$ ; without this constraint, abstraction  $\alpha$  in Table 5 would not be a function. This condition may be easily verified by applying a standard most general unifier.

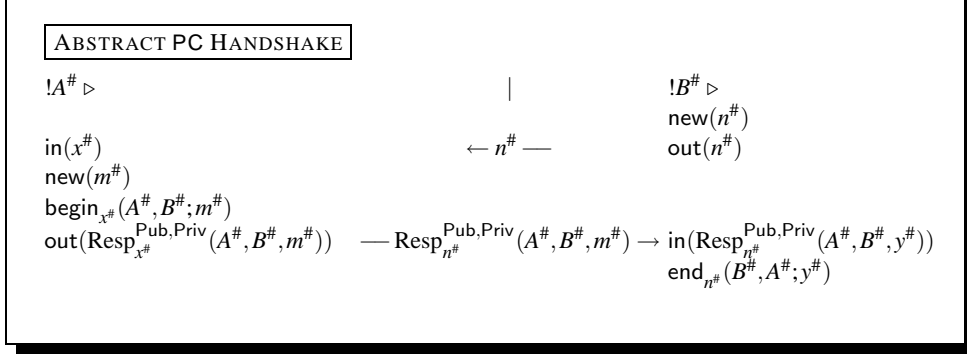
*Encryption* We require that encrypted messages are abstracted at most at the same security level of the concrete messages. Public key encryption is at most  $\text{Taint}$ , digital signature is at most  $\text{Int}$ , symmetric key has no constraints as it is at the higher level ( $\text{Priv}$ ). Identities are placed in the correct positions: a public key specifies the intender receiver, a signature proves the identity of the sender and a symmetric key can be only used by the respective owners. Notice that it is allowed to abstract a message to a lower level of security. In fact, this is not a problem for the soundness of the abstraction but could, on the other hand, make the abstract protocol insecure even if the concrete one is safe, thus losing precision in the abstraction.

Finally, the abstraction of  $\rho$ -spi traces and processes simply amounts to abstract every term occurring therein. Notice that terms of the form  $\{\!\{ C \}\!\}_k$  occurring in input actually specify decryptions: since the abstraction works on encryptions, we substitute each decryption key with the corresponding encryption one. The full definition is given in Table 17 of Appendix C.

*Example 3.* To illustrate, let us consider again the protocol of Example 1. Let us define the encryption abstraction as

$$f = \{\!\{ B, z, x \}\!\}_{k_{AB}} \mapsto \text{Resp}_{x^\#}^{\text{Pub,Priv}}(A^\#, B^\#, z^\#)$$

The abstract narration resulting from the protocol abstraction is depicted below.

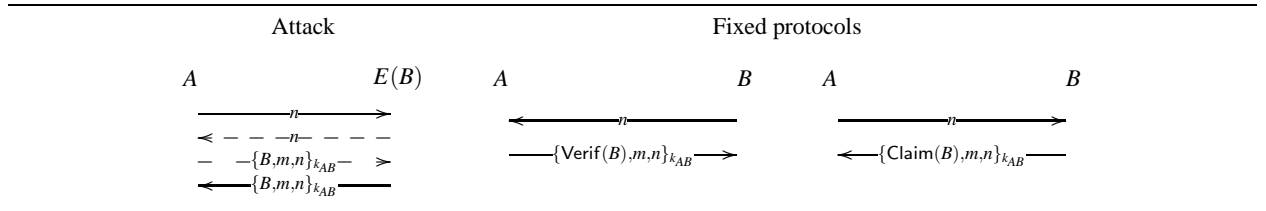


This process is the same as the one in Example 2 apart from the nonce sent in clear that is abstracted in  $n^\#$  instead of  $\text{Chal}_{n^\#}^{\text{Pub,Priv}}(B^\#, A^\#)$ . Indeed, the name does not contain enough information for determining whether it is a challenge or a response. This issue is addressed in Section 5.1. Suppose now that we want to abstract a second protocol, similar to the previous one but with the claimant instead of the verifier's identifier inside the ciphertext. The protocol and the abstraction function for the ciphertext is reported below:

$A$ $\text{--- } n \rightarrow$ $\leftarrow \{B, m, n\}_{k_{AB}} \text{ ---}$	$B$ $\text{--- } n \rightarrow$ $\leftarrow \{B, m, n\}_{k_{AB}} \text{ ---}$	$g = \{\!\{ B, z, x \}\!\}_{k_{AB}} \mapsto \text{Resp}_{x^\#}^{\text{Pub,Priv}}(B^\#, A^\#, z^\#)$
---	---	---

Unfortunately,  $f$  and  $g$  cannot be combined together as they map the same encryption to two different abstract messages:  $f(\{\!\{ B, z, x \}\!\}_{k_{AB}}) = \text{Resp}_{x^\#}^{\text{Pub,Priv}}(A^\#, B^\#, z^\#)$  and  $g(\{\!\{ B, z, x \}\!\}_{k_{AB}}) = \text{Resp}_{x^\#}^{\text{Pub,Priv}}(B^\#, A^\#, z^\#)$ , thus violating property *Unique Abstraction* in Table 6. As a matter of fact, the two protocols are safe as far as they are not put in parallel composition. Indeed, their concurrent execution gives rise to the following standard *reflection* attack depicted on the left side of Table 7.  $A$  is running both the first ( $\text{---}$ ) and the second ( $\text{---}$ ) protocol, as responder and initiator respectively, and, at the end of the session, she authenticates  $B$  even if  $B$  is not present at all in the protocol session. The problem is that  $A$  generates in the first protocol a ciphertext that the enemy may reply to  $A$ , by impersonating  $B$  running the second protocol. The problem can be fixed, and the attack prevented, by the use of tags telling the claimant from the verifier, as shown in Table 7.

**Table 7** Attack on the multi-protocol system and fix



The use of tags makes the two ciphertexts syntactically different and, consequently, the union of the two “fixed” functions  $f_{fix}$  and  $g_{fix}$  below

$$\begin{aligned}
f_{fix} &= \{\text{Verif}(B), z, x\}_{k_{AB}} \mapsto \text{Resp}_{x^\#}^{\text{Pub,Priv}}(A^\#, B^\#, z^\#) \\
g_{fix} &= \{\text{Claim}(B), z, x\}_{k_{AB}} \mapsto \text{Resp}_{x^\#}^{\text{Pub,Priv}}(B^\#, A^\#, z^\#)
\end{aligned}$$

is still a function and, in fact, it is an encryption abstraction.

## 4.2 Soundness Result

The main result states that every concrete computation has a direct counterpart in the abstract model. Indeed, we abstract away from the structure of terms, while every reduction rule in the concrete semantics has a representative in the abstract one.

**Theorem 1 (Soundness).** *If  $\langle s, P \rangle \rightarrow^* \langle s', P' \rangle$ , then  $\langle \alpha(s), \alpha(P) \rangle \rightarrow_a^* \langle \alpha(s'), \alpha(P') \rangle$ .*

## 5 Effect System

In this section, we introduce an effect system for determining the safety of CR processes. The analysis is modular as the safety of a CR protocol can be determined by checking the code of each principal individually and this can be achieved in an independent manner. Furthermore, the analysis is also compositional in that the parallel composition of two safe protocols is always safe. Our use of effects for locally checking the correct behavior of principals is inspired from [8]. Superseding [8] however, we do not need any typing environment as CR syntax makes explicit the role of terms, which considerably simplifies the analysis. The effect system checks the safety of nonce handshakes by relying on the following intuitive principles:

*Verifier's side* The verifier should assert an end assertion only after the successful completion of a suitable nonce handshake based on a fresh nonce.

*Claimant's side* The claimant should respond to a received challenge only after the assertion of a suitable begin assertion.

### 5.1 Prevalidation

This kind of reasoning appears conceptually simple and elegant but requires to solve some technical issues first: (i) nonces should be used only as subscripts of challenge-response components as they might otherwise be unintentionally leaked; and (ii) nonces sent or received in clear on the network do not contain enough information for predicting whether they represent challenges or responses. We solve both of these problems by relying on a processing of CR protocol descriptions, reported in Table 8. The idea is to inspect the code for (i) identifying nonces and messages used in the correspondence assertions and checking that they are distinct and (ii) replacing every occurrence of a nonce as plaintext (i.e., not bound in a challenge or response component) in input or output with either  $\text{Chal}_{N^\#}^{\text{Pub}, \bullet}(A^\#, I^\#)$  or  $\text{Resp}_{N^\#}^{\bullet, \text{Pub}}(A^\#, I^\#)$ . The new symbol  $\bullet$  constitutes a place-holder for an arbitrary security level. Indeed, a Pub challenge does not force the security level of the response, which might be either Priv or Int. Similarly, a Pub response does not force the security level of the challenge, which might be either Priv or Taint. Notice that an inspection of the begin-end assertions in the abstract process suffices for determining the principals running the handshake and for classifying a nonce as challenge or response. For instance, the output of a nonce  $n^\#$  used in  $\text{end}_{n^\#}(A^\#, B^\#, M^\#)$  represents a challenge from  $A^\#$  to  $B^\#$ . Finally, we avoid undesired failures in the first part of the algorithm due to clashes in bound names/variables by  $\alpha$ -converting processes to make all of them distinct.

**Table 8** Prevalidation algorithm

$preval(P^\#)$  behaves as follows:

- check that  $nonces(P^\#) \cap msgs(P^\#) = \emptyset$ , otherwise fail.
- for every  $begin_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#) \in assertions(P^\#)$  replaces
  - each occurrence of  $N^\#$  as plaintext in input (resp. output) with  $Chal_{N^\#}^{Pub, \bullet}(I^\#, A^\#)$  (resp.  $Resp_{N^\#}^{\bullet, Pub}(A^\#, I^\#)$ )
- for every  $end_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#) \in assertions(P^\#)$  label
  - each occurrence of  $N^\#$  as plaintext in output (resp. input) with  $Chal_{N^\#}^{Pub, \bullet}(A^\#, I^\#)$  (resp.  $Resp_{N^\#}^{\bullet, Pub}(I^\#, A^\#)$ )

Notice that: -  $assertions(P^\#)$  yields the multiset of begin-end assertions in  $P^\#$

-  $nonces(P^\#) = \{N^\# \mid begin_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#) \text{ or } end_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#) \in assertions(P^\#)\}$

-  $msgs(P^\#) = \{v^\# \mid begin_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#) \text{ or } end_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#) \in assertions(P^\#) \text{ and } v^\# \in terms(M_1^\#, M_2^\#)\}$ ,  
where  $v^\#$  ranges over CR names and variables.

*Example 4.* Let us consider the CR process discussed in Example 3, which we call  $PC$ . We report below the process yielded by the prevalidation algorithm, namely  $preval(PC)$ .

$$preval(PC) \triangleq !A^\# \triangleright in(Chal_{x^\#}^{Pub, \bullet}(B^\#, A^\#)).new(n^\#).begin_{x^\#}(A^\#, B^\#, m^\#).out(Resp_{x^\#}^{Pub, Priv}(A^\#, B^\#, m^\#)) \\ | !B^\# \triangleright new(n^\#).out(Chal_{n^\#}^{Pub, \bullet}(B^\#, A^\#)).in(Resp_{n^\#}^{Pub, Priv}(A^\#, B^\#, y^\#)).end_{n^\#}(B^\#, A^\#, y^\#)$$

Notice that the nonce  $n^\#$  sent by  $A^\#$  constitutes a challenge from  $B^\#$  to  $A^\#$  and is thus replaced by  $Chal_{n^\#}^{Pub, \bullet}(B^\#, A^\#)$ . The same reasoning applies to  $B^\#$ 's code where  $n^\#$  is replaced by  $x^\#$ .

Finally, we remark that replacing  $N^\#$  by either  $Chal_{N^\#}^{Pub, \bullet}(B^\#, A^\#)$  or  $Resp_{N^\#}^{\bullet, Pub}(B^\#, A^\#)$  does not affect the semantic behavior of CR processes. This is formalized by the following lemma, where  $P^\#_{\approx}$  denotes the process obtained from  $P^\#$  by replacing some of the plaintexts  $N^\#$  occurring in input or output with messages of the form  $Chal_{N^\#}^{Pub, \bullet}(I^\#, J^\#)$  or  $Resp_{N^\#}^{\bullet, Pub}(I^\#, J^\#)$ ; the same manipulation on the abstract trace  $s^\#$  is denoted by  $s^\#_{\approx}$ .

**Lemma 1 (Equivalence).** *If  $\langle s^\#, P^\# \rangle \rightarrow \langle s'^\#, P'^\# \rangle$ , then  $\langle s^\#_{\approx}, P^\#_{\approx} \rangle \rightarrow \langle s'^\#_{\approx}, P'^\#_{\approx} \rangle$*

*Proof (Sketch).* The proof relies on the fact that  $N^\#$ ,  $Chal_{N^\#}^{Pub, \bullet}(I^\#, J^\#)$  and  $Resp_{N^\#}^{\bullet, Pub}(I^\#, J^\#)$  are equivalent from the environment's point of view. Formally, they can be deduced from each other by rule FORGEABLE and READABLE, cf. Table 16 in Appendix B containing the CR manipulation rules of the environment.

*Remark 1 (Process Well-Formedness).* In the following, we shall only consider abstract processes in which nonces do not occur as plaintexts or authenticated messages, namely  $nonces(P^\#) \cap (msgs(P^\#) \cup plaintexts(P^\#)) = \emptyset$ , where  $plaintexts(P^\#) = \{N^\# \mid in(\dots, N^\#, \dots) \in P^\# \text{ or } out(\dots, N^\#, \dots) \in P^\#\}$ .

