Dynamic Types for Authentication *

Michele Bugliesi Riccardo Focardi
Matteo Maffei

Dipartimento di Informatica, Università Ca' Foscari di Venezia,
Via Torino 155, I-30172 Mestre (Ve), Italy
{michele,focardi,maffei}@dsi.unive.it

Abstract

We propose a type and effect system for *authentication* protocols built upon a tagging scheme that formalizes the intended semantics of ciphertexts. The main result is that the validation of each component in isolation is provably sound and *fully compositional*: if all the protocol participants are independently validated, then the protocol as a whole guarantees authentication in the presence of Dolev-Yao intruders possibly sharing long term keys with honest principals. Protocol are thus validated in the presence of both malicious outsiders and compromised insiders. The highly compositional nature of the analysis makes it suitable for multi-protocol systems, where different protocols might be executed concurrently.

1 Introduction

The importance of language-based security in the formal analysis of security protocols dates back to Abadi's seminal work [1] on *secrecy by typing*. Since then, a number of language-based techniques have been applied in the analysis of an increasingly large class of security protocols [2, 8, 9, 13, 14, 18, 20, 21, 34]. These approaches have the advantage of reasoning about security at the language level, thus clarifying *why* a message component is there and *how* security is achieved. This is particularly important for authentication protocols, where flaws are often originated by a certain degree of ambiguity in the encrypted messages circulated in the handshakes. Language-based reasoning on authentication is therefore particularly valuable, as it forces one to clarify protocol specifications by making explicit the underlying security mechanisms.

Authentication protocols are security protocols whose purpose is to enable two entities to achieve mutual and reliable agreement on some piece of information, typically

*Extended and revised version of [13]. Work partially supported by EU Contract IST-2001-32617 "Models and Types for Security in Mobile Distributed Systems" (MyThS), MIUR Project COFIN 2004013015 "Abstract Interpretation: Design and Applications" (AIDA), MIUR Project FIRB RBAU018RCZ "Abstract interpretation and model checking for the verification of embedded systems" (SPY-Mod) and MIUR Project 2005015785 "Logical Foundations of Distributed Systems and Mobile Code".

the identity of the other party, its presence, the origin of a message, its intended destination. Achieving the intended agreement guarantees is subtle because they typically are the result of the encryption/decryption of sequences of messages composed of different parts, with each part providing a "piece" of the authentication guarantee. To illustrate, consider the following example:

$$\begin{array}{ccc} A & & B \\ & \leftarrow n \text{ —} & \\ & \text{— } \{B,m,n\}_{\mathsf{Priv}(k_A)} \rightarrow & \end{array}$$

Here, and throughout, we write $\mathsf{Priv}(k_A)$ and $\mathsf{Pub}(k_A)$ to denote the private and the public key of entity A. Bob, the initiator, sends to Alice a random challenge (i.e., a nonce) n; Alice, the responder, signs the challenge together with a message m and Bob's identifier B; the protocol completes with the equality check on the nonce performed by B after decrypting the message with A's public key. The aim of the protocol is to guarantee to B that m *is authentic from* A, i.e., that m has been freshly sent by A to B. Each message component of the encrypted packet $\{B,m,n\}_{\mathsf{Priv}(k_A)}$ has a specific purpose in the protocol: the nonce n guarantees that the packet is fresh, i.e., it is not a replay of an old protocol session since n is used only once; the identifier B specifies the intended receiver of m; the signature guarantees that A is the sender. Collectively, the exchange of the three components provides the following *agreement* [29, 35] property: (i) the two participants agree on each other's identity, namely, B is guaranteed that its responder is A, as it appears to be, and dually, A engages a protocol session with initiator B; (ii) the two parties agree on the origin and the intended destination of the message m, i.e., that it was originated by A and intended for B.

Our present approach is based on the following methodology: (1) specify properties by annotating an executable specification of the protocol with correspondence assertions; (2) annotate the protocol with suitable types and tags; (3) verify the assertions by running a type checker.

This is the same methodology as the one proposed by Gordon and Jeffrey in [20, 21]. However, the technique we employ is different, as we rely on tags to annotate the ciphertexts exchanged in the protocol to assist the typed analysis. This is best illustrated with an example.

Consider again the simple authentication protocol we illustrated earlier. As we argued, the authenticity properties of the protocol are the result of the combined effect of specific guarantees conveyed by each of the components of the message $\{B,m,n\}_{\mathsf{Sign_A}}$. The system of [21] captures these guarantees by assigning the following type to the private key that signs the message:

$$\begin{array}{ll} \mathit{Key}(& B : \mathsf{Principal}, \\ & m : \mathsf{Payload}, \\ & n : \mathsf{Public\ Response}[\mathsf{end}\ A\ \textit{sending}\ m\ \textit{to}\ B] \quad) \end{array}$$

The structure of this dependent type renders the intended dependencies among the message components: in particular, the nonce type provides a complete specification of the role of the nonce in the protocol. The safety proof for the protocol is then, essentially,

a consequence of the typing rules guaranteeing that the protocol participants manipulate the nonce according to this intended usage. The fairly rich structure of the key and nonce types makes the approach very flexible, and expressive. On the other hand, there appear to be trade-offs. First, step (2) of the method (see above) may become rather complex, for choosing the correct types of nonces and keys may turn out to be non-trivial. Secondly, the degree of compositionality of the analysis seems to be slightly undermined by the very structure of the types, which essentially encode much of the structure of the protocol itself. As a consequence, it seems hardly possible to factor the authenticity properties proved for the protocol into corresponding properties established locally for each of the principals, independently of the context.