## 5.2 Effects and their Usefulness for Proving Safety

**Table 9** Effects

$f ::= fresh(n^\#)$	<i>(fresh nonce)</i>	
$[! ?]Chal_{N^\#}^{\ell, \ell'}(I^\#, J^\#, M^\#)$	<i>(challenge)</i>	if $\ell = Pub$ , then $M^\# = \varepsilon$
$[! ?]Resp_{N^\#}^{\ell, \ell'}(I^\#, J^\#, M^\#)$	<i>(response)</i>	
if $\ell = Pub$ , then $M^\# = \varepsilon$		
	$e ::= [f_1, \dots, f_n]$	<i>(effect)</i>
$e ::= [f_1, \dots, f_n]$	<i>(effect)</i>	

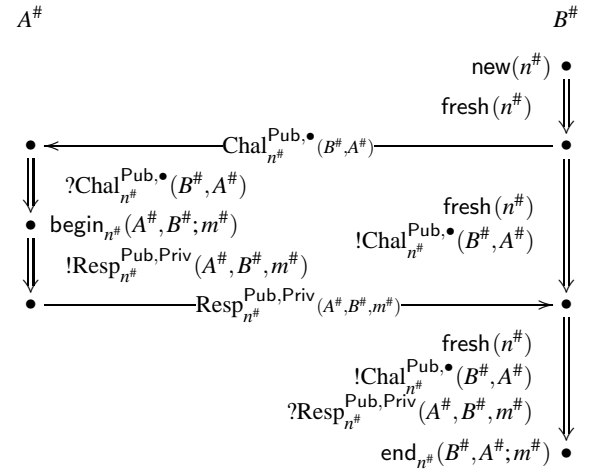
The link between correspondence assertions and the nonce handshake followed by principals can be achieved by tracking the generation-reception of challenges and responses and the freshness of nonces. Effects are used for tracking the freshness of nonces and the generation-reception of challenges and responses. This tracking can be conducted by means of *effects*. Formally, effects are multisets of atomic effects; their syntax is reported in Table 9. The semantics of each atomic effect is as follows:

- The atomic effect  $\text{fresh}(n^\#)$  tracks the freshness of nonce  $n^\#$ , allowing the occurrence of a successive  $\text{end}_{n^\#}(\dots)$  assertion on the same nonce. Indeed, a new name is fresh until it is used for justifying an  $\text{end}_{n^\#}(\dots)$  assertion.
- The atomic effect  $?\text{Chal}_{N^\#}^{\ell,\ell'}(I^\#,J^\#,M^\#)$  (resp.  $!\text{Chal}_{N^\#}^{\ell,\ell'}(I^\#,J^\#,M^\#)$ ) tracks the reception (resp. generation) of a challenge with nonce  $N^\#$  sent by  $I^\#$  to  $J^\#$  for authenticating  $M^\#$ , thus allowing the occurrence of a successive  $\text{begin}_{N^\#}(J^\#,I^\#,M^\#;\cdot)$  (resp.  $\text{end}_{N^\#}(I^\#,J^\#,M^\#;\cdot)$ ) assertion. The security levels  $\ell$  and  $\ell'$  have the same semantics as in Section 3.
- The atomic effect  $?\text{Resp}_{N^\#}^{\ell,\ell'}(I^\#,J^\#,M^\#)$  tracks the reception of a response with nonce  $N^\#$  sent by  $I^\#$  to  $J^\#$  for authenticating  $M^\#$ , thus allowing the occurrence of a successive  $\text{end}_{N^\#}(J^\#,I^\#;\cdot;M^\#)$  assertion.
- The atomic effect  $!\text{Resp}_{N^\#}^{\ell,\ell'}(I^\#,J^\#,M^\#)$  has different semantics as it *enables*  $I^\#$  to generate a response for  $J^\#$  with nonce  $N^\#$  for authenticating  $M^\#$ . This atomic effect is justified by a  $\text{begin}_{N^\#}(I^\#,J^\#;\cdot;M^\#)$  assertion. This asymmetry in the use of effects is discussed below.

**Effects for the PC handshake** For giving the intuition on how effects are used for validating processes, let us consider the abstract protocol of Example 4. The effects for the protocol are depicted in Table 10.

$B^\#$  generates a nonce  $n^\#$ , whose freshness is tracked by the atomic effect  $\text{fresh}(n^\#)$ , and sends it in a challenge to  $A^\#$ :  $!\text{Chal}_{n^\#}^{\text{Pub},\bullet}(B^\#,A^\#)$  tracks on the verifier's code the generation of the challenge. The reception of this message is tracked by  $?\text{Chal}_{n^\#}^{\text{Pub},\bullet}(B^\#,A^\#)$ . The following  $\text{begin}_{n^\#}(A^\#,B^\#,m^\#)$  requires the reception of a challenge generated by  $A^\#$  (effect  $?\text{Chal}_{n^\#}^{\text{Pub},\bullet}(B^\#,A^\#)$ ) and justifies the generation of a response for authenticating  $m^\#$  (effect  $!\text{Resp}_{n^\#}^{\text{Pub,Priv}}(A^\#,B^\#,m^\#)$ ). This is the reason why effects of the form  $!\text{Resp}_{N^\#}^{\ell,\ell'}(\dots)$  are used for *enabling* rather than tracking the generation of responses, as mentioned above. Hence  $!\text{Resp}_{n^\#}^{\text{Pub,Priv}}(A^\#,B^\#,m^\#)$  enables  $A^\#$  to generate the corresponding response, which is eventually received by  $B^\#$ , and is tracked by  $?\text{Resp}_{n^\#}^{\text{Pub,Priv}}(A^\#,B^\#,m^\#)$ . Since the nonce handshake has been successfully completed (effects  $!\text{Chal}_{n^\#}^{\text{Pub},\bullet}(B^\#,A^\#)$  and  $?\text{Resp}_{n^\#}^{\text{Pub,Priv}}(A^\#,B^\#,m^\#)$ ) and the nonce is fresh (effect  $\text{fresh}(n^\#)$ ),  $B^\#$  can safely assert  $\text{end}_{n^\#}(B^\#,A^\#,m^\#)$ .

**Table 10** PC handshake with effects



### 5.3 Determining the Safety of Processes

For simplifying the presentation of processes, in the following we introduce some additional conventions. We write  $![f_1, \dots, f_n]$  and  $?[f_1, \dots, f_n]$  to denote  $![f_1, \dots, !f_n]$  and  $[?f_1, \dots, ?f_n]$ , respectively. Furthermore, +

**Table 11** Validation Rules

Terms

$$\begin{array}{lll} \text{ATOM} & \text{CHAL} & \text{RESP} \\ a^\# : ([\ ]; [\ ]) & \text{Chal}_{N^\#}^{\ell_C, \ell'_C} (I^\#, J^\#, M^\#) : ([\text{Chal}_{N^\#}^{\ell_C, \ell'_C} (I^\#, J^\#, M^\#)]; [\ ]) & \text{Resp}_{N^\#}^{\ell_C, \ell'_C} (I^\#, J^\#, M^\#) : ([\ ]; [\text{Resp}_{N^\#}^{\ell_C, \ell'_C} (I^\#, J^\#, M^\#)]) \end{array}$$

$$\begin{array}{c} \text{PAIR} \\ \frac{C^\# : (e_C; e_R) \quad C'^\# : (e'_C; e'_R)}{(C^\#, C'^\#) : (e_C + e'_C; e_R + e'_R)} \end{array}$$

Processes

$$\begin{array}{llllll} \text{NIL} & \text{REPL} & \text{PAR} & \text{ID} & \text{NEW} & \\ \mathbf{0} : [\ ] & \frac{P^\# : [\ ]}{!P^\# : [\ ]} & \frac{P^\# : e_P \quad Q^\# : e_Q}{P^\# | Q^\# : e_P + e_Q} & \frac{P^\# : e}{A^\# \triangleright P^\# : e} & \frac{P^\# : e \quad n^\# \notin \text{bn}(P^\#) \cup \text{names}(e - [\text{fresh}(n^\#)])}{\text{new}(n^\#).P^\# : e - [\text{fresh}(n^\#)]} & \\ & & \text{IN} & & \text{OUT} & \\ & & \frac{P^\# : e + ?e_C + ?e_R \quad C^\# : (e_C; e_R)}{\text{in}(C^\#).P^\# : e} & & \frac{P^\# : e + !e_C \quad C^\# : (e_C; e_R)}{\text{out}(C^\#).P^\# : e + !e_R} & \\ & & \text{BEGIN} & & & \\ & & \frac{\ell_C, \ell_R \diamond \ell'_C, \ell'_R \quad P^\# : e + [! \text{Resp}_{N^\#}^{\ell'_C, \ell'_R} (A^\#, I^\#, M_2^\#)] \quad M_2^\# \text{ ground}}{\text{begin}_{N^\#} (A^\#, I^\#, M_1^\#, M_2^\#).P^\# : e + [? \text{Chal}_{N^\#}^{\ell_C, \ell_R} (I^\#, A^\#, M_1^\#)]} & & & \\ & & \text{END} & & & \\ & & \frac{\ell_C, \ell_R \diamond \ell'_C, \ell'_R \quad P^\# : e \quad M_1^\# \text{ ground}}{\text{end}_{n^\#} (A^\#, I^\#, M_1^\#, M_2^\#).P^\# : e + [! \text{Chal}_{n^\#}^{\ell_C, \ell_R} (A^\#, I^\#, M_1^\#), ? \text{Resp}_{n^\#}^{\ell'_C, \ell'_R} (I^\#, A^\#, M_2^\#), \text{fresh}(n^\#)]} & & & \end{array}$$

Notice that: - An abstract term is ground if it contains neither variables nor  $\top$ -  $\ell_C, \ell_R \diamond \ell'_C, \ell'_R$  holds if:

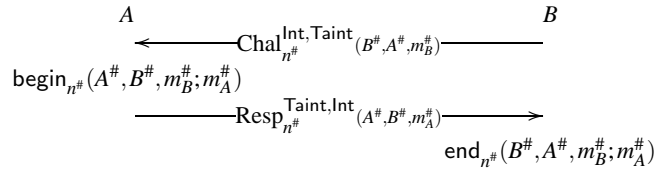
- (i) either  $(\ell_C = \ell'_C \neq \bullet \text{ and } \ell_R = \ell'_R \neq \bullet)$  or  $(\text{Pub} = \ell_C = \ell'_C \text{ and } \ell_R = \bullet)$  or  $(\ell'_C = \bullet \text{ and } \ell_R = \ell'_R = \text{Pub})$ ; and
- (ii)  $\ell_C \in \{\text{Taint}, \text{Priv}\}$  or  $\ell'_R \in \{\text{Int}, \text{Priv}\}$ ; and
- (iii)  $\ell_C = \text{Taint}$  implies  $\ell'_R \neq \text{Int}$

and  $-$  are the usual union and subtraction operators on multi-sets:  $e_1 + e_2$  yields the effect composed of all the atomic effects in  $e_1$  plus the ones in  $e_2$ , while  $e_1 - e_2$  yields the effect obtained by removing, if present, an occurrence of each atomic effect in  $e_2$  from  $e_1$ . If an atomic effect of  $e_2$  does not occur in  $e_1$  then the subtraction of that atomic effect leaves  $e_1$  unchanged.

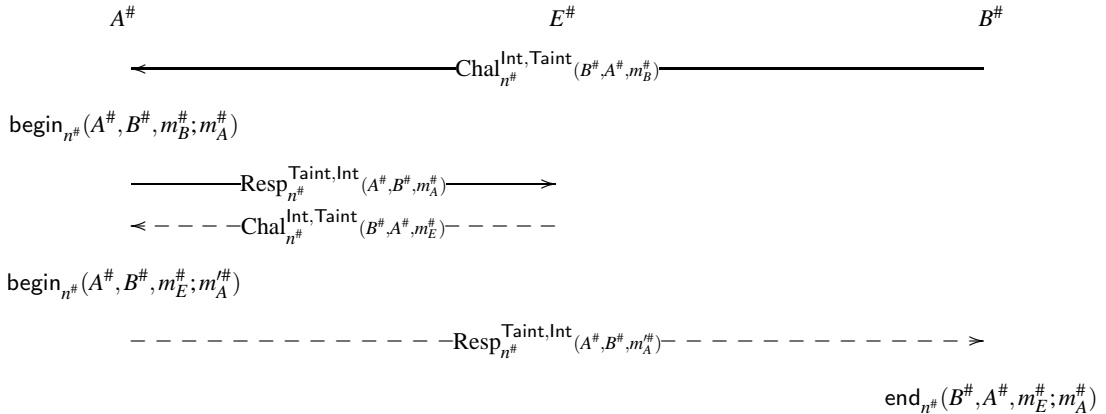
The binding between abstract messages and their effects is formalized in Table 11 by the judgement  $M^\# : (e_C; e_R)$ , read as “ $M^\#$  has challenge effect  $e_C$  and response effect  $e_R$ ”. The main judgement used in our analysis is  $P^\# : e$ , read as “ $P^\#$  has effect  $e$ ”, meaning that process  $P^\#$  is safe under the conditions expressed by effect  $e$ . For instance,  $P^\# : [\text{fresh}(n^\#)]$  means that  $P^\#$  is safe if  $n^\#$  is fresh. In the following we shall give informal explanations on the process judgements of Table 11. The validation of a process is defined by induction on its structure and the null process is the base case: Since each validation rule univocally determines an effect from the one of the continuation process, the effect assigned to processes is unique. NIL validates process  $\mathbf{0}$  under empty effect since validating the null process does not require any specific assumption on the process’ behavior. REPL validates the replication of a process under empty effect  $[\ ]$ , if that process is in turn validated under empty effect. Requiring the effect to be empty is necessary in order to preserve the safety, e.g. replicating a process with effect  $\text{fresh}(n^\#)$  may generate an infinite number

of processes exploiting the freshness of the same nonce  $n^\#$  to complete authentication sessions; this, of course, is not safe as a nonce should be used only once. PAR validates the parallel composition of two processes under the union of their effects, stating that the parallel composition of two processes is safe if both of the processes are safe. ID skips identity declarations as they are not relevant in the analysis. NEW justifies, through the atomic effect  $\text{fresh}(n^\#)$ , at most one use of  $n^\#$  as fresh nonce in the continuation process. Indeed, the deletion of one occurrence of  $\text{fresh}(n^\#)$  in the thesis allows the continuation process to exploit the nonce for asserting one end assertion (cf. rule END). IN justifies in the continuation process the reception of the challenges and responses composing the received message. OUT justifies in the continuation process the reception of the challenges and requires the permission to generate the responses composing the message sent on the network. As discussed in Section 5.2, the assertion  $\text{begin}_{N^\#}(A^\#, I^\#, M_1^\#; M_2^\#)$  (rule BEGIN) requires the reception of  $\text{Chal}_{N^\#}^{\ell_C, \ell_R}(I^\#, A^\#, M_1^\#)$  and justifies the generation of  $\text{Resp}_{N^\#}^{\ell'_C, \ell'_R}(A^\#, I^\#, M_2^\#)$ . Similarly, the assertion  $\text{end}_{n^\#}(A^\#, I^\#, M_1^\#; M_2^\#)$  (rule END) requires the freshness of  $n^\#$ , the generation of  $\text{Chal}_{n^\#}^{\ell_C, \ell_R}(A^\#, I^\#, M_1^\#)$  and the reception of  $\text{Resp}_{n^\#}^{\ell'_C, \ell'_R}(I^\#, A^\#, M_2^\#)$ . Notice that we check on the syntax of begin and end assertions that messages sent in the response by the claimant are ground and so are the ones sent in a challenge by the verifier. This suffices for proving that  $\top$  never occurs within authenticated messages at run time. This is crucial for carrying over safety properties from the abstract to the concrete semantics since  $\top$  might otherwise be instantiated differently in the matching begin and end assertions. Finally, the relation  $\ell_C, \ell_R \diamond \ell'_C, \ell'_R$  checks that the security levels of challenges and responses are consistent and safe, namely either the challenge security level is in  $\{\text{Taint}, \text{Priv}\}$  or the response security level is in  $\{\text{Int}, \text{Priv}\}$ , as discussed in Section 3.

*Remark 2.* Notice that we prevent the combination of tainted challenges with integer responses. The reason is that such handshakes are flawed with respect to our authentication property. To illustrate, let us consider the following CR protocol:



$B^\#$  sends to  $A^\#$  a tainted challenge with nonce  $n^\#$  and message  $m_B^\#$  and  $B^\#$  answers by an integer response with message  $m_A^\#$ . At a first sight, this protocol seems safe as  $A^\#$  proves her identity by decrypting the tainted challenge and signing the response. Unfortunately, the protocol is affected by the following attack:



$A^\#$  engages in two protocol sessions: the former ( $\rightarrow$ ) is started by  $B^\#$  by sending a challenge with nonce  $n^\#$  and message  $m_B^\#$ , the latter ( $\dashrightarrow$ ) is initiated by the enemy impersonating  $B^\#$ . Indeed, the response by  $A^\#$  is integer, thus readable:  $E^\#$  gets the knowledge of the nonce  $n^\#$  and can send it back to  $A^\#$  in a tainted challenge with the new message  $m_E$ . Hence  $A^\#$  outputs an integer response to  $B^\#$  for authenticating the new message  $m_A^\#$ . Eventually,  $B^\#$  authenticates  $A^\#$  receiving message  $m_A^\#$  and sending message  $m_A^\#$  in the same protocol session. Unfortunately, this happened in two different protocol sessions and thus the correspondence assertions do not match. If we considered a weaker authentication property where challenge and response messages are authenticated separately, namely splitting  $\text{begin}_{n^\#}(A^\#, B^\#, m_B^\#, m_A^\#)$  into the two assertions  $\text{begin}_{n^\#}(A^\#, B^\#, m_B^\#)$  and  $\text{begin}_{n^\#}(A^\#, B^\#, m_A^\#)$  and similarly for the end assertion, we would obtain a safe execution trace.

## 5.4 Safety Result

The following theorem states that CR processes with empty effect are safe.

**Theorem 2 (Effect Safety).** *If  $P^\# : []$ , then  $P^\#$  is safe.*

The following theorem constitutes the main result of our framework: if the abstraction of a process is safe, then such a process is safe as well, i.e., a proof of authentication in the CR calculus automatically entails authentication in the  $\rho$ -spi calculus. The proof follows from the previous theorems.

**Theorem 3 (Safety).** *If  $\text{preval}(\alpha(P)) : []$ , then  $P$  is safe.*

*Proof (Sketch).* Assume by contradiction that  $P$  is not safe. Then there exists an unsafe trace  $s \in \text{traces}(P)$ . By Theorem 1,  $\alpha(s) \in \text{traces}(\alpha(P))$ . By Theorem 2,  $\alpha(s)$  is safe. Thus also  $\alpha(s)_\approx$  is safe. By an inspection of rules BEGIN and END, begin-end assertions in  $\alpha(s)_\approx$  do not contain the top element. Thus  $s$  is safe, giving a contradiction.  $\square$

Finally, the effect system is modular and the analysis compositional, as stated by the following theorem.

**Theorem 4 (Modularity and Compositionality).** *Let  $P^\#$  be an abstract process of the form  $!P_1^\# | \dots | !P_m^\#$ . Then  $P^\# : []$  if and only if  $!P_i^\# : [], \forall i \in [1, m]$ .*

Depending on whether one reads the process  $P^\#$  as a protocol, executed by different principals, or a multi-protocol system, composed of different protocols, this theorem states that (i) a protocol is safe if all participants are successfully, and independently, checked (*modularity*), and (ii) a multi-protocol system composed of safe protocols is safe (*compositionality*).

## 6 Conclusion

We have presented a novel approach for concisely abstracting authentication protocols and for subsequently analyzing those abstractions in a sound manner, i.e., deriving authentication guarantees for protocol abstractions suffices for proving these guarantees for the actual protocols. The approach enjoys compositionality guarantees, allows for a modular analysis of abstractions, and ensures that many authentication protocols share a common abstraction so that validating it entails security guarantees for all these protocols and all

compositions thereof. Moreover, such a validation can be performed automatically using static analysis techniques based on the effect system proposed in this paper. We remark that our approach is extensible in that extensions to abstractions of additional protocol classes immediately enjoy a soundness theorem provided that these extensions satisfy the conditions on the encryption abstraction. Finally, we tested our technique on several existing protocols; some examples are in Appendix E.

As future work, we plan to further reduce the complexity of the proofs conducted within our abstraction and enhance its expressiveness by an abstract interpretation suitably reflecting different protocol sessions into one abstract session so that authentication becomes directly decidable in the abstract domain, without the need of further approximating it as done in the effect system presented in this paper. Furthermore, we plan to exploit recent results on linking symbolic cryptography with actual cryptographic algorithms to verify truly abstract authentication protocols in a way that ensures strong authentication guarantees even for concrete, cryptographic implementations.

## References

1. M. Abadi. Secrecy by typing in security protocols. *JACM*, 46(5):749–786, 1999.
2. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theoretical Computer Science*, 298(3):387–415, 2003.
3. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
4. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 126–140. IEEE Computer Society Press, June 2003.
5. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis can find new flaws too. In *Proceedings of the Workshop on Issues on the Theory of Security (WITS'04)*. Elsevier, 2004.
6. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 01*, volume 2076, pages 667–681. LNCS 2076, Springer Verlag, 2001.
7. M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Logic in Computer Science*, pages 157–166, 1999.
8. M. Bugliesi, R. Focardi, and M. Maffei. Dynamic types for authentication. To appear in *Journal of Computer Security*.
9. M. Bugliesi, R. Focardi, and M. Maffei. Principles for entity authentication. In *Proceedings of 5th International Conference Perspectives of System Informatics (PSI 2003)*, volume 2890 of *Lecture Notes in Computer Science*, pages 294–307. Springer-Verlag, July 2003.
10. M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer-Verlag, 2004.
11. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
12. N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):677–721, 2004.
13. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *Journal of Computer Security*, 11(4):451–519, 2004.
14. A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3/4):435–484, 2004.
15. G. Lowe. “A Hierarchy of Authentication Specification”. In *Proceedings of the 10th Computer Security Foundation Workshop (CSFW'97)*, pages 31–44. IEEE Computer Society Press, 1997.
16. M. Maffei. *Dynamic Typing of Security Protocols*. PhD thesis, Department of Computer Science, Ca' Foscari University of Venice, 2006.
17. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.
18. T.Y.C. Woo and S.S. Lam. “A Semantic Model for Authentication Protocols”. In *Proceedings of 1993 IEEE Symposium on Security and Privacy*, pages 178–194, 1993.

## A Calculi Well-Formedness

**$\rho$ -spi calculus** The  $\rho$ -spi syntax is quite liberal, giving the possibility of writing nonsense processes like  $A \triangleright B \triangleright P$ , where an identity declaration nests within another one, or  $A \triangleright \text{begin}_n(B, I, M_1, M_2)$ , where  $A$  asserts a begin-event in place of  $B$ . Moreover, a shared key  $k_{IJ}$  should only be used by  $I$  and  $J$  and a private key  $k_I^-$  should only be used by  $I$ . Process well-formedness rules out the above mentioned undesired process behaviors and enforces the correct use of long-term keys. It is formalized through the standard notion of scope: we say that the *scope* of  $A \triangleright$  in process  $A \triangleright P$ , is process  $P$ .

**Definition 3 (Process Well-Formedness).** *A process  $P$  is well-formed if the following conditions hold:*

1. *All the occurrences of  $A \triangleright$  in  $P$  are not in the scope of some other  $B \triangleright$ ;*
2. *Every  $\text{begin}_N(A, I, M_1, M_2)$  in  $P$  is in the scope of  $A \triangleright$ ;*
3. *Every  $\text{end}_N(B, J, M_1, M_2)$  in  $P$  is in the scope of  $B \triangleright$ ;*
4. *Every symmetric key  $k_{IJ}$  occurring in  $P$  is in the scope of either  $I \triangleright$  or  $J \triangleright$ ;*
5. *Every private key  $k_I^-$  occurring in  $P$  is in the scope of  $I \triangleright$ .*

Well-formedness is trivially verifiable by a simple syntactic inspection.

**CR calculus** As for  $\rho$ -spi calculus, we have a notion of process well-formedness that rules out undesired process behaviors and enforces the correct use of challenges and responses. In the following, we shall always consider well-formed processes.

**Definition 4 (Abstract Process Well-Formedness).** *A CR process  $P^\#$  is well-formed if the following conditions hold:*

1. *All the occurrences of  $A^\# \triangleright$  in  $P^\#$  are not in the scope of some other  $B^\# \triangleright$ ;*
2. *Every  $\text{begin}_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#)$  in  $P^\#$  is in the scope of  $A^\# \triangleright$ ;*
3. *Every  $\text{end}_{N^\#}(B^\#, J^\#, M_1^\#, M_2^\#)$  in  $P^\#$  is in the scope of  $B^\# \triangleright$ ;*
4. *Every  $\text{out}(\dots, \text{Chal/Resp}_{N^\#}^{(\ell, \ell')}(I^\#, J^\#, M^\#), \dots)$  in  $P$ , with  $\ell \geq \text{Int}$ , is in the scope of  $I^\#$ .*
5. *Every  $\text{in}(\dots, \text{Chal/Resp}_{N^\#}^{(\ell, \ell')}(I^\#, J^\#, M^\#), \dots)$  in  $P$ , with  $\ell' \geq \text{Taint}$ , is in the scope of  $J^\#$ .*

Notice that the notion of well-formedness includes consistency checks between challenges and responses generated and received by a principal and their security level. Specifically, only  $I^\#$  can generate  $\text{Chal/Resp}_{N^\#}^{(\ell, \ell')}(I^\#, J^\#, M^\#)$ , if  $\ell \geq \text{Int}$  (item 4), and only  $J^\#$  may read the content of  $\text{Chal/Resp}_{N^\#}^{(\ell, \ell')}(I^\#, J^\#, M^\#)$ , if  $\ell' \geq \text{Taint}$  (item 5). As for  $\rho$ -spi, well-formedness conditions are trivially verifiable by a simple syntactic inspection.

## B Calculi Semantics

**$\rho$ -spi calculus semantics** The dynamics of  $\rho$ -spi is formalized by means of a transition relation between *configurations*, i.e., pairs  $\langle s, P \rangle$ , where  $s \in \text{Act}^*$  is a trace,  $P$  is a (closed) process. Each transition  $\langle s, P \rangle \vdash \langle s :: \alpha, P' \rangle$  simulates one computation step in  $P$  and records the corresponding action in the trace.

We distinguish between the static terms of the calculus, and the *run-time messages* which are actually sent and received. The latter are ground, i.e., they do not contain variables, and have the same syntax as

**Table 12** Transition System for  $\rho$ -spi

**Transition rules:** We omit the symmetric rule of PAR.

<p>RES</p> $\frac{a \notin \text{bn}(s) \cup \text{fn}(s)}{\langle s, \text{new}(n).P \rangle \rightarrow \langle s :: \text{new}(n), P \rangle}$	<p>INPUT</p> $\frac{s \vdash G \quad \sigma = \text{bind}(C, G) \neq \uparrow}{\langle s, \text{in}(C).P \rangle \rightarrow \langle s :: \text{in}(C\sigma), P\sigma \rangle}$	<p>OUTPUT</p> $\langle s, \text{out}(R).P \rangle \rightarrow \langle s :: \text{out}(R), P \rangle$
<p>BEGIN</p> $\langle s, \text{begin}_G(A, I, G_1; G_2).P \rangle \rightarrow \langle s :: \text{begin}_G(A, I, G_1; G_2), P \rangle$	<p>END</p> $\langle s, \text{end}_G(A, I, G_1; G_2).P \rangle \rightarrow \langle s :: \text{end}_G(A, I, G_1; G_2), P \rangle$	
<p>PRINCIPAL</p> $\frac{\langle s, P \rangle \rightarrow \langle s :: \alpha, P' \rangle}{\langle s, A \triangleright P \rangle \rightarrow \langle s :: A \triangleright \alpha, A \triangleright P' \rangle}$	<p>PAR</p> $\frac{\langle s, P \rangle \rightarrow \langle s', P' \rangle}{\langle s, P   Q \rangle \rightarrow \langle s', P'   Q \rangle}$	<p>REPLICATION</p> $\langle s, !P \rangle \rightarrow \langle s, P \mid !P \rangle$

**Table 13** Binding terms and run-time messages

$\begin{aligned} \text{bind}(a, a) &= [] \\ \text{bind}(x, a) &= [a/x] \\ \text{bind}(x, \{G\}_K) &= [\{G\}_K/x] \\ \text{bind}(\text{Tag}(C), \text{Tag}(G)) &= \text{bind}(M, G) \\ \text{bind}((C, C'), (G, G')) &= \text{bind}(C, G) \uplus \text{bind}(C', G') \\ \text{bind}(\llbracket C \rrbracket_{\bar{k}}, \{G\}_k) &= \text{bind}(C, G) \\ \text{bind}(C, G) &= \uparrow \end{aligned}$	<p>otherwise</p>
$\sigma_1 \uplus \sigma_2 = \sigma_1 \cup \sigma_2$	<p>if <math>x \in \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) \Rightarrow \sigma_1(x) = \sigma_2(x)</math></p>
$\sigma_1 \uplus \sigma_2 = \uparrow$	<p>otherwise</p>

static terms apart from cryptography which is denoted  $\{\dots\}_k$  instead of  $\llbracket \dots \rrbracket_k$ . We will use  $G$  to denote *run-time messages*. At run-time, term variables can be bound to run-time messages producing special terms containing a mixture of  $\{\dots\}_k$  and  $\llbracket \dots \rrbracket_k$ . We call these extended terms *run-time terms*, ranged over by  $R$ . Run-time terms can occur in input and output primitives only, given that calculus syntax forbids the use of cryptography in the other primitives. For this reason, at run-time, begin and end will only contain run-time messages  $G$ . The set of all possible actions, noted  $Act$ , includes the action  $\text{in}(R)$  generated by input,  $\text{out}(R)$  by output,  $\text{begin}_G(A, I, G_1; G_2)$  and  $\text{end}_G(A, I, G_1; G_2)$  by ‘begin’ and ‘end’, and  $\text{new}(n)$  by restriction. Actions are prefixed by  $A \triangleright$  when executed by a principal  $A$ . We show the intuition behind run-time terms through simple examples: Consider action  $\text{in}(\llbracket \{n\}_{k_{AB}} \rrbracket_{k_A^-})$ . It represents the input of message  $\{\{n\}_{k_{AB}}\}_{k_A^+}$  of which only the outermost encryption has been decrypted (notice that the innermost encryption is a run-time message). This may be achieved using primitive  $\text{in}(\llbracket x \rrbracket_{k_A^-})$ , which will bound  $x$  to  $\{n\}_{k_{AB}}$ . Through primitive  $\text{in}(\llbracket \llbracket y \rrbracket_{k_{AB}} \rrbracket_{k_A^-})$ , instead, both encryptions would be decrypted binding  $y$  to  $n$ . Thus, run-time terms allows us to trace what is encrypted or decrypted and what is just sent or received.