Our approach is different. We render the inter-dependencies among message components directly on terms rather than on their types, by imposing a richer, *tagged*, structure to our terms. We identify "minimal" set of authentication *patterns* based on corresponding forms of nonce handshakes, and make explicit in the encrypted messages the pattern they correspond to. For example, message $\{B,m,n\}_{\mathsf{Priv}(k_A)}$ is tagged as $\{\mathsf{Id}(B),\mathsf{Auth}(m),\mathsf{Verif}_{\mathsf{PC}}(n)\}_{\mathsf{Priv}(k_A)}$ to signal that the nonce n is authenticating m to the verifier B in a PC nonce handshake: PC stands for Public-Cipher as the nonce is sent out in clear and is received encrypted in a ciphertext. We will discuss nonce handshakes in detail in Section 2. Tags are dynamically identified by the recipient of a message, upon decryption, and employed to achieve/provide authentication guarantees. Intuitively, the information associated through dependent types in the Gordon-Jeffrey type system, is conveyed here in the structure of our tagged terms. This allows us to use simple dynamic types to just enforce the secrecy of keys and secret challenges, and simple effects to reason about authentication.
The advantages of our approach may be summarized as follows.

It is fully compositional: since the uniform use of tags is imposed by the typing rules, each party may be checked in isolation, i.e., using an independent typing environment. Indeed, all parties do share the same typing assumptions for keys, but such types only convey information on the principals holding/sharing the keys and, more importantly, this information is independent of the specific protocol that the entities will be running (hence, we do not need to include in the type assumptions for keys any information about the structure of the messages that will be encrypted with such keys). This is different from the Gordon-Jeffrey approach: as we noted above, in [20, 21] the types of the keys encode information on the protocol steps, so that different protocols between the same parties require different typings for the same keys or else, different keys.

This strong form of compositionality allows us to safely mix different protocols once their sequential components are type-checked; thus, our tagging discipline naturally scales to multi-protocol settings. Furthermore, the fact that tags correspond to a small set of a-priori selected patterns, makes the type system quite simple and easy to use; the human effort required is very small. Notably, even though the set of authentication patterns we have selected is small, it is expressive enough to capture many of the protocols in literature; this gives also new insights on which are the basic mechanisms for guaranteeing authentication.

The ρ-spi semantics provides the environment with long-term keys which are re-

garded as trusted by honest principals. Type-checked protocols guarantee authentication between honest principals even in presence of insider attacks, i.e., attacks carried out by enemies running protocol sessions with honest principals. Insider attacks are not considered in [20, 21].

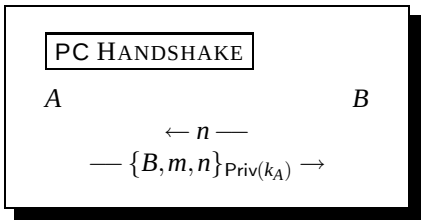
We remark that our safety results rely critically on the assumption that the messages exchanged in the protocol are tagged: hence our tags play a static as well a dynamic role, and the safety theorem assumes that the semantics of protocols is itself tagged. While this may be seen as a limitation, our tagging mechanism turns out to be less demanding than those employed to resolve message ambiguities in many existing protocol implementations and protocol analysis techniques (cf. Section 7).

Plan of the paper The rest of the paper is organized as follows: Section 2 overviews some basic concepts about authentication protocols. Section 3 illustrates the ρ-spi calculus and its operational semantics. In Section 4 we present our type and effect system and its main properties. In Section 5 we analyze the Splice/AS Protocol. Section 6 discusses how the type and effect system fits the analysis of multi-protocol systems. In Section 7 we conclude with final remarks and a discussion of other related work.

2 Nonce-Based Authentication Protocols

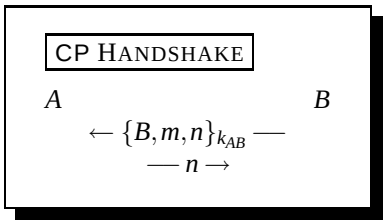
Challenge-and-response authentication protocols require time-variant parameters like, e.g., times-tamps or nonces to guarantee message freshness. Here we focus on nonce-based protocols: a nonce is a value (generally implemented as a random number) used in just one authentication exchange [33]. Suppose A (the *claimant*) wants to authenticate with B (the *verifier*). Nonce-based protocols may be classified into three categories depending on what is encrypted and what is sent in clear.

Plain-Cipher (PC) B sends out the nonce in clear and receives it back encrypted together with a message which is authenticated. A proves her identity to B by showing the knowledge of the encryption key. As an example, let us consider the protocol illustrated in Section 1:



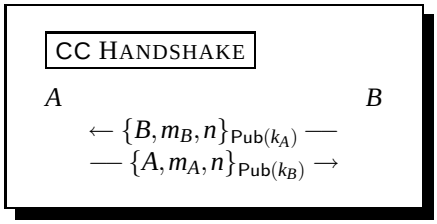
This protocol authenticates A sending message m to B, since only A may have generated the ciphertext. The same effect can be achieved in a symmetric cryptosystem using a shared key k_{AB} in place of $\mathsf{Priv}(k_A)$.