Some transitions apply substitutions to processes: formally, a substitution  $\sigma : x \mapsto G$  is a function from variables to run-time messages. Often substitutions are written explicitly by  $[G_1/x_1, \dots, G_n/x_n]$ . The application of the substitution  $\sigma$  to the process  $S$  is denoted by  $P\sigma$  and applies only to free occurrences of the variables in  $P$ . We use a number of notation conventions. The restriction operator  $\text{new}(n).S$  is a binder for name  $n$ , the input primitive is a binder for the variables that occur in term  $C$ . In all cases the scope of the

**Table 14** Message Manipulation Rules

$[R]$  takes a ground run-time term  $R$  and returns the corresponding run-time message, i.e., changes all the occurrences of  $\{\dots\}_k$  into  $\{\dots\}_k$ .

OUT $\frac{out(R) \in s}{s \vdash [R]}$	ENV $\frac{a \notin \text{bn}(s)}{s \vdash a}$	PAIR $\frac{s \vdash G_1 \quad s \vdash G_2}{s \vdash (G_1, G_2)}$	PAIR DES $\frac{s \vdash (G_1, G_2) \quad i \in [1, 2]}{s \vdash G_i}$	TAG $\frac{s \vdash G}{s \vdash \text{Tag}(G)}$	TAG DES $\frac{s \vdash \text{Tag}(G)}{s \vdash G}$
ENCRYPTION $\frac{s \vdash G \quad s \vdash k}{s \vdash \{G\}_k}$	DECRYPTION $\frac{s \vdash \{G\}_k \quad s \vdash \bar{k}}{s \vdash G}$	PUBLIC KEYS $s \vdash k_i^+$	ENEMY KEYS $s \vdash k_{EI} \quad s \vdash k_{IE} \quad s \vdash k_E^-$		

binders is the continuation process. Similarly,  $new(n)$  is a binder for names and its scope is the continuation trace. The notions of free/bound names and variables, both for processes and traces, arise as expected.

Table 12 collects transition rules: RES generates a new name  $n$  by checking that it differs from all the names already used in the trace  $s$ . It is possible to force this condition by applying  $\alpha$ -conversion to  $n$ , i.e., by substituting  $n$  and all of its free occurrences in  $S$  with a different name. Indeed, as in companion transition systems, see, e.g. [7], we implicitly identify processes up to renaming of bound variables and names, i.e., up to  $\alpha$ -equivalence. INPUT requires message  $G$ , read from the network, to be computable by the environment: the environment knowledge is defined by the message manipulation rules reported in Table 14 and discussed below. The run-time message  $G$  is read only if it can be pattern-matched with the input term  $C$  via the function  $bind$  of Table 13 and discussed below. In such a case, all the variables in  $C$  are bound to the respective submessages of  $G$  producing the run-time term  $C\sigma$ . OUTPUT, BEGIN and END are self-explanatory. Finally, PRINCIPAL adds the principal name to the performed action, PAR interleaves two different protocol executions and REPLICATION arbitrarily replicates a principal.

Pattern-matching is formalized through the function  $bind$ , defined in Table 13. We write  $\bar{k}$  to denote the decryption key corresponding to  $k$ . For symmetric cryptography we have  $\bar{k}_{AB} = k_{AB}$ , for asymmetric it holds  $\bar{k}_A^+ = k_A^-$  and  $\bar{k}_A^- = k_A^+$ . Function  $bind$  takes as input a static term  $C$  and a run-time message  $G$  and, in case it exists, yields the substitution  $\sigma$  which makes  $C$  equal to  $G$ , up to the different notation for encryption. If pattern-matching fails,  $bind$  returns  $\uparrow$ . It is defined by cases on the structure of term  $C$ : a name matches a name with empty substitution; a variable can be bound to either an atomic name  $a$  or to a ciphertext; tagged terms require the same tag, and pairs match pairs yielding a substitution which is the union  $\uplus$  of the ones achieved for the subterms; the union  $\sigma_1 \uplus \sigma_2$  succeeds only when variables substituted by both  $\sigma_1$  and  $\sigma_2$  are bound to the same run-time message; finally, decryptions must be performed with the correct decryption key. In all the other cases,  $bind$  returns failure  $\uparrow$ .

The message manipulation rules, in Table 14, formalize the environment actions. Rule OUT says that every message sent on the network is known by the environment. ENV allows the environment to know any name which is not bound (i.e., restricted) in the trace. By PAIR and PAIR DES, the environment may construct and destruct pairs. TAG and TAG DES allow the environment to tag and untag messages. By ENCRYPTION, and DECRYPTION the environment can encrypt and decrypt messages only knowing the required keys. By PUBLIC KEYS, all the public keys are known by the environment. Finally, by ENEMY KEYS, the environment may be provided with its own private keys and with long-term keys shared with honest participants. This gives the possibility to the enemy to start authentication sessions and, generally speaking, to interact with the other participants by pretending to be a trusted principal. As an example, let us consider the following transitions :

$$\begin{aligned}
& \langle \varepsilon, A \triangleright \text{in}(\{x\}_{k_A^-}).\text{out}(x).\mathbf{0} \rangle \\
\rightarrow & \langle A \triangleright \text{in}(\{n\}_{k_A^+}), A \triangleright \text{out}(n).\mathbf{0} \rangle \\
\rightarrow & \langle A \triangleright \text{in}(\{n\}_{k_A^+}) :: A \triangleright \text{out}(n), \mathbf{0} \rangle
\end{aligned}$$

A reads the message  $\{n\}_{k_A^+}$ , which can be generated by the environment using rules ENV, PUBLIC KEYS and ENCRYPTION. Function  $\text{bind}(\{x\}_{k_A^-}, \{n\}_{k_A^+})$  succeeds returning  $\sigma = [n/x]$ . Hence the process performs a  $\text{in}(\{x\}_{k_A^-} \sigma) = \text{in}(\{n\}_{k_A^-})$  moving to  $\text{out}(x)\sigma = \text{out}(n)$ . This action represents a reception with decryption of message  $\{n\}_{k_A^+}$ . Then, the output is performed and the process terminates.

**Table 15** CR calculus: Binding abstract terms and run-time messages

$$\begin{aligned}
\text{bind}^\#(a^\#, a^\#) &= [] \\
\text{bind}^\#(x^\#, a^\#) &= [a^\#/x^\#] \\
\text{bind}^\#((C_1^\#, C_2^\#), (G_1^\#, G_2^\#)) &= \text{bind}^\#(C_1^\#, G_1^\#) \uplus \text{bind}^\#(C_2^\#, G_2^\#) \\
\text{bind}^\#(\text{Chal}_{N^\#}^{(\ell, \ell')}(A^\#, B^\#, M^\#), \text{Chal}_{G_1^\#}^{(\ell, \ell')}(A^\#, B^\#, G_2^\#)) &= \text{bind}^\#(N^\#, G_1^\#) \uplus \text{bind}^\#(M^\#, G_2^\#) \\
\text{bind}^\#(\text{Resp}_{N^\#}^{(\ell, \ell')}(A^\#, B^\#, M^\#), \text{Resp}_{G_1^\#}^{(\ell, \ell')}(A^\#, B^\#, G_2^\#)) &= \text{bind}^\#(N^\#, G_1^\#) \uplus \text{bind}^\#(M^\#, G_2^\#) \\
\text{bind}^\#(C^\#, G^\#) &= \uparrow \quad \text{otherwise}
\end{aligned}$$

**Table 16** CR calculus: Abstract Message Manipulation Rules

$$\begin{array}{c}
\text{OUT} \\
\frac{\text{out}(G^\#) \in s}{s \vdash G^\#} \\
\\
\text{ENV} \\
\frac{a^\# \notin \text{bn}(s)}{s \vdash a^\#} \\
\\
\text{PAIR} \\
\frac{s \vdash G_1^\# \quad s \vdash G_2^\#}{s \vdash (G_1^\#, G_2^\#)} \\
\\
\text{PAIR DES} \\
\frac{s \vdash (G_1^\#, G_2^\#) \quad i \in [1, 2]}{s \vdash G_i^\#} \\
\\
\text{TOP} \\
s \vdash \top \\
\\
\text{FORGEABLE} \\
\frac{s \vdash G_1^\#, G_2^\# \quad \ell \leq \text{Taint} \vee I^\# = E^\# \vee J^\# = E^\#}{s \vdash \text{Chal}_{G_1^\#}^{\ell, \ell'}(I^\#, J^\#, G_2^\#) \quad s \vdash \text{Resp}_{G_1^\#}^{\ell, \ell'}(I^\#, J^\#, G_2^\#)} \\
\\
\text{READABLE} \\
\frac{s \vdash \text{Chal}_{G_1^\#}^{\ell, \ell'}(I^\#, J^\#, G_2^\#) \vee s \vdash \text{Resp}_{G_1^\#}^{\ell, \ell'}(I^\#, J^\#, G_2^\#)}{\ell \leq \text{Int} \vee I^\# = E^\# \vee J^\# = E^\#} \\
\frac{}{s \vdash G_1^\#, G_2^\#}
\end{array}$$

**CR calculus semantics** The semantics of CR calculus is based on the same transition rules of  $\rho$ -spi calculus presented in previous section (Table 12). The only difference is the way terms are bound and deducted by the intruder. Table 15 defines the function  $\text{bind}^\#$  used for the input primitive of CR calculus. Table 16 describes message manipulation rules for abstract messages. Notice that security labels are not violated by the intruder, similarly to what is done with cryptography in the concrete calculus. Here and throughout this paper,  $G^\#$  and  $R^\#$  range over ground abstract messages.

## C Abstraction of Run-Time Terms, Traces and Processes

The abstraction of run-time terms, reported in Table 17, is very similar to the one of syntactic terms presented in Table 5. The main difference is that ciphertexts may occur at run-time as result of variable instantiation.

The abstraction of ciphertexts occurring within processes is  $\top$  as they are proved to be already known by the environment and their input-output does not affect the abstraction. A more precise abstraction of ciphertexts is required for abstracting environment's knowledge, as discussed in the long version of this paper. The abstraction of traces is obtained by abstracting every term occurring therein. The abstraction of processes is similar and is thus omitted from Table 5.

**Table 17** Abstraction of run-time terms

Run Time Terms	Traces
$\alpha(a) = a^\#$	$\alpha(\varepsilon) = \varepsilon$
$\alpha(R_1, R_2) = (\alpha(R_1), \alpha(R_2))$	$\alpha([A \triangleright] \text{new}(n) :: t) = [\alpha(A) \triangleright] \text{new}(\alpha(n)) :: \alpha(t)$
$\alpha(\text{Tag}(R)) = \alpha(R)$	$\alpha([A \triangleright] \text{in}(R) :: t) = [\alpha(A) \triangleright] \text{in}(\alpha(R)) :: \alpha(t)$
$\alpha(\{R\}_k) = \begin{cases} \underline{f}(\{R\}_k) & \text{if } \{R\}_k \in \text{dom}(\underline{f}) \\ \alpha(R) & \text{otherwise} \end{cases}$	$\alpha([A \triangleright] \text{out}(R) :: t) = [\alpha(A) \triangleright] \text{out}(\alpha(R)) :: \alpha(t)$
$\alpha(\{G\}_k) = \top$	$\alpha([A \triangleright] \text{begin}_G(A, I, G_1; G_2) :: t) = [\alpha(A) \triangleright] \text{begin}_{\alpha(G)}(\alpha(A), \alpha(I), \alpha(G_1); \alpha(G_2)) :: \alpha(t)$
	$\alpha([A \triangleright] \text{end}_G(A, I, G_1; G_2) :: t) = [\alpha(A) \triangleright] \text{end}_{\alpha(G)}(\alpha(A), \alpha(I), \alpha(G_1); \alpha(G_2)) :: \alpha(t)$

## D Proofs of Soundness and Safety

In this section we formulate the proofs of the main theorems. In particular, in Section D.1 we prove the soundness of the abstraction and in Section D.2 the safety of the analysis.

### D.1 Soundness of the Abstraction

We use a special abstraction  $\alpha_e(G)$  of run-time messages  $G$ , defined in table 18. As proved by the following lemma, this abstraction is more precise than the usual  $\alpha$ , as we need to consider all the potential challenges and responses, independently from the fact that processes will input them.

**Lemma 2 (Environment Abstraction).**  $s^\# \vdash \alpha_e(\llbracket R \rrbracket)$  implies  $s^\# \vdash \alpha(R)$ .

*Proof.* The proof is by induction on the structure of  $R$ .

**Base Case** The base case is  $R = a$ , i.e.,  $R$  is a name. The proof is immediate as  $\alpha_e(a) = \alpha(a) = a^\#$ .

**Inductive Cases** If  $R = \{G\}_k$ , then  $\alpha(R) = \top$  and  $s^\# \vdash \top$  by TOP; if  $R = (R_1, R_2)$ , then by induction hypothesis  $s^\# \vdash \alpha(R_i)$ , for every  $i \in [1, 2]$ , and, by the message manipulation rule PAIR,  $s^\# \vdash (\alpha(R_1), \alpha(R_2))$ ; the other cases follow immediately by definition of  $\alpha_e$ .

The next lemma states that abstract names occurring as nonces and messages in an abstraction of an encryption are derivable from an abstract trace  $s^\#$  if and only if the abstraction of the ciphertext content is also derivable from  $s^\#$ .

**Lemma 3.** Let  $\{G\}_k \in \text{dom}(\underline{f}_e)$  and  $\alpha_e(\{G\}_k) = \text{Chal/Resp}_{G_1^\#}^{\ell, \ell'}(I^\#, J^\#, G_2^\#)$ . Then  $s^\# \vdash \alpha(G)$  if and only if  $s^\# \vdash G_1^\#, G_2^\#$

---

**Table 18** Abstraction for the environment

---

**Run Time Messages**

$$\begin{aligned}\alpha_e(a) &= a^\# \\ \alpha_e(G_1, G_2) &= (\alpha_e(G_1), \alpha_e(G_2)) \\ \alpha_e(\text{Tag}(G)) &= \alpha_e(G) \\ \alpha_e(\{G\}_k) &= \begin{cases} \underline{f}_e(\{G\}_k) & \text{if } \{G\}_k \in \text{dom}(\underline{f}_e) \\ \alpha_e(G) & \text{otherwise} \end{cases}\end{aligned}$$

where  $\underline{f}_e(\llbracket R \rrbracket) \stackrel{\text{def}}{=} \underline{f}(R)$

---

*Proof.* This result derives from the definition of  $\underline{f}$  and from condition *Format* in Table 6. In fact,  $\underline{f}$  is a closure of  $f$  in which variables on the left (concrete) are bound to names or ciphertexts, and variables on the right (abstract) are bound to names or  $\top$ , respectively. For this reason  $\alpha(G)$  is a tuple of atomic abstract names and  $\top$ 's which also occur in  $G_1^\#, G_2^\#$  and vice-versa.  $\square$

The following definition introduces a notion of well-formedness for  $\rho$ -spi traces, requiring that run-time ciphertexts sent as output, and not generated therein, have been received before. It is easy to see that every trace generated by a  $\rho$ -spi process is well-formed. For this reason, in the following we shall only consider well-formed traces.