Cipher-Plain (CP) B sends out the nonce encrypted and receives it back in clear. A proves her identity to B by showing the knowledge of the decryption key. For example:



where k_{AB} is a symmetric key shared between A and B. This protocol authenticates A receiving message m from B, since only A may have decrypted the ciphertext. (The identifier B is used to "break" the symmetry of the key and avoid the so called *reflection* attacks: when decrypting the message, A knows that it is not a challenge she generated for B in a parallel protocol session). A similar effect is achieved with asymmetric keys, using $\mathsf{Pub}(k_A)$ in place of the symmetric key k_{AB}.

Cipher-Cipher (CC) The nonce is sent out and received back encrypted. This is useful if both entities want to exchange messages. A proves her identity to B by showing the knowledge of either the encryption key (as in PC) or the decryption key (as in CP). For example:



This protocol authenticates A sending message m_A to B and receiving message m_B from B, as only A may have decrypted the first ciphertext. Again, the same effect may be achieved using either the private keys $\mathsf{Priv}(k_B)$ and $\mathsf{Priv}(k_A)$, or a symmetric key k_{AB} in place of the two public keys used in the displayed narration.

Notice that the above mentioned categories are a generalization of POSH (Public Out Secret Home), SOPH (Secret Out Public Home) and SOSH (Secret Out Secret Home), introduced in [21]. We actually relax "Secret" into "Cipher" as handshakes may be composed of signed messages, guaranteeing integrity rather than secrecy. For example, as already noticed, PC includes protocols with a cleartext challenge and a signed response, which would not "fit well" into the POSH category.

3 The ρ-Spi Calculus

The ρ-spi calculus derives from the spi calculus [5], and inherits many of the features of *Lysa* [8], a version of the spi calculus proposed for the analysis of authentication protocols. ρ-spi differs from both calculi in several respects: it incorporates the notion of tagged message exchange from [12], it provides new authentication-specific constructs, and offers primitives for declaring process identities and keys.

Table 1 The syntax of ρ-spi calculus.

Notation: $TAG \in \{\mathsf{Id}, \mathsf{Auth}, \mathsf{Verif}_H, \mathsf{Claim}_H | H = \mathsf{PC}, \mathsf{CP}, \mathsf{CC?}, \mathsf{CC!}\}$
$m \in \mathcal{N} \cup \mathcal{V}$

$\mathcal{M}, \mathcal{K}$ ::= *Patterns*		P, Q ::= *Processes*	
a, b, k, n	names	$I \triangleright S$	(principal)
x, y, z	variables	$I \triangleright !S$	(replication)
$\mathsf{Pub}(m)$	public key	$P \mid Q$	(composition)
$\mathsf{Priv}(m)$	private key	let $k = \mathsf{sym\text{-}key}(I_1, I_2).P$	(symmetric-key assignment)
$TAG(\mathcal{M})$	tagged pattern	let $k = \mathsf{asym\text{-}key}(I).P$	(asymmetric-key assignment)
$(\mathcal{M}_1, \mathcal{M}_2)$	pair		

S ::= *Sequential Processes*	
$\mathbf{0}$	(nil)
$\mathsf{new}(n).S$	(restriction)
$\mathsf{in}(\mathcal{M}).S$	(input)
$\mathsf{out}(\mathcal{M}).S$	(output)
$\mathsf{encrypt}\{\mathcal{M}\}_{\mathcal{K}}$ as $x.S$	(symmetric encryption)
$\mathsf{encrypt}\{\lvert\mathcal{M}\rvert\}_{\mathcal{K}}$ as $x.S$	(asymmetric encryption)
decrypt x as $\{\mathcal{M}\}_{\mathcal{K}}.S$	(symmetric decryption)
decrypt x as $\{\lvert\mathcal{M}\rvert\}_{\mathcal{K}}.S$	(asymmetric decryption)
$\mathsf{begin}_{\mathcal{M}}(I_1, I_2, \mathcal{M}_1; \mathcal{M}_2).S$	(begin)
$\mathsf{end}_{\mathcal{M}}(I_1, I_2, \mathcal{M}_1; \mathcal{M}_2).S$	(end)

Table 2 PC Protocol in ρ-spi calculus

$$
\begin{array}{rcl}
\mathit{Protocol} & \triangleq & \mathsf{let}\ k_A = \mathsf{asym\text{-}key}(A)\ .\ (B \triangleright !\mathit{Initiator} \mid A \triangleright !\mathit{Responder}) \\
\mathit{Initiator} & \triangleq & \mathsf{new}(n).\mathsf{out}(n).\mathsf{in}(z). \\
 & & \quad \mathsf{decrypt}\ z\ \mathsf{as}\ \{\lvert \mathsf{Id}(B), \mathsf{Auth}(x), \mathsf{Verif}_{\mathsf{PC}}(n) \rvert\}_{\mathsf{Pub}(k_A)}.\mathsf{end}_n(B, A; x).\mathbf{0} \\
\mathit{Responder} & \triangleq & \mathsf{in}(x).\mathsf{new}(m).\mathsf{begin}_x(A, B; m). \\
 & & \quad \mathsf{encrypt}\ \{\lvert \mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{PC}}(x) \rvert\}_{\mathsf{Priv}(k_A)}\ \mathsf{as}\ z.\mathsf{out}(z).\mathbf{0}
\end{array}
$$

The syntax is reported in Table 1 and described below. Patterns, denoted by $\mathcal{M}, \mathcal{K}$, are recursively defined over names, variables, public and private keys, tagged patterns and pairs [1]. We presuppose two countable sets: $\mathcal{N}$ of names and $\mathcal{V}$ of variables. We reserve a, b, k, n for names and x, y, z for variables, with m ranging over both names and variables. The special name ε, used to denote the empty message, will be always omitted, e.g., primitive $\mathsf{begin}_{\varepsilon}(I_1, I_2, n; \varepsilon)$ will be written as $\mathsf{begin}(I_1, I_2, n;)$. Identities $\mathcal{ID}$ are a subset of names and are ranged over by I and J. Identities are further partitioned into *trusted principals* $\mathcal{ID}_{\mathcal{T}}$, ranged over A and B, and *enemies* $\mathcal{ID}_{\mathcal{E}}$, ranged over by E. The pair composed by a public key and the corresponding private one is noted by $\mathsf{Pub}(m)$, $\mathsf{Priv}(m)$, similarly to [5]. In the rest of the paper, we will use the following notation convention: $\overline{\mathsf{Pub}} = \mathsf{Priv}$ and vice-versa. Tags, denoted by *TAG*, are a special category of names. They specify the role of each message component. Specifically: all identifiers relevant to authentication are tagged by Id; messages that should be authenticated are tagged by Auth; finally, nonces are tagged by Verif_H or Claim_H, with $H \in \{\mathsf{PC}, \mathsf{CP}, \mathsf{CC!}, \mathsf{CC?}\}$. Such tags specify the role played by the entity tagged by Id (verifier or claimant) and the kind of nonce handshake. Notice that in CC nonce handshakes we distinguish challenge from response ciphertexts, denoted by CC? and CC!, respectively.

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

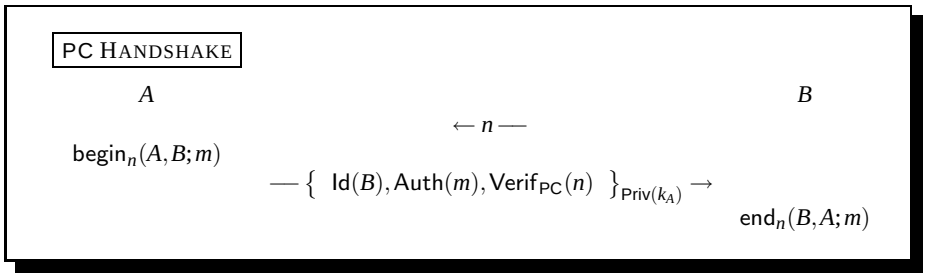
Sequential processes may never fork into parallel components: this assumption helps assign unique identities to sequential processes, and involves no significant loss of expressive power as protocol principals are typically specified as sequential processes, possibly sharing some long-term keys. The sequential process **0** is the null process that does nothing, as usual. Process $\mathsf{new}(n).S$ generates a fresh name n local to S. We presuppose a unique anonymous public channel, the network, from/to which all principals, including intruders, read/send messages. Similarly to *Lysa*, our input primitive may atomically test part of the message read, by pattern-matching. If the input message matches the pattern, then the variables occurring in the pattern are bound to the remaining sub-part of the message; otherwise the message is not read at all. For example, process '$\mathsf{in}(\mathsf{Claim}_{\mathsf{PC}}(x)).P$' may only read messages of the form $\mathsf{Claim}_{\mathsf{PC}}(M)$, binding x to M in P. Encryption just binds x to the encrypted message, while decryption checks if the message contained in x matches the form $\{\mathcal{M}\}_{\mathcal{K}}$ (or $\{|\mathcal{M}|\}_{\mathcal{K}}$), i.e., the payload matches $\mathcal{M}$ and is encrypted with the appropriate key. Only in this case x is decrypted and the variables in the pattern $\mathcal{M}$ get bound to the decrypted messages. Similarly to the input primitive, decryption may also test part of the decrypted messages by pattern-matching mechanism. Finally, the $\mathsf{begin}_{\mathcal{M}}(I_1, I_2, \mathcal{M}_1; \mathcal{M}_2).S$ and $\mathsf{end}_{\mathcal{M}}(I_2, I_1, \mathcal{M}_1; \mathcal{M}_2).S$ primitives are used to check the *correspondence assertions* [36]

[1] 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.

in a nonce handshake between I_1 and I_2 based on nonce $\mathcal{M}$. The former primitive declares that I_1 confirms the reception of message $\mathcal{M}_1$ and is willing to authenticate message $\mathcal{M}_2$ with I_2; the latter one indicates that I_2 gets confirmation from I_1 of the reception of message $\mathcal{M}_1$ and authenticates message $\mathcal{M}_2$.

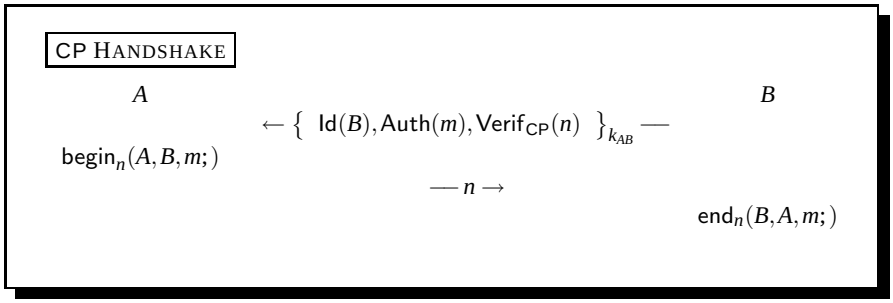
It is important to note that we make a distinction between *static* terms, or *patterns*, and dynamic terms, or *messages*. The former, noted $\mathcal{M}$, define the set of *syntactically legal* terms; the latter, noted M, N, K, define the set of terms that may arise at run time and are formalized in Table 12 of Appendix A. The difference is that messages may have possibly nested encryptions, while patterns may not: notice, to this regard, that in the syntax of processes, encryptions may only be formed by means of the encrypt prefix.