**Definition 5 (Trace Well-Formedness).** *A trace  $s$  is well-formed if the following conditions hold:*

- if  $s = s' :: \text{out}(R)$  and  $\{G\}_k \in \text{terms}(R)$ , then  $\exists G'$  s.t.  $\text{in}(G') \in s'$  and  $\{G\}_k \in \text{terms}(G')$  (every ciphertext, which is not generated in the output, has been received before)

The next lemma states that the environment knows every ciphertext, even if nested, circulating on the network.

**Lemma 4 (Nesting and Environment's Knowledge).** *If  $s \vdash G$  and  $\{G'\}_k \in \text{terms}(G)$ , then  $s \vdash \{G'\}_k$*

*Proof.* By induction on the length of  $s$  and on the derivation length of  $s \vdash G$ .

**Base Case**  $s = \varepsilon$  We start by considering an empty trace, which is the base case for the external induction:

**Base Case** We only have one possible derivation of length one, namely *Env*, since *Out* has no effect on an empty trace. The proof is trivial as names, of course, do not contain nested ciphertexts.

**Inductive Cases** The rules for pairs, tags, encryptions and decryptions trivially follow from the induction hypothesis.

**Inductive Case**  $s \neq \varepsilon$  We now consider a non-empty trace  $s$ .

**Base Cases** The only interesting case is *Out*:

**Out** The judgement  $s \vdash \llbracket R \rrbracket$  derives by  $\text{out}(R) \in s$ . Let us suppose that  $\{G\}_k \in \text{terms}(\llbracket R \rrbracket)$ . By *PAIR DES*,  $s$  derives the top-level ciphertexts in  $\llbracket R \rrbracket$ . Let us consider the nested ones: since, by syntactic restriction,  $R$  does not contain nested encryptions, we have that  $\{G\}_k \in \text{terms}(R)$ . Since the trace is well-formed (cf. Definition 5), we have that  $\{G\}_k$  has been previously received as input, possibly nested. The thesis follows from the induction hypothesis on the length of  $s$ .

**Inductive Cases** The proof derives straightforwardly from the induction hypothesis on the derivation length of  $s \vdash G$ .  $\square$

The following proposition states that if a message  $G$  is known to the environment after the trace  $s$ , then  $\alpha_e(G)$  is known after  $\alpha(s)$ .

**Proposition 1 (Knowledge Abstraction).** *If  $s \vdash G$ , then  $\alpha(s) \vdash \alpha_e(G)$ ;*

*Proof.* By induction on the length of  $s$  and on the derivation length of  $s \vdash G$ .

**Base Case  $s = \varepsilon$**  We start by considering an empty trace, which is the base case for the external induction:

**Base Cases** We only have one possible derivation of length one: Env, since Out has no effect on an empty trace:

**Env** for every  $a$  we have  $\varepsilon \vdash a$  and  $\varepsilon \vdash \alpha(a) = a^\#$ . Of course, an empty trace does not have any bound name so every name and every abstract name is derivable from  $\varepsilon$ .

**Inductive Case** Since the rules for pairs and tags trivially follow from the induction hypothesis, we focus on the two relevant cases of encryption and decryption:

**Encryption** In the following, we discuss public-key encryption: the proof for encryptions via enemy-keys is similar. From  $\varepsilon \vdash G$  we may derive  $\varepsilon \vdash \{G\}_{k_1^+}$ . By induction,  $\varepsilon \vdash \alpha_e(G)$ . We now have two cases: if  $\{G\}_{k_1^+} \notin \text{dom}(\underline{f}_e)$  we have that  $\alpha_e(\{G\}_{k_1^+}) = \alpha_e(G)$  thus trivially obtaining the thesis. Otherwise,  $\alpha_e(\{G\}_{k_1^+}) = \underline{f}_e(\{G\}_{k_1^+})$ . By condition *Encryption* of Table 6, we know that  $\underline{f}_e(\{G\}_{k_1^+})$  can be either  $\text{Chal}_{G_1^\#}^{\ell, \ell'}(J^\#, I^\#, G_2^\#)$  or  $\text{Resp}_{G_1^\#}^{\ell', \ell}(J^\#, I^\#, G_2^\#)$ , with  $\ell \leq \text{Taint}$ . Moreover, by Lemma 2, we have that  $\varepsilon \vdash \alpha_e(G)$  implies  $\varepsilon \vdash \alpha(G)$ . By Lemma 3 we obtain that  $\varepsilon \vdash G_1^\#, G_2^\#$ . By rule Forgeable, we obtain that  $\varepsilon \vdash \underline{f}_e(\{G\}_{k_1^+})$ , i.e.,  $\varepsilon \vdash \alpha_e(\{G\}_{k_1^+})$ .

**Decryption** Since, with an empty trace, encryptions can only be generated by the environment itself,  $\varepsilon \vdash \{G\}_k$  has been for sure obtained using the encryption rule by the hypothesis  $\varepsilon \vdash G$ . By applying the induction hypothesis we directly obtain  $\varepsilon \vdash \alpha_e(G)$ .

**Inductive Case  $s \neq \varepsilon$**  We now consider a non-empty trace  $s$ .

**Base Cases** We have two possible derivations of length one, namely Out and Env:

**Out** The judgement  $s \vdash [R]$  derives from  $\text{out}(R) \in s$ . By trace abstraction we know that  $\text{out}(\alpha(R)) \in \alpha(s)$  and, by Out,  $\alpha(s) \vdash \alpha(R)$ . Notice, now, that  $\alpha(R)$  and  $\alpha_e([R])$  may only differ for some  $\{G\}_k \in \text{terms}(R)$  which is abstracted into  $\top$  in  $\alpha(R)$ . However, by Definition 5, a message  $\{G\}_k$  can be present in  $R$  only if it has been previously read from the environment. Let  $s'$  be the prefix of  $s$  where such an input occurs. By Lemma 4,  $s' \vdash \{G\}_k$  and, by induction hypothesis,  $\alpha(s') \vdash \alpha_e(\{G\}_k)$  from which the thesis.

**Env** The judgement  $s \vdash a$  requires  $a \notin \text{bn}(s)$ . By the rule for abstracting new (which is the only binder for names) we obtain that  $\alpha(a) \notin \text{bn}(\alpha(s))$ . Since  $\alpha_e(a) = \alpha(a)$ , we have that  $\alpha(s) \vdash \alpha_e(a)$ , as desired.

**Inductive Case** Rules for pairs and tags are trivial. We focus on the two relevant cases of encryption and decryption

**Encryption** This case is exactly the same as the one we have seen with an empty trace.

**Decryption** This case is new, since with empty trace we had no encryptions apart from the ones generated by the environment itself. We discuss private key decryption. From  $s \vdash \{G\}_{k_1^-}$  we may derive  $s \vdash G$ . Assume that  $\{G\}_{k_1^-}$  has not been generated by the environment, otherwise the thesis is trivially achieved as for the case with empty trace above. By induction,  $\alpha(s) \vdash \alpha_e(\{G\}_{k_1^-})$ . We now have two cases: if  $\{G\}_{k_1^-} \notin \text{dom}(\underline{f}_e)$  we have that  $\alpha_e(\{G\}_{k_1^-}) = \alpha_e(G)$  thus trivially obtaining the thesis. Otherwise,  $\alpha_e(\{G\}_{k_1^-}) = \underline{f}_e(\{G\}_{k_1^-})$ . By condition *Encryption* of Table 6, we know that  $\underline{f}_e(\{G\}_{k_1^-})$  can be either  $\text{Chal}_{G_1^\#}^{\ell, \ell'}(I^\#, J^\#, G_2^\#)$  or  $\text{Resp}_{G_1^\#}^{\ell', \ell}(I^\#, J^\#, G_2^\#)$ , with  $\ell \leq \text{Int}$ .

By rule *Readable*, we obtain that  $\alpha(s) \vdash G_1^\#, G_2^\#$ . By Lemma 3 we obtain that  $\alpha(s) \vdash \alpha(G)$ . We need to show that this implies  $\alpha(s) \vdash \alpha_e(G)$ . Recall that we have assumed that  $\{G\}_{k_i^-}$  has not been generated by the environment. Thus it must have been sent as output by a process, namely  $out(R) \in s$  with either  $\{G\}_{k_i^-} \in terms(R)$  or  $\{G\}_{k_i^-} \in terms([R])$ . The former case is trivial as, by the well-formedness of  $s$  and Lemma 4, we have that  $s' \vdash \{G\}_{k_i^-}$ , for some prefix  $s'$  of  $s$ , from which the result by induction hypothesis. The latter case is more interesting as it represents an encryption performed by a process. Notice that, by *Format* of Table 6, we know that  $R$  cannot contain encryptions, thus, by the well-formedness of  $s$  and Lemma 4, every nested run-time ciphertext  $\{G'\}_k$  possibly occurring in  $G$  has been previously received. By induction on the length of  $s$ , we have that  $\alpha(s) \vdash \{G'\}_k$  and, since these nested ciphertexts are the only part of  $G$  approximated by  $\alpha$  into  $\top$ , we obtain the thesis, i.e.,  $\alpha(s) \vdash \alpha_e(G)$ .  $\square$

The next corollary is useful for the final theorem:

**Corollary 1.**  $s \vdash [C]\sigma$  implies  $\alpha(s) \vdash \alpha(C\sigma)$ .

*Proof.* By the previous proposition we know that  $s \vdash [C]\sigma$  implies  $\alpha(s) \vdash \alpha_e([C]\sigma)$ . By observing that  $C$  does not contain run-time messages, i.e., messages of the form  $\{G\}_k$ , we easily obtain that  $\alpha_e([C]\sigma) = \alpha(C\sigma)$ .

The following lemma says that the abstraction of terms, traces and processes is closed under the substitution of variables with either names or ciphertexts. Here and throughout this paper, we write  $\mathcal{P}$  to denote an arbitrary pattern, namely either a term or a trace or a process, and  $\mathcal{P}^\#$  to denote an abstract pattern.

**Lemma 5 (Substitution).** For every  $\sigma : x \mapsto a \mid \{G\}_k$  and  $\mathcal{P}$ ,  $\alpha(\mathcal{P}\sigma) = \alpha(\mathcal{P})\sigma^\#$ .

*Proof.* We proceed by cases, according to  $\mathcal{P}$ .

*Terms* By induction on the structure of  $\mathcal{P}$ .

$\mathcal{P} = a$  The proof is straightforward as the domain of  $\sigma$  is composed of variables and thus  $\alpha(a\sigma) = \alpha(a) = \alpha(a)\sigma^\#$ .

$\mathcal{P} = x$  If  $x\sigma = a$ , then  $\alpha(x\sigma) = \alpha(a) = a^\# = x^\#\sigma^\# = \alpha(x)\sigma^\#$ , as desired. If  $x\sigma = \{G\}_k$ , then  $\alpha(x\sigma) = \alpha(\{G\}_k) = \top = x^\#\sigma^\# = \alpha(x)\sigma^\#$ .

$\mathcal{P} = (R_1, R_2)$  The result is obtained directly by the induction hypothesis.

$\mathcal{P} = \llbracket R \rrbracket_k$  If  $\llbracket R \rrbracket_k \in dom(\underline{f})$ , then  $\llbracket R \rrbracket_k\sigma \in dom(\underline{f})$  as  $\underline{f}$  is the closure of  $f$  under variable substitution. Thus  $\alpha(\llbracket R \rrbracket_k\sigma) = \underline{f}(\llbracket R \rrbracket_k\sigma) \triangleq \underline{f}(\llbracket R \rrbracket_k)\sigma^\# = \alpha(\llbracket R \rrbracket_k)\sigma^\#$ , as desired. If  $\llbracket R \rrbracket_k \notin dom(\underline{f})$ , then the result is obtained by the induction hypothesis.

$\mathcal{P} = \{G\}_k$  The proof is trivial as  $\{G\}_k$  does not contain variables and  $\alpha(\{G\}_k) = \top$ .

*Traces and Processes* By trivial induction on the length of the trace and the process, respectively, and by the previous item.  $\square$

The next lemma states that the abstraction of processes is preserved under reduction.

**Theorem 1 (Soundness)** If  $\langle s, P \rangle \rightarrow \langle s', P' \rangle$ , then  $\langle \alpha(s), \alpha(P) \rangle \rightarrow \langle \alpha(s'), \alpha(P') \rangle$

*Proof.* By induction on the length of the derivation of  $\langle s, P \rangle \rightarrow \langle s', P' \rangle$ .

**Base** By cases, according to the semantic rule applied. We only discuss the most interesting case, namely INPUT, as the other ones follow straightforwardly from an inspection of the concrete reduction rules and the corresponding abstract ones. Then  $\langle s, \text{in}(C).P \rangle \rightarrow \langle s :: \text{in}(C)\sigma, P\sigma \rangle$ , with  $\sigma : x \mapsto a \mid \{G\}_k$ . By the  $\rho$ -spi rule INPUT,  $s \vdash \llbracket C \rrbracket \sigma$ . By Corollary 1,  $\alpha(s) \vdash \alpha(\overline{C}\sigma)$ . By Lemma 5,  $\alpha(\overline{C}\sigma) = \alpha(\overline{C})\sigma^\#$ . By the CR semantic rule INPUT,  $\langle \alpha(s), \alpha(P) \rangle \rightarrow \langle \alpha(s) :: \text{in}(\alpha(\overline{C})\sigma^\#), \alpha(P)\sigma^\# \rangle$ . By Lemma 5,  $\alpha(P\sigma) = \alpha(P)\sigma^\#$ , as desired.

**Induction** The proof straightforwardly derives from the induction hypothesis. □

## D.2 Safety of the Analysis

Intuitively, the following lemma states that if a process has effect  $e$ , then  $e$  cannot contain more than one challenge atomic effect for every nonce.