Example 1 To illustrate the use of tags and correspondence assertions, let us consider the protocols presented so far. The first protocol in Section 1 can be decorated with tags and correspondence assertions as follows:



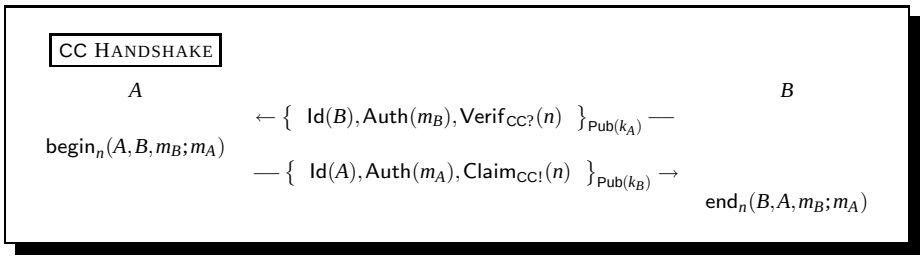
The tagged structure specifies that that the nonce n is used by the verifier B for authenticating m with A in a PC nonce handshake. According to the correspondence assertions, at the end of the protocol B authenticates A *sending* message m.

The second protocol in Section 2 can be decorated as follows:



Indeed, the nonce n is used by the verifier B for authenticating m with A in a CP nonce handshake. According to the correspondence assertions, at the end of the protocol B authenticates A *receiving* message m.

Finally, the third protocol in Section 2 can be decorated as follows:



Notice that the use of ? and ! in the nonce tag disambiguates whether the ciphertext is used in a CC handshake as challenge or response. By an inspection of the correspondence assertions, at the end of the protocol B authenticates A receiving message m_B and sending message m_A.

■

Example 2 To illustrate the ρ-spi calculus syntax, let us consider the first protocol in Example 1. The ρ-spi calculus specification is in Table 2. After declaring the key pair for A, an unbounded number of instances of B as initiator and an unbounded number of instances of A as responder are run in parallel. The *Initiator B* generates a fresh nonce and sends it in clear on the network. Then it reads a message from the network and tries to decrypt it with the public key of the responder A and checks that its own identifier tagged by Id is the first component of the message payload and the nonce tagged by $\mathsf{Verif}_{\mathsf{PC}}$ is fresh, i.e., it is the one just generated. If this is the case, B authenticates A sending message x, namely the message tagged by Auth. The *Responder A* receives a nonce from the network, generates a new message m, declares the start of the session with the initiator B for authenticating message m, signs m together with the responder identifier and the nonce and sends the obtained ciphertext on the network. Notice that all message components are tagged.

■

Operational Semantics. We define the operational semantics of ρ-spi in terms of *traces*, after [10]. 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, 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 in the trace.

Principals do not directly synchronize with each other. Instead, they may receive from the unique channel an arbitrary message M known by the environment, which models the Dolev-Yao intruder: the knowledge of the environment is formalized by a set of deduction rules stating that the environment knows all the identity labels, the messages sent on the network, the content of ciphertexts whose decryption key is known, ciphertexts created by its knowledge and all the keys declared as owned by E together with all the public keys. Finally, the environment may also create new fresh names. Even though this approach forces us to define the Dolev-Yao rules explicitly, rather than relying on the standard reduction rules, it allows us to give typing rules

only for well-behaved processes, thus simplifying the resulting type and effect system (cf. Section 4). Furthermore, our notion of safety introduced below does not need to consider the possibility of events performed by the attacker, since they are not syntactically possible. For more detail on the transition relation and the deductive system formalizing the knowledge of the environment, please refer to Appendix A.

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 [29, 36] 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 $s = s_1 :: end_n(B,A,M_1;M_2) :: s_2$, then $s_1 = s_1' :: begin_n(A,B,M_1;M_2) :: s_1''$, and $s_1' :: s_1'' :: s_2$ is safe. A process P is safe if, $\forall s \in T(P), s$ is safe.*

A trace is *safe* if every $end_n(B,A,M_1;M_2)$ is preceded by a distinct $begin_n(A,B,M_1;M_2)$. This definition is a natural extension of the standard notion of safety (cf.[20]) to a setting where enemies are provided with keys regarded as trusted by other participants. intuitively, this guarantees that whenever B authenticates A receiving M_1 and sending M_2, then A has received M_1 and has sent M_2 in a handshake with B based on nonce n.

Example 3 To illustrate the semantics of the calculus and the notion of safety, let us consider a flawed simplification of the first protocol of Example 1, obtained by eliminating the nonce.

$$\begin{array}{cc} A & B \\ \multicolumn{2}{c}{\text{—} \{\mathsf{Id}(B), \mathsf{Auth}(m)\}_{\mathsf{Priv}(k_A)} \rightarrow} \end{array}$$

We report the ρ-spi calculus specification below, decorated with tags and correspondence assertions:

$$\begin{array}{ll} \mathit{Initiator}_{\mathit{flawed}} & \triangleq \mathsf{in}(z).\mathsf{decrypt}\ z\ \mathsf{as}\ \{|\mathsf{Id}(B), \mathsf{Auth}(x)|\}_{\mathsf{Pub}(k_A)}.\mathsf{end}(B,A;x).\mathbf{0} \\ \mathit{Responder}_{\mathit{flawed}} & \triangleq \mathsf{new}(m).\mathsf{begin}(A,B;m).\mathsf{encrypt}\ \{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)}\ \mathsf{as}\ z.\mathsf{out}(z).\mathbf{0} \end{array}$$