**Lemma 6 (Uniqueness of Challenges).** *Let  $P^\#$  be a process and  $e$  an effect s.t.*

1.  $P^\# : e + [! \text{Chal}_{n^\#}^{\ell_C, \ell_R}(\dots), \text{fresh}(n^\#)]$
2.  $\text{fresh}(n^\#) \notin e$  and  $n^\# \notin \text{bn}(P^\#)$

*Then  $\nexists \ell'_C, \ell'_R$  s.t.  $! \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(\dots) \in e$  or  $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(\dots), \dots) \in P^\#$*

*Proof.* It is easy to see that in the derivation of  $P^\# : e + [! \text{Chal}_{n^\#}^{\ell_C, \ell_R}(\dots), \text{fresh}(n^\#)]$ , the atomic effects  $! \text{Chal}_{n^\#}^{\ell_C, \ell_R}(\dots)$  and  $\text{fresh}(n^\#)$  are removed by the validation rule END. Let us assume by contradiction that  $! \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(\dots) \in e$ , for some  $\ell'_C, \ell'_R$ . Thus either two occurrences of  $\text{fresh}(n^\#)$  are in  $e$ , or at least one of them is inserted along the derivation by RES. The contradiction arises by condition 2 in the hypothesis. The reasoning for proving  $\nexists \ell'_C, \ell'_R$  s.t.  $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(\dots), \dots) \in P^\#$  is the same. □

We introduce some notational conventions:  $|f|_e$  denotes the number of occurrences of the atomic effect  $f$  in the effect  $e$  and, similarly,  $|\alpha|_s$  denotes the number of occurrences of the action  $\alpha$  in the trace  $s$ . We write  $\text{nonces}(e)$  and  $\text{msg}(e)$  to denote the nonces and messages, respectively, in the challenge-response atomic effects in  $e$ . Finally,  $\text{plaintexts}(P^\#)$  denotes the names-variables that are sent or received in  $P^\#$  as plaintexts, namely not as subscripts of challenge-response messages. The following definition provides some invariants on traces, processes and effects. Theorem 5 (Effect Preservation) proves that the reduction of successfully validated processes preserves these invariants and Theorem 3 (Safety) exploits this result for proving the safety of processes with empty effect.

**Definition 6 (Balanced Configuration).** *Let  $P^\#$  be a process and  $e$  an effect such that  $P^\# : e$ . We say that a trace  $s^\#$ ,  $P^\#$  and  $e$  are balanced iff the following conditions hold:*

1.  $\text{names}(e) \subseteq \text{fn}(s^\#) \cup \text{bn}(s^\#)$
2. if  $n^\# \in \text{bn}(s^\#)$  and  $n^\# \in \text{fn}(P^\#)$ , then  $n^\# \notin \text{bn}(P^\#)$
3. if  $s \vdash \text{Chal/Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$  and  $s^\# \not\vdash n^\#$ , then  $n^\# \notin \text{msg}(P^\#) \cup \text{plaintexts}(P^\#)$
4.  $|\text{fresh}(n^\#)|_e + |\text{end}_{n^\#}(\cdot)|_{s^\#} \leq |\text{new}(n^\#)|_{s^\#} \leq 1$

5. if  $P^\# = P_1^\# | P_2^\#$  then  $\exists e_1, e_2$  s.t.  $e_1 + e_2 = e$  and  $s^\#, P_i^\#, e_i$  are balanced,  $\forall i \in [1, 2]$
6. if  $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#), \dots) \in P^\#$   
then (i)  $s^\# \vdash M^\# \Rightarrow n^\# \notin \text{names}(M^\#)$  and (ii)  $n^\# \notin \text{plaintexts}(P^\#) \cup \text{msgs}(P^\#)$
7. if  $[!|?]\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in e$  (resp.  $?\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in e$ ),  
then  $s^\# \vdash \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in e$  (resp.  $s^\# \vdash \text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$ )
8. if  $!\text{Resp}_{n^\#}^{\ell_C, \ell_R}(A^\#, B^\#, M_2^\#) \in e$ ,  
then  $\text{begin}_{n^\#}(A^\#, B^\#, M_1^\#; M_2^\#) \in s^\#$  and  $s^\# \vdash \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(B^\#, A^\#, M_1^\#)$ , with  $\ell'_C, \ell'_R \diamond \ell_C, \ell_R$
9. if  $\text{new}(n^\#) \in s^\#$  and  $s^\# \vdash \text{Chal}/\text{Resp}_{n^\#}^{\ell_C, \ell_R}(A^\#, B^\#, M^\#)$ , with  $\ell_C, \ell_R \geq \text{Taint}$ ,  
then  $s^\# \not\vdash n^\#$
10. if  $\text{out}(\dots, \text{Resp}_{n^\#}^{\ell_C, \ell_R}(B^\#, A^\#, M_2^\#), \dots) \in s^\#$ ,  
then  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#; M_2^\#) \in s^\#$ , and  $s^\# \vdash \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(A^\#, B^\#, M_1^\#)$ , with  $\ell'_C, \ell'_R \diamond \ell_C, \ell_R$
11. if  $!\text{Chal}_{n^\#}^{\ell, \text{Pub}}(B^\#, A^\#, M^\#) \in e$  and  $s^\# \vdash n^\#$ , with  $\ell \geq \text{Taint}$ ,  
then  $\text{begin}_{n^\#}(A^\#, B^\#, M^\#) \in s^\#$
12. if  $\text{fresh}(n^\#) \in e$  and  $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#), \dots) \in s^\#$ ,  
then  $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in e$

The intuitive reading is as follows:

1. The names in the effect occur in the trace.
2. Processes are  $\alpha$ -converted so as to avoid clashes in bound names
3. The nonces, which circulate on the network within challenge-response messages and are not known to the enemy, do not occur within plaintexts or authenticated messages.
4. If a bound name  $n^\#$  is a nonce, then either  $n^\#$  is fresh or one  $\text{end}_{n^\#}(\cdot)$  has been asserted.
5. Balancing is propagated through parallel composition.
6. If a nonce is sent as challenge, then it is unknown to the enemy and it does not occur as either plaintext or authenticated message in any following output.
7. The presence of effects *tracking* the reception-generation of challenge-response messages implies the presence of such messages in the knowledge of the environment.
8. The presence of effects *justifying* the generation of a response implies that a suitable begin assertion has been asserted and the corresponding challenge is known to the environment.
9. The nonces sent in a Priv or Taint challenge (resp. response) requiring a Priv or Taint response (resp. challenge) are not known to the environment.
10. If  $B^\#$  sends a response with nonce  $n^\#$  and message  $M_2^\#$  ( $\text{out}(\dots, \text{Resp}_{n^\#}^{\ell_C, \ell_R}(B^\#, A^\#, M_2^\#), \dots) \in s^\#$ ), then  $B^\#$  has asserted  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#; M_2^\#)$  and the environment knows the corresponding challenge ( $s^\# \vdash \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(A^\#, B^\#, M_1^\#)$ ).
11. If the nonce is sent in a challenge requiring a Pub response ( $!\text{Chal}_{n^\#}^{\ell, \text{Pub}}(B^\#, A^\#, M^\#) \in e$ ) and it is known to the environment ( $s^\# \vdash n^\#$ ), then  $A^\#$  has asserted  $\text{begin}_{n^\#}(A^\#, B^\#, M^\#)$ .

12. This condition says that if the nonce  $n^\#$  is fresh ( $\text{fresh}(n^\#) \in e$ ) and a challenge with nonce  $n^\#$  and message  $M^\#$  is sent on the network ( $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#), \dots) \in s^\#$ ), then  $A^\#$  has not completed the handshake yet ( $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in e$ ).

The next theorem states that successfully validated processes preserve under reduction the invariants of Definition 6.

**Theorem 5 (Preservation).** *Let  $P^\#$  and  $Q^\#$  be processes,  $s^\#$  a trace and  $e$  an effect s.t.*

- $\langle s^\#, Q^\# \rangle \rightarrow \langle s'^\#, P^\# \rangle$
- $Q^\# : e$
- $s^\#, Q^\#, e$  are balanced

then it is possible to find an effect  $e'$  such that

- $P^\# : e'$
- $s'^\#, P^\#, e'$  are balanced

*Proof.* We reason by induction on the length of the derivation for  $\langle s^\#, Q^\# \rangle \rightarrow \langle s'^\#, P^\# \rangle$  (if  $Q^\#$  is a parallel composition or a process with an identifier, then the semantic derivation is composed of more steps). The proof of the base case proceeds by cases, according to the semantic rule applied.

**REPLICATION**  $\boxed{!P^\# : \square \rightarrow !P^\# | P^\# : \square}$

By REPLICATION,  $!P^\# : \square$  only if  $P^\# : \square$ . Notice that since the type and effect system is syntax directed, typing rules can be read upside-down: thus, from  $!P^\# : \square$  we derive  $P^\# : \square$ . By PAR,  $!P^\# | P^\# : \square$ .

**NEW**  $\boxed{\text{new}(n^\#).P^\# : e - [\text{fresh}(n^\#)] \rightarrow P^\# : e}$

The reduction above is proved by the validation rule NEW. We need to prove that the target configuration composed of  $s :: \text{new}(n^\#), P^\#, e$  is still balanced.

**Condition 2** By the validation rule NEW,  $n^\# \notin \text{bn}(P^\#)$ , as desired.

**Condition 4** By RES,  $n^\# \notin \text{fn}(s^\#) \cup \text{bn}(s^\#)$ : thus  $|\text{new}(n^\#)|_{s^\#} = 0$ ,  $|\text{end}_{n^\#}(\cdot)|_{s^\#} = 0$  and, by condition 1,  $|\text{fresh}(n^\#)|_e = 0$ . Hence  $|\text{fresh}(n^\#)|_e + |\text{end}_{n^\#}(\cdot)|_{s^\# :: \text{new}(n^\#)} \leq |\text{new}(n^\#)|_{s^\# :: \text{new}(n^\#)}$ , as desired.

**Condition 12** Let us suppose by contradiction that condition 12 does not hold on the target configuration. Hence  $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#), \dots) \in s^\#$  and  $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \notin e$ . Thus  $n^\# \in \text{fn}(s^\#) \cup \text{bn}(s^\#)$ . By the semantic rule RES,  $n^\# \notin \text{fn}(s^\#) \cup \text{bn}(s^\#)$ , giving a contradiction.

**BEGIN**  $\boxed{\text{begin}_{N^\#}(A^\#, I^\#, M_1^\#, M_2^\#).P^\# : e + [?\text{Chal}_{N^\#}^{\ell_C, \ell_R}(I^\#, A^\#, M_1^\#)] \rightarrow P^\# : e + [!\text{Resp}_{N^\#}^{\ell'_C, \ell'_R}(A^\#, I^\#, M_2^\#)]}$

The reduction above is proved by the validation rule BEGIN. The only interesting condition for the balancing of the target configuration is 8, which immediately follows by an inspection of the target configuration. The proof for the other ones derives directly from the balancing of the source configuration.

**INPUT**  $\boxed{\text{in}(C^\#).P^\# : e \rightarrow P^\#\sigma : e+?e_C+?e_R\sigma, \text{ where } s^\# \vdash C^\#\sigma \text{ and } C^\# : (e_C; e_R)}$

The validation rule INPUT proves  $\text{in}(C^\#).P^\# : e$ , if the judgement  $P^\# : e+?e_C+?e_R$  holds, where  $C^\# : (e_C; e_R)$ . By the semantic rule INPUT,  $s^\# \vdash C^\#\sigma$ . By Lemma 5 (Substitution),  $P^\#\sigma : e + e_C + e_R\sigma$ . The target configuration is trivially balanced.

**OUTPUT**  $\boxed{\text{out}(R^\#).P^\# : e+!e_R \rightarrow P^\# : e+!e_C, \text{ where } R^\# : (e_C; e_R)}$

The reduction above is proved by OUTPUT. In the following, we discuss the interesting cases for the balancing of the target configuration:

**Condition 6** Let us suppose by contradiction that  $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell'_C, \ell'_R}(A^\#, B^\#, M^\#), \dots) \in P^\#$ , for some  $\ell'_C, \ell'_R$ , and the output of  $R^\#$  reveals a term containing  $n^\#$ , namely  $s :: \text{out}(R^\#) \vdash M^\#$  with  $n^\# \in \text{names}(M^\#)$ . Since the source configuration is balanced, by condition 6  $n^\# \notin \text{msgs}(P^\#) \cup \text{plaintexts}(P^\#)$ . Let us suppose that  $\text{Chal}_{n^\#}^{\ell_C, \ell_R}(\dots) \in \text{terms}(R^\#)$ , for some  $\ell_C, \ell_R$ . Since  $\text{Chal}_{n^\#}^{\ell_C, \ell_R}(\dots) \in e_C$ , and the process is successfully validated,  $\text{fresh}(n^\#) \in e$ . By condition 4,  $\text{new}(n^\#) \in s^\#$ . By condition 2,  $n^\# \notin \text{bn}(P^\#)$ . The contradiction arises by Lemma 6. The reasoning is similar for  $\text{Resp}_{n^\#}^{\ell_C, \ell_R}(\dots) \in \text{terms}(R^\#)$  and relies on the balancing conditions 8 and 12.

**Condition 9** Let us assume by contradiction  $\text{new}(n^\#) \in s^\#, s^\# :: \text{out}(R^\#) \vdash \text{Chal}/\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$ , with  $\ell, \ell' \geq \text{Taint}$  and  $s^\# :: \text{out}(R^\#) \vdash n^\#$ . Since the source configuration is balanced, by condition 9 at least one between  $n^\#$  and  $\text{Chal}/\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$  is unknown to the environment. We proceed by cases according to what is revealed by the output of  $R^\#$ :

$s^\# \vdash \text{Chal}/\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$  and  $s^\# \not\vdash n^\#$  We proceed by cases on the form of  $R^\#$ :

$R^\# = \dots, n^\#, \dots$

By condition 3,  $n^\# \notin \text{plaintexts}(\text{out}(R^\#).P^\#)$ , thus giving a contradiction.

$R^\# = \dots, \text{Chal}_{N^\#}^{\ell_C, \ell_R}(I^\#, J^\#, M^\#), \dots$ , with  $n^\# \in \text{names}(N^\#, M^\#)$  and  $\ell_C \leq \text{Int}$

Since the source configuration is balanced and  $s^\# \vdash \text{Chal}/\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$ , by condition 3,  $n^\# \notin \text{names}(M^\#)$ . Thus  $n^\# \in \text{names}(N^\#)$  and  $\text{Chal}_{N^\#}^{\ell_C, \ell_R}(I^\#, J^\#, M^\#) \in e_R$ . Since the process is successfully validated, it is easy to see that  $\text{fresh}(n^\#) \in e$  and  $n^\# = N^\#$ . Let us suppose that  $s^\# \vdash \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$ : since  $s^\# \not\vdash n^\#, \text{out}(\dots, \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#), \dots) \in s^\#$ .

By condition 12,  $!\text{Chal}/\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in e$ . The contradiction arises by Lemma 6. If  $s^\# \vdash \text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$ , then the reasoning is similar and relies on the balancing conditions 8 and 12.

$R^\# = \dots, \text{Resp}_{N^\#}^{\ell_C, \ell_R}(I^\#, J^\#, M^\#), \dots$ , with  $n^\# \in \text{names}(N^\#, M^\#)$  and  $\ell_C \leq \text{Int}$

By OUTPUT,  $\text{Resp}_{N^\#}^{\ell_C, \ell_R}(I^\#, J^\#, M^\#) \in e_R$  and, by condition 8,  $s^\# \vdash \text{Chal}_{N^\#}^{\ell'_C, \ell'_R}(J^\#, I^\#, M^\#)$ , with  $\ell'_C, \ell'_R \diamond \ell_C, \ell_R$ . By the message manipulation rule READABLE,  $s^\# \vdash N^\#, M^\#$  and, consequently,  $s^\# \vdash n^\#$ . This contradicts the hypothesis  $s^\# \not\vdash n^\#$ .