This protocol suffers of the following standard replay attack, where E impersonates A by just replaying a previously intercepted message:

$$\begin{array}{lcll} A & \rightarrow & B: & \{\mathsf{Id}(B), \mathsf{Auth}(m)\}_{\mathsf{Priv}(k_A)} \\ E(A) & \rightarrow & B: & \{\mathsf{Id}(B), \mathsf{Auth}(m)\}_{\mathsf{Priv}(k_A)} \end{array}$$

The attack mentioned above corresponds to the following execution trace:

$$\begin{array}{l} \mathit{asym-key}(k_A,A) :: \mathit{new}(m) :: \mathbf{begin}(A,B;m) :: \\ \mathit{encrypt}\{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)} :: \mathit{out}(\{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)}) :: \\ \mathit{in}(\{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)}) :: \mathit{decrypt}\{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)} :: \mathbf{end}(B,A;m) :: \\ \mathit{in}(\{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)}) :: \mathit{decrypt}\{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)} :: \mathbf{end}(B,A;m) \end{array}$$

Notice that the same message $\{|\mathsf{Id}(B), \mathsf{Auth}(m)|\}_{\mathsf{Priv}(k_A)}$ is read twice by two different instances of the Responder. This causes two ends with just one begin, thus making this trace unsafe. The presence of the nonce, as discussed in the Introduction, repairs the protocol avoiding this replay attack.

■

4 The ρ-Spi Type and Effect System

Our analysis is based on types, regulating the use of nonces and keys, and effects, tracking protocol behaviour. More specifically, types are mainly used to check that the secrecy of keys and nonces is preserved: processes should never leak long-term keys and secret nonces generated for CP and CC encrypted challenges, i.e., nonces that the claimant is challenged to decrypt for confirming its own identity. Effects, instead, keep track of nonce freshness, challenges and responses. For example, the generation of a new nonce is tracked by a specific effect which is removed as soon as the nonce is checked by the verifier. In this way, we obtain the basic property of nonces that should be used only once, i.e., only for a single protocol session. The link between types and effects is established through the type of encrypted messages, which specifies, in the form of effects, whether the encrypted message is a challenge or a response. This information is derived by the type of the key and by the tags included in the encrypted message. Through key types and tags it is possible for the protocol principals to remotely agree on the encrypted message types, thus allowing a local and compositional validation in which the only a-priori shared knowledge is the type of long-term keys.

Our type and effect system is based on some basic notions and rules (Section 4.1). The analysis exploits the tags for message components, compiling them into static types for ciphertexts (Section 4.2). Typing invariants based on all of the above described nonce handshakes (Section 4.3) draw the typing rules for processes (Section 4.4). Finally, the type and effect system is proved to be sound and strongly compositional (Section 4.5).

4.1 Basic Notions and Rules

In this section, we introduce the basic notions of our analysis: typing judgements, types and effects, subtyping and typed ρ-spi calculus.

4.1.1 Judgements

The type and effect system relies on the following judgements:

Γ	$\vdash \diamond$	(Good Environment)
Γ	$\vdash M : T$	(Message Type)
Γ	$\vdash P : e$	(Process Type)
$I;\Gamma$	$\vdash S : e$	(Sequential Process Type)

Table 3 Types definitions and basic typing rules

Notation: $\ell \in \{Un, Ciph, Priv, Int\}$
In $[?|!]\mathsf{Chal}^{\ell}_{N}(I,J,M)$ and $[?|!]\mathsf{Resp}^{\ell}_{N}(I,J,M)$, $\ell = Un \Rightarrow M = ()$.

Types

$T ::=$	$\mathsf{SharedKey}(I,J)$	symmetric key
	$\mathsf{Key}(I)$	asymmetric seed
	$\mathsf{PublicKey}(I)$	public key
	$\mathsf{PrivateKey}(I)$	private key
	Un	untrusted
	$\mathsf{Nonce}^{\ell}(I,J)$	nonce
	$\mathsf{Enc}(e;f)$	ciphertext

Atomic Effects

$t ::=$	$\mathsf{fresh}^{\ell}(n)$	freshness
	$[?\|!]\mathsf{Chal}^{\ell}_{N}(I,J,M)$	challenge
	$[?\|!]\mathsf{Resp}^{\ell}_{N}(I,J,M)$	response

Effects

$e, f ::=$	$[t_1, \ldots, t_n]$	multiset of atomic effects

Typing Environment

EMPTY ENV
$$\varnothing \vdash \diamond$$

GOOD ENV
$$\frac{\Gamma \vdash \diamond \qquad m \notin dom(\Gamma) \qquad fn(T) \subseteq dom(\Gamma)}{\Gamma, m : T \vdash \diamond}$$

PROJECTION
$$\frac{\Gamma, a : T, \Gamma' \vdash \diamond}{\Gamma, a : T, \Gamma' \vdash a : T}$$

Typing Rules for Asymmetric Keys

PUBLIC KEY
$$\frac{\Gamma \vdash k : \mathsf{Key}(I)}{\Gamma \vdash \mathsf{Pub}(k) : \mathsf{PublicKey}(I)}$$

PRIVATE KEY
$$\frac{\Gamma \vdash k : \mathsf{Key}(I)}{\Gamma \vdash \mathsf{Priv}(k) : \mathsf{PrivateKey}(I)}$$

Typing Rules for Untrusted Terms
$\mathsf{Key} \in \{\mathsf{Pub}, \mathsf{Priv}\}$.