$s \vdash n^\#$  and  $s^\# \not\vdash \text{Chal}/\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$  Thus either  $\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in \text{terms}(R^\#)$  or  $\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in \text{terms}(R^\#)$ . In the former case, the contradiction arises by Lemma 6; in the latter case, by condition 8.

$s \not\vdash n^\#$  and  $s^\# \not\vdash \text{Chal}/\text{Resp}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#)$  The reasoning is similar to the one in the previous items.

**Condition 10** The proof follows directly from condition 8.

**Condition 11** Let us assume by contradiction that  $\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in \text{terms}(R^\#)$  and  $s^\# \vdash n^\#$ . The contradiction arises by condition 6. If, instead,  $\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M^\#) \in e$  and  $s :: \text{out}(R) \vdash n^\#$ , namely the output reveals  $n^\#$ , then the reasoning is similar to the one for condition 9.

**END**  $\boxed{\text{end}_{n^\#}(A^\#, I^\#, M_1^\#, M_2^\#).P^\# : e + [\text{fresh}(n^\#), !\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, I^\#, M_1^\#), ?\text{Resp}_{n^\#}^{\ell, \ell'}(I^\#, A^\#, M_2^\#)] \rightarrow P^\# : e}$

The reduction above is proved by END. We need to check that the target configuration is still balanced.

**Condition 4** Trivial, by the balancing of the source configuration and an inspection of the target one.

**Condition 12** This condition might fail if  $\text{fresh}(n^\#) \in e$ . However, by condition 4,  $\text{fresh}(n^\#) \notin e$ .

The inductive step concerns processes associated to an identifier and processes in parallel composition:

**IDENTITY**  $\boxed{A^\# \triangleright P^\# : e \rightarrow A^\# \triangleright P'^\# : e'}$

If  $A^\# \triangleright P^\# : e$ , then, by IDENTITY,  $P^\# : e$ . If  $\langle s^\#, P^\# \rangle \rightarrow \langle s'^\#, P'^\# \rangle$ , then, by induction hypothesis,  $\exists e'$  such that  $P'^\# : e'$ . The result follows from IDENTITY and the balancing of the target configuration from the induction hypothesis.

**PAR**  $\boxed{P_1^\# | P_2^\# : e_1 + e_2 \rightarrow P_1^\# | P_2^\# : e'_1 + e_2}$

By PAR,  $P_i^\# : e_i$ , with  $i \in [1, 2]$ . Let us suppose that  $\langle s^\#, P_1^\# \rangle \rightarrow \langle s'^\#, P_1'^\# \rangle$ : the symmetric case is analogous.

By induction hypothesis, we can find an effect  $e'_1$  s.t.  $P_1'^\# : e'_1$ . By PAR,  $P_1'^\# | P_2^\# : e'_1 + e_2$ .

By hypothesis the source configuration is balanced and, by condition 5,  $s^\#, P_i^\#, e_i$  are balanced,  $\forall i \in [1, 2]$ .

By induction hypothesis, even  $s'^\#, P_1'^\#, e'_1$  are balanced. It remains to prove that  $s'^\#, P_1'^\# | P_2^\#, e'_1 + e_2$  are balanced. We only discuss the interesting cases.

**Condition 4** The only interesting cases are  $s'^\# = s^\# :: \alpha$ , with  $\alpha \in \{\text{new}(n^\#), \text{end}_{n^\#}(\cdot)\}$ .

$\alpha = \text{new}(n^\#)$  By the semantic rule RES,  $n^\# \notin \text{names}(s^\#)$  and, by condition 1,  $n^\# \notin \text{names}(e_1) \cup \text{names}(e_2)$  and the inequality trivially holds.

$\alpha = \text{end}_{n^\#}(\cdot)$  By the typing rule END,  $\text{fresh}(n^\#) \in e_1$ . By condition 4,  $\text{end}_{n^\#}(\cdot) \notin s^\#$  and  $\text{fresh}(n^\#) \notin e_2$ . By induction hypothesis  $s :: \text{end}_{n^\#}(\cdot), P_1^\#, e_1$  are balanced: by condition 4 and the typing rule END,  $\text{fresh}(n^\#) \notin e'_1$ . Hence  $|\text{fresh}(n^\#)|_{e'_1 + e_2} = 0$  and  $|\text{end}_{n^\#}(\cdot)|_{s :: \text{end}_{n^\#}(\cdot)} = 1$ , getting  $0 + 1 \leq 1$  as desired.

**Condition 12** Let us suppose by contradiction that condition 12 does not hold, i.e.,  $\text{fresh}(n^\#) \in e'_1 + e_2$ ,  $\text{out}(\dots, \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#), \dots) \in s^\# :: \alpha^\#$  and  $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#) \notin e'_1 + e_2$ . The only interesting case is  $\alpha = \text{end}_{n^\#}(A^\#, B^\#, M_1^\#, M_2^\#)$  with  $\text{fresh}(n^\#) \in e_2$  as  $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#)$  is removed by END, i.e.,  $e_1 = e'_1 + [\text{fresh}(n^\#), !\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#), ?\text{Resp}_{n^\#}^{\ell, \ell'}(B^\#, A^\#, M_2^\#)]$ . Since the source configuration is balanced and  $\text{fresh}(n^\#) \in e_1$ , by condition 4,  $\text{fresh}(n^\#) \notin e'_1 + e_2$ , giving a contradiction.

The Safety Theorem exploits the preservation of the balancing conditions under process reduction for proving the safety of processes with empty effect. As stated in Remark 1, we only consider processes where  $\text{nonces}(P^\#) \cap (\text{msgs}(P^\#) \cup \text{plaintexts}(P^\#)) = \emptyset$  so that condition 6 holds. Notice that this is always true for processes yielded by the prevalidation algorithm.

**THEOREM 3 (SAFETY)** *If  $P^\# : []$ , then  $P^\#$  is safe.*

*Proof.* We need to show that every trace generated by  $P^\#$  is safe. We reason by induction on the length of the derivation of  $\langle \varepsilon, P^\# \rangle \rightarrow^* \langle s^\#, Q^\# \rangle$ . The base case is the null derivation which trivially holds as the null trace  $\varepsilon$  is safe.

Let  $\langle \varepsilon, P^\# \rangle \rightarrow^* \langle s^\#, A^\# \triangleright \text{end}_{n^\#}(A^\#, B^\#, M_1^\#, M_2^\#).R^\# | Q^\# \rangle \rightarrow \langle s^\# :: A^\# \triangleright \text{end}_{n^\#}(A^\#, B^\#, M_1^\#, M_2^\#), A^\# \triangleright R^\# | Q^\# \rangle$  be a semantic derivation leading to an *end* assertion.

The proof is divided in two steps. First, we prove that  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#, M_2^\#) \in s^\#$ , thus showing that every  $\text{end}_{n^\#}(A^\#, B^\#, M_1^\#, M_2^\#)$  in  $s^\#$  is preceded by a  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#, M_2^\#)$  (Non-Injective Agreement). Next, we also prove that  $s^\#$  does not contain actions having the form  $\text{end}_{n^\#}(\cdot)$ , thus proving that every  $\text{end}_{n^\#}(A^\#, B^\#, M_1^\#, M_2^\#)$  in  $s^\#$  is preceded by a *distinct*  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#, M_2^\#)$  (Agreement).

**Non-Injective Agreement** By Theorem 5,  $\exists e$  s.t.  $A^\# \triangleright \text{end}_{n^\#}(A^\#, B^\#, M_1^\#, M_2^\#).R^\# | Q^\# : e$ . By an inspection of the validation rules, that judgement is proved by PAR and END. Hence,  $\text{fresh}(n^\#) \in e$ ,  $!\text{Chal}_{n^\#}^{\ell_C, \ell_R}(A^\#, B^\#, M_1^\#) \in e$  and  $?\text{Resp}_{n^\#}^{\ell_C, \ell_R}(B^\#, A^\#, M_2^\#) \in e$ . We proceed by cases according to  $\ell'_R$ .

$\ell'_R = \text{Priv}$  By the balancing condition 7,  $s^\# \vdash \text{Resp}_{n^\#}^{\ell'_C, \ell'_R}(B^\#, A^\#, M_2^\#)$ . Since the ciphertext is not forgeable by the environment,  $\text{out}(\dots, \text{Resp}_{n^\#}^{\ell'_C, \ell'_R}(B^\#, A^\#, M_2^\#), \dots) \in s^\#$  and, by the balancing condition 10 we get  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#, M_2^\#) \in s^\#$  and  $s^\# \vdash \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#)$ , with  $\ell, \ell' \diamond \ell'_C, \ell'_R$ . We have to show that  $M_1^\# = M_1'^\#$ . If  $\ell'_C = \text{Pub}$ , then  $M_1^\# = M_1'^\# = \varepsilon$ , as desired. If  $\ell'_C \neq \text{Pub}$  then the challenge is not forgeable by the environment: indeed, if  $\ell'_C = \text{Taint}$ , then, by the balancing condition 9,  $s^\# \not\vdash n^\#$ . By the balancing condition 12,  $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#) \in e$ . By Lemma 6,  $M_1^\# = M_1'^\#$ , as desired.

$\ell'_R = \text{Int}$  By the balancing condition 7,  $s^\# \vdash \text{Resp}_{n^\#}^{\ell'_C, \ell'_R}(B^\#, A^\#, M_2^\#)$ . Since the ciphertext is not forgeable by the environment,  $\text{out}(\dots, \text{Resp}_{n^\#}^{\ell'_C, \ell'_R}(B^\#, A^\#, M_2^\#), \dots) \in s^\#$  and, by the balancing condition 10, we get  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#, M_2^\#) \in s^\#$  and  $s^\# \vdash \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#)$ , with  $\ell, \ell' \diamond \ell'_C, \ell'_R$ . We have to show that  $M_1^\# = M_1'^\#$ . If  $\ell'_C = \text{Pub}$ , then  $M_1^\# = M_1'^\# = \varepsilon$ , as desired. Let us suppose that  $\ell'_C \neq \text{Pub}$ . By the constraints on security levels imposed by END,  $\ell'_C \in \{\text{Priv}, \text{Int}\}$ . Thus the ciphertext is not forgeable by the environment and, by the balancing condition 12,  $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#) \in e$ . By Lemma 6,  $M_1^\# = M_1'^\#$ , as desired.

$\ell'_R = \text{Taint}$  By the constraints on security levels imposed by END,  $\ell'_C \in \{\text{Priv}, \text{Taint}\}$ . By the balancing condition 9,  $s^\# \not\vdash n^\#$ . By the balancing condition 7,  $s^\# \vdash \text{Resp}_{n^\#}^{\ell'_C, \ell'_R}(B^\#, A^\#, M_2^\#)$ . Since the ciphertext is not forgeable by the environment,  $\text{out}(\dots, \text{Resp}_{n^\#}^{\ell'_C, \ell'_R}(B^\#, A^\#, M_2^\#), \dots) \in s^\#$  and, by the balancing condition 10, we get  $\text{begin}_{n^\#}(B^\#, A^\#, M_1^\#, M_2^\#) \in s^\#$  and  $s^\# \vdash \text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#)$ , with  $\ell, \ell' \diamond \ell'_C, \ell'_R$ . We have to show that  $M_1^\# = M_1'^\#$ . Since  $s^\# \not\vdash n^\#$ , the challenge is not forgeable by the environment and, by the balancing condition 12,  $!\text{Chal}_{n^\#}^{\ell, \ell'}(A^\#, B^\#, M_1^\#) \in e$ . By Lemma 6,  $M_1^\# = M_1'^\#$ , as desired.

$\ell'_R = \text{Pub}$  The proof straightforwardly derives from the balancing condition 11 and Lemma 6.

**Agreement** The balancing condition 4 proves that  $\text{end}_{n^\#}(\cdot) \notin s^\#$ , as desired.  $\square$

## E Extensions

In this section we extend the class of protocols analyzed by considering (i) nested encryptions and (ii) ciphertexts representing both a challenge and a response, typically employed in mutual authentication protocols where principals authenticate with each other.

---

**Table 19** Encryption abstraction with nesting

---

A partial function  $f : \{C\}_k \mapsto \text{Chal}/\text{Resp}_{N^\#}^{\ell, \ell'}(I^\#, J^\#, M^\#)$  is an *encryption abstraction* if the following conditions hold:

*Format* -  $\text{names}(C) \subseteq ID$  and  $\text{names}(N^\#) = \text{names}(M^\#) = \emptyset$ ;  
-  $x \in \text{vars}(C)$  if and only if  $x^\# \in \text{vars}(N^\#) \cup \text{vars}(M^\#)$ ;

*Unique Abstraction* if  $\exists C, C' \in \text{dom}(f)$  s.t.  $[C]\sigma = [C']\sigma'$  with  $\sigma, \sigma' : x \mapsto a|\{G\}_k$ , then  $C = C'$ ;

*Nesting* if  $\exists C' \in \text{dom}(f)$  and  $\{C\}_k$  s.t.  $\{C\}_k \in \text{terms}(C')$ ,  
then  $\nexists \sigma$  s.t.  $\{C\}_k \sigma \in \text{dom}(f)$ , with  $\sigma : x \mapsto a|\{G\}_k$  (nested encryptions do not belong to the domain of  $f$ );

*Encryption* as in Table 6

---

## E.1 Nesting of Encryptions

Extending the analysis to nested encryptions requires to tackle the following problem: since encryptions and decryptions performed by principals may disclose ciphertexts which were formerly nested and unknown to the environment, we have to guarantee that the abstraction of such ciphertexts gets known to the abstract environment as well.

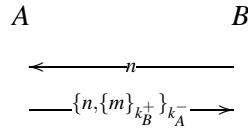
We address this issue by constraining the form of nested components. In particular, we require that nested components represent neither challenges nor responses.

This is technically achieved by refining the definition of encryption abstraction as reported in Table 19. Notice that *Format* does not prevent nested encryptions, which are instead constrained by condition *Nesting*. This requires that nested encryptions represent neither challenges nor responses, i.e., they do not belong to the domain of  $f$ , namely the closure of  $f$ . Finally, notice that *Unique Abstraction* is modified so as to check that every run-time message, possibly containing nested ciphertexts, can be generated by at most one encryption in the domain of  $f$ . The abstraction of encryptions is modified as follows:

$$\alpha(\{C\}_k) = \begin{cases} f(\{C\}_k) & \text{if } \{C\}_k \in \text{dom}(f) \\ \perp & \text{otherwise} \end{cases}$$

We introduce the new failure symbol  $\perp$ , symbolizing encryptions that do not belong to the domain of  $f$  and, consequently, are not correctly handled by the abstraction. Soundness and safety theorems hold only for processes where the symbol  $\perp$  does not occur.