UN SYMM CIPH
$$\frac{\Gamma \vdash M : \mathsf{Un} \qquad \Gamma \vdash K : \mathsf{Un}}{\Gamma \vdash \{M\}_K : \mathsf{Un}}$$

UN ASYMM CIPH
$$\frac{\Gamma \vdash M : \mathsf{Un} \qquad \Gamma \vdash \mathsf{Key}(K) : \mathsf{Un}}{\Gamma \vdash \{|M|\}_{\mathsf{Key}(K)} : \mathsf{Un}}$$

UNTRUSTED PAIR
$$\frac{\Gamma \vdash M_1 : \mathsf{Un} \qquad \Gamma \vdash M_2 : \mathsf{Un}}{\Gamma \vdash (M_1, M_2) : \mathsf{Un}}$$

UN KEY PAIR
$$\frac{\Gamma \vdash K : \mathsf{Un}}{\Gamma \vdash \mathsf{Key}(K) : \mathsf{Un}}$$

UN TAG
$$\frac{\Gamma \vdash M : \mathsf{Un}}{\Gamma \vdash TAG(M) : \mathsf{Un}}$$

The judgement $\Gamma \vdash \diamond$ is read as "the typing environment Γ is well-formed", i.e., Γ is an ordered set of bindings between names/variables and types such that Γ is a function and all the names referred by types in Γ have a type in Γ. Judgement $\Gamma \vdash M : T$, read as "M has type T in Γ", means that either M is bound to T in Γ or type T for M is derivable by the type bindings in Γ.

The main judgement for our approach is $\Gamma \vdash P : e$, symbolizing that process P can be typed under Γ with the effect e. This judgement relies on the similar one for sequential processes $I; \Gamma \vdash S : e$, in which I is the identity of the principal running S. As mentioned above, Γ regulates the use of terms like keys and nonces, while e is used to track relevant events assumed to happen before P. For example, sequential process $\mathsf{out}(k).S$ can never be typed if k has a long-term key type in Γ, as long-term keys cannot be leaked. Moreover, $\mathsf{end}_n(A, B, m_1; m_2).S$ type-checks only when the effect requiring the freshness of n is included in e, since a nonce can be used for concluding just one protocol session.

4.1.2 Types and Effects

The definitions of the types and effects as well as the rules for deriving basic judgments are in Table 3. A long-term key shared between I and J has type $\mathsf{SharedKey}(I, J)$ and it can only be used by I and J. A name used for creating a key pair owned by I has type $\mathsf{Key}(I)$; the public and private keys generated by that name have type $\mathsf{PublicKey}(I)$ and $\mathsf{PrivateKey}(I)$, respectively. While private keys can only be used by their owners, public keys are available to every principal. Every untagged term potentially known by the enemy has type Un. A nonce used by I and J has type $\mathsf{Nonce}^{\ell}(I, J)$, where the label $\ell \in \{Un, Ciph, Priv, Int\}$ specifies secrecy and integrity properties of the nonce: it is *Un* when the nonce is sent in clear on the network (in PC challenges and CP responses), *Ciph* when the nonce is sent encrypted but it is supposed to be sent back in clear (CP challenges), *Priv* when the nonce secrecy is preserved by the handshake (CC handshakes using public or symmetric keys) and *Int* when the integrity of the nonce is guaranteed by signing it (CC handshakes using private keys). Ciphertexts generated or decrypted by trusted principals are regulated by type $\mathsf{Enc}(e; f)$. The effects e and f depend on the ciphertext semantics. If the ciphertext is a challenge sent by I to J containing the nonce N and the message M, then $e = [\mathsf{Chal}^{\ell}_N(I, J, M)]$, otherwise $e = []$. Similarly, if the ciphertext is a response sent by I to J containing the nonce N and the message M, then $f = [\mathsf{Resp}^{\ell}_N(I, J, M)]$, otherwise $f = []$. We use square brackets to denote both multisets of atomic effects (as done above) and optional arguments: for example, $[?|!]\mathsf{Chal}^{\ell}_N(I, J, M)$ denotes the three possible atomic effects $\mathsf{Chal}^{\ell}_N(I, J, M)$, $!\mathsf{Chal}^{\ell}_N(I, J, M)$ and $?\mathsf{Chal}^{\ell}_N(I, J, M)$

The label ℓ has the same semantics as above. The intuitive meaning of the remaining effects is given below:

- The atomic effect $fresh^{\ell}(n)$ tracks the freshness of the nonce n. A nonce is fresh if it is a new name and it has not been used for justifying an end event yet.
- The atomic effect $?\mathsf{Chal}^{\ell}_N(I, J, M)$ tracks the decryption of a ciphertext representing a challenge, containing the nonce N and the message M, sent by I to J. Similarly for the atomic effect $?\mathsf{Resp}^{\ell}_N(I, J, M)$.

Table 4 Subtyping rules

$$\textsc{Subsumption}\quad \frac{\Gamma \vdash N : T' \qquad T' <: T}{\Gamma \vdash N : T}$$

$\mathsf{Enc}(e_C; e_R) <: \mathsf{Un}$		(Trusted Ciphertext)
$\mathsf{Un} <: \mathsf{Nonce}^{\ell}(I,J)$	$\ell \in \{Priv, Ciph\}$	(Tainted Nonce)
$\mathsf{Nonce}^{Int}(I,J) <: \mathsf{Un}$		(Public Nonce)
$\mathsf{PublicKey}(I) <: \mathsf{Un}$		(Public Key)
$T <: \mathsf{Un}$ and $\mathsf{Un} <: T$	$T \neq \mathsf{Enc}(e; f) \quad fn(T) \cap \mathcal{ID}_{\mathcal{I}} \neq \emptyset$	(Public Name)

- The atomic effect $!\mathsf{Chal}^{\ell}_{N}(I,J,M)$ tracks the generation of a ciphertext representing a challenge, containing the nonce N and the message M, sent by I to J.
- The atomic effect $!\mathsf{Resp}^{\ell}_{N}(I,J,M)$ has different semantics as it *enables* the generation of a ciphertext representing a response, containing the nonce N and the message M, sent by I to J. This asymmetry in the use of effects is discussed in Section 4.3.

4.1.3 Typing Environment and Basic Rules

Well-Formedness conditions are mostly standard. By (EMPTY ENV), the empty environment is well-formed. An environment $\Gamma, m : T$ is well-formed (GOOD ENV) only if Γ is well-formed, the type T depends on names defined in Γ and m does not belong to the domain of Γ. Notice that dependent types require the typing environment to be ordered. Names and variables always occur free in types and effects.

The typing rules for public and private keys are straightforward. UN SYMM CIPH and UN ASYMM CIPH give the type Un to ciphertexts built by untrusted terms, thus typing ciphertexts generated by the enemy. Similarly, UNTRUSTED PAIR says that a tuple of untrusted terms has type Un, UN KEY PAIR gives type Un to asymmetric keys built upon untrusted terms and UN TAG says that tagged untrusted terms are of type Un. The typing rules for $\mathsf{Enc}(e; f)$, regulating encryptions and decryptions performed by trusted principals, are discussed in Section 4.2.

4.1.4 Subtyping

Subtyping introduces the partial order $T' <: T$ among types, meaning that T' is "more specific" than T, i.e., what can be done with a term of type T can be done even with a term of the more specific type T'. Rule SUBSUMPTION of Table 4 formalizes the above fact, by stating that if N has type T' in Γ and $T' <: T$ then N has also type T in Γ. The intuition behind the subtyping relation is illustrated below: (Trusted Ciphertext) gives type Un to ciphertexts generated by trusted principals, as they can be sent on the untrusted network. To motivate (Tainted Nonce), let us consider the asymmetric version of the second protocol in Example 1, using a CP nonce handshake:

$$\begin{array}{cc} A & B \\ \leftarrow \{|\mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{CP}}(n)|\}_{\mathsf{Pub}(k_A)} & \text{---} \\ \text{---} & n \rightarrow \end{array}$$

Notice that A does not know whether the ciphertext has been created by B or by the enemy since it is encrypted with A's public key and the enemy knows all the public keys. In the former case, the type of the nonce n is $\mathsf{Nonce}^{Ciph}(A,B)$, in the latter it is Un. Hence, at run-time, the variable with type $\mathsf{Nonce}^{Ciph}(A,B)$ might be substituted by a term with type Un. The subtyping rule (Tainted Nonce) addresses this issue. Signed nonces may be read by the enemy thus the type $\mathsf{Nonce}^{Int}(I,J)$ is subtype of Un. Since the enemy knows all the public keys, $\mathsf{PublicKey}(I)$ is subtype of Un. Finally, the enemy is provided with long-term keys regarded as trusted by other principals so that he may start authentication sessions pretending to be trusted. During such protocol sessions, the enemy may get knowledge of nonces generated by trusted principals: for instance, A might generate a nonce with type $\mathsf{Nonce}^{Priv}(A,E)$ for starting a CC handshake with the enemy. For this reason, by (Public Name), nonce and key types depending on an enemy identity label are subtypes of Un and vice-versa.

Notice that subtyping is never used in the protocol validation. Instead, it is required in the proof of type preservation at run-time.

4.1.5 Typed Calculus and Notation

For easing the presentation, we have given so far an untyped version of the restriction. Moreover, a cast operator is needed by the type and effect system, as discussed in Section 4.4. Formally, the syntax of sequential processes is extended as follows:

S	::=	$\ldots$	as in Table 1
		$\mathsf{new}(n:T).S$	(typed restriction)
		$\mathsf{cast}\ \mathcal{M}\ \mathsf{is}\ (x:T).S$	(cast operator)

The type in the restriction as well as the cast operator have no computational import. (for more details see Appendix A.) For sake of readability we assume a number of type and effect equalities:

$\mathsf{Un} = \mathsf{Nonce}^{Un}(I,J)$	(Untrusted Nonce)
$\mathsf{SharedKey}(J,I) = \mathsf{SharedKey}(I,J)$	(Symmetric Keys)
$\mathsf{Chal}^{\ell}_{(N_1,\ldots,N_n)}(M) = \mathsf{Chal}^{\ell}_{N_1}(M),\ldots,\mathsf{Chal}^{\ell}_{N_n}(M)$	(Multiple Challenge)
$\mathsf{Resp}^{\ell}_{(N_1,\ldots,N_n)}(M) = \mathsf{Resp}^{\ell}_{N_1}(M),\ldots,\mathsf{Resp}^{\ell}_{N_n}(M)$	(Multiple Response)
$\mathsf{Chal}^{Un}_{N}() = \mathsf{Chal}^{Un}_{N}(I,J)$	(Untrusted Challenge)
$\mathsf{Resp}^{Un}_{N}() = \mathsf{Resp}^{Un}_{N}(I,J)$	(Untrusted Response)

Notice the last two equalities