*Example 5.* To illustrate the use of nested encryptions, let us consider the following protocol:



The nonce  $n$  is sent as plaintext in the challenge while the response is signed with  $A$ 's private key, thus being integer. The authenticated message  $m$  is encrypted with  $B$ 's public-key, thus simultaneously protecting the privacy of  $m$  and specifying the intended verifier. We can define the encryption abstraction as follows:

$$f : \{x, \{y\}_{k_B^+}\}_{k_A^-} \rightarrow \text{Resp}_{x^\#}^{\text{Pub, Int}}(A^\#, B^\#, y^\#)$$

Notice that condition *Nesting* is trivially satisfied and  $f$  is an encryption abstraction. The CR protocol narration is depicted below:

$$\begin{array}{ccc}
A & & B \\
\longleftarrow n^\# & & \\
\text{---Resp}_{n^\#}^{\text{Pub,Int}}(A^\#, B^\#, m^\#)\text{---} & & 
\end{array}$$

Notice that this CR protocol actually abstracts several  $\rho$ -spi protocols, besides the one previously reported. For instance, some other possible instances are depicted below along with the corresponding encryption abstraction:

$$\begin{array}{ccc}
A & & B & & A & & B \\
\longleftarrow n & & & & \longleftarrow n & & \\
\text{---}\{B, n, m\}_{k_A^-}\text{---} & & & & \text{---}\{n, m\}_{k_B^+}\}_{k_A^-}\text{---} & & 
\end{array}
\quad f(C) = \begin{cases} \text{Resp}_{y^\#}^{\text{Pub,Int}}(A^\#, B^\#, y^\#) & \text{if } C = \{\{B, x, y\}_{k_A^-}\} \\ \text{Resp}_{x^\#}^{\text{Pub,Int}}(A^\#, B^\#, y^\#) & \text{if } C = \{\{x, y\}_{k_B^+}\}_{k_A^-} \\ \uparrow & \text{otherwise} \end{cases}$$

We leave to the interested reader the definition of the corresponding  $\rho$ -spi processes and the validation of their CR abstraction.

**Proofs** Since the effect system has not been modified, we have only to prove the soundness of the abstraction. We shall discuss the changes with respect to the proofs in Section D. We intentionally preserve the names of lemmas and theorems so as to facilitate the comparison. The lemma on the abstraction of the environment is mostly the same, apart from the requirement that the term abstraction is not  $\perp$ .

**Lemma 7 (Environment Abstraction).**  $s^\# \vdash \alpha_e(\llbracket R \rrbracket)$  implies  $s^\# \vdash \alpha(R)$ , if  $\alpha(R) \neq \perp$ .

*Proof.* The proof is the same as the one of Lemma 2.

The next lemma says that if the environment knows an encryption in the domain of  $f$ , then it knows also every nested run-time ciphertext occurring therein.

**Lemma 8 (Nesting and Environment's Knowledge).** Let  $s$  a trace s.t.  $\perp \notin \text{terms}(\alpha(s))$ . Then  $\forall \sigma, G, G'$  s.t.

- $s \vdash G$
- $G' \in \text{terms}(G)$
- $G' = \llbracket C \rrbracket \sigma$ , for some  $C \in \text{dom}(f)$

we have that  $\forall \{G''\}_k \in \text{range}(\sigma)$ ,  $s \vdash \{G''\}_k$

*Proof.* By induction on the length of  $s$  and on the derivation length of  $s \vdash G$ .

**Base Case**  $s = \varepsilon$  As in Lemma 4.

**Inductive Case**  $s \neq \varepsilon$  We now consider a non-empty trace  $s$ .

**Base Cases** The only interesting case is **Out**:

**Out** The judgement  $s \vdash \llbracket R \rrbracket$  derives by  $\text{out}(R) \in s$ . Let us suppose that  $G' \in \text{terms}(\llbracket R \rrbracket)$  and  $G' = \llbracket C \rrbracket \sigma$ , for some  $C \in \text{dom}(f)$ . If  $G'$  is not generated by an encryption in  $R$ , then, since the trace is well-formed (cf. Definition 5), it turns out that  $G'$  has been previously received as input, possibly nested: since we assume that  $\perp$  does not occur in  $\alpha(s)$ , the thesis derives directly from the induction hypothesis. Let us suppose that  $G'$  is generated by an encryption in  $R$ . Recall that  $G' = \llbracket C \rrbracket \sigma$ : by condition *Unique Abstraction*,  $C$  is unique. Since  $\alpha(R) \neq \perp$  and the trace is well-formed,  $\forall \{G''\}_k \in \text{range}(\sigma)$ ,  $\{G''\}_k$  has been previously received as input, possibly nested: The thesis derives directly from the induction hypothesis.

**Inductive Cases** The proof derives straightforwardly from the induction hypothesis.  $\square$

Finally, we discuss the proof of the main proposition.

**Proposition 2 (Knowledge Preservation).** *If  $s \vdash G$ , then  $\alpha(s) \vdash \alpha_e(G)$ ;*

*Proof.* By induction on the length of  $s$  and on the derivation length of  $s \vdash G$ .

**Base Case  $s = \varepsilon$**  We start by considering an empty trace, which is the base case for the external induction:

**Base Cases** As in Proposition 1.

**Inductive Cases** Since the rules for pairs and tags trivially follow from the induction hypothesis, we focus on the two relevant cases of encryption and decryption:

**Encryption** In the following, we discuss public-key encryption: the proof for encryptions via enemy-keys is similar. From  $\varepsilon \vdash G$  we may derive  $\varepsilon \vdash \{G\}_{k_1^+}$ . By induction,  $\varepsilon \vdash \alpha_e(G)$ . We now have two cases: if  $\{G\}_{k_1^+} \notin \text{dom}(\underline{f}_e)$  we have that  $\alpha_e(\{G\}_{k_1^+}) = \alpha_e(G)$  thus trivially obtaining the thesis. Otherwise,  $\alpha_e(\{G\}_{k_1^+}) = \underline{f}_e(\{G\}_{k_1^+})$ . By condition *Encryption* of Table 6, we know that  $\underline{f}_e(\{G\}_{k_1^+})$  can be either  $\text{Chal}_{G_1^\#}^{\ell, \ell'}(J^\#, I^\#, G_2^\#)$  or  $\text{Resp}_{G_1^\#}^{\ell', \ell}(J^\#, I^\#, G_2^\#)$ , with  $\ell \leq \text{Taint}$ . Moreover, by Lemma 7, we have that  $\varepsilon \vdash \alpha_e(G)$  implies  $\varepsilon \vdash \alpha(G)$ . Notice that  $\alpha(G) \neq \perp$  as run-time messages are never abstracted into  $\perp$ . By Lemma 3 we obtain that  $\varepsilon \vdash G_1^\#, G_2^\#$ . By rule *Forgeable*, we obtain that  $\varepsilon \vdash \underline{f}_e(\{G\}_{k_1^+})$ , i.e.,  $\varepsilon \vdash \alpha_e(\{G\}_{k_1^+})$ .

**Decryption** Since, with an empty trace, encryptions can only be generated by the environment itself,  $\varepsilon \vdash \{G\}_k$  has been for sure obtained using the encryption rule by the hypothesis  $\varepsilon \vdash G$ . By applying the induction hypothesis we directly obtain  $\varepsilon \vdash \alpha_e(G)$ .

**Inductive Case  $s \neq \varepsilon$**  We now consider a non-empty trace  $s$ .

**Base Cases** We have two possible derivations of length one, namely *Out* and *Env*:

**Out** The judgement  $s \vdash [R]$  derives by  $\text{out}(R) \in s$ . By trace abstraction we know that  $\text{out}(\alpha(R)) \in \alpha(s)$  and, by *Out*,  $\alpha(s) \vdash \alpha(R)$ . Since we assume  $\alpha(R) \neq \perp$ ,  $\alpha(R)$  and  $\alpha_e([R])$  may only differ for some  $\{G\}_k \in \text{terms}(R)$  which is abstracted into  $\top$  in  $\alpha(R)$ . However, by Definition 5, a message  $\{G\}_k$  can be present in  $R$  only if it has been previously read from the environment. Let  $s'$  be the prefix of  $s$  where such an input occurs. It is easy to see that  $s' \vdash \{G\}_k$ : if  $\{G\}_k$  is nested, this is proved by Lemma 8. By induction hypothesis,  $\alpha(s') \vdash \alpha_e(\{G\}_k)$  from which the thesis.

**Env** As in Proposition 1.

**Inductive Case** Rules for pairs and tags are trivial. We focus on the two relevant cases of encryption and decryption

**Encryption** This case is exactly the same as the one we have seen with an empty trace.

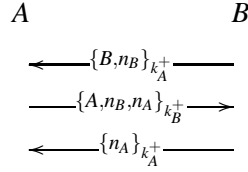
**Decryption** This case is new, since with empty trace we had no encryptions apart the ones generated by the environment itself. We discuss private key decryption. From  $s \vdash \{G\}_{k_1^-}$  we may derive  $s \vdash G$ . Assume that  $\{G\}_{k_1^-}$  has not been generated by the environment, otherwise the thesis is trivially achieved as for the case with empty trace above. By induction,  $\alpha(s) \vdash \alpha_e(\{G\}_{k_1^-})$ . We now have two cases: if  $\{G\}_{k_1^-} \notin \text{dom}(\underline{f}_e)$  we have that  $\alpha_e(\{G\}_{k_1^-}) = \alpha_e(G)$  thus trivially obtaining the thesis. Otherwise,  $\alpha_e(\{G\}_{k_1^-}) = \underline{f}_e(\{G\}_{k_1^-})$ . By condition *Encryption* of Table 6, we know that  $\underline{f}_e(\{G\}_{k_1^-})$  can be either  $\text{Chal}_{G_1^\#}^{\ell, \ell'}(I^\#, J^\#, G_2^\#)$  or  $\text{Resp}_{G_1^\#}^{\ell', \ell}(I^\#, J^\#, G_2^\#)$ , with  $\ell \leq \text{Int}$ . By rule *Readable*, we obtain that  $\alpha(s) \vdash G_1^\#, G_2^\#$ . By Lemma 3 we obtain that  $\alpha(s) \vdash \alpha(G)$ . We need to show that this implies  $\alpha(s) \vdash \alpha_e(G)$ . Recall that we have assumed that  $\{G\}_{k_1^-}$  has not been generated by the environment. Thus it must have been sent as output by a process, namely

$out(R) \in s$  with either  $\{G\}_{k_l^-} \in terms(s)$  or  $\{G\}_{k_l^-} \in terms([R])$ . The former case is trivial as, by the well-formedness of  $s$  and Lemma 8, we have that  $s' \vdash \{G\}_{k_l^-}$ , for some prefix  $s'$  of  $s$ , from which the result by induction hypothesis. The latter case is more interesting as it represents an encryption performed by a process. By condition *Format* of Table 6, all terms, apart from nested run-time ciphertexts, are preserved by the abstraction. Furthermore, by condition *Nesting* of Table 6, nested encryptions  $\{\!\{R\}\!\}_k$  do not belong to the domain of  $\underline{e}_e$  and  $\alpha_e(\{\!\{R\}\!\}_k)$  is thus by definition  $\alpha_e(R)$ . Run time ciphertexts are abstracted into  $\top$ : however, by induction hypothesis this have been previously received from the network and, by Lemma 8, are already known to the environment. Hence, possibly applying the induction hypothesis on run-time ciphertexts, we obtain the thesis, i.e.,  $\alpha(s) \vdash \alpha_e(G)$ .  $\square$

## E.2 Mutual Authentication

Table 20 extends the encryption abstraction by considering ciphertexts that are both challenges and responses. The modifications affect conditions *Format* and *Nesting* and simply amount to propagate the original condition to all the challenge-response messages in the abstraction of the ciphertext. The modification required in the proof are marginal and harmless. We leave them to the interested reader.

*Example 6.* To illustrate, let us consider the well-known Needham-Schroeder-Lowe public-key authentication protocol.



The encryption abstraction for the three ciphertexts may be defined as follows:

$$f(C) = \begin{cases} \text{Chal}_{x^\#}^{\text{Taint}, \text{Taint}}(B^\#, A^\#) & \text{if } C = \{\!\{B, x\}\!\}_{k_A^+} \\ (\text{Resp}_{x^\#}^{\text{Taint}, \text{Taint}}(A^\#, B^\#), \text{Chal}_{y^\#}^{\text{Taint}, \text{Pub}}(A^\#, B^\#)) & \text{if } C = \{\!\{A, x, y\}\!\}_{k_B^+} \\ \text{Resp}_{x^\#}^{\text{Taint}, \text{Pub}}(B, A) & \text{if } C = \{\!\{x\}\!\}_{k_A^+} \\ \uparrow & \text{otherwise} \end{cases}$$

Notice that the third ciphertext is abstracted into an untrusted response: this respects condition *Encryption* which requires  $\ell \leq \text{Taint}$ . Indeed, this ciphertext does not convey any authenticated message and might be sent in clear, thus simplifying the protocol. We remark that the structure of the three ciphertexts is different and thus the use of tags is not required: in fact,  $f$  is an encryption abstraction. We leave to the interested reader the definition of the  $\rho$ -spi process narration and the validation of its abstraction.

---

**Table 20** Encryption abstraction with nesting and mutual authentication

---

A partial function  $f : \{\!| C \!\!\}^k \mapsto C_1^\#, \dots, C_n^\#$ , with  $C_i^\# \in \{\text{Chal}_{N_i^\#}^{\ell, \ell'}(I^\#, J^\#, M_i^\#), \text{Resp}_{N_i^\#}^{\ell, \ell'}(I^\#, J^\#, M_i^\#)\}$  is an *encryption abstraction* if the following conditions hold:

*Format* -  $\text{names}(C) \subseteq ID$  and  $\text{names}(N_i^\#) = \text{names}(M_i^\#) = \emptyset$ ;  
 -  $x \in \text{vars}(C)$  if and only if  $x^\# \in \text{vars}(N_i^\#) \cup \text{vars}(M_i^\#)$ , for some  $i \in [1, n]$ ;

*Unique Abstraction* As in Table 19

*Nesting* As in Table 19

*Encryption* If  $f(\{\!| C \!\!\}^k) = C_1^\#, \dots, C_n^\#$ , with  $C_i^\# \in \{\text{Chal}_{N_i^\#}^{\ell, \ell'}(I^\#, J^\#, M_i^\#), \text{Resp}_{N_i^\#}^{\ell, \ell'}(I^\#, J^\#, M_i^\#)\}$  then

- either  $k = k_J^+$  and  $l \leq \text{Taint}$ ;
  - or  $k = k_I^-$  and  $l \leq \text{Int}$
  - or  $k \in \{k_{IJ}, k_{JI}\}$
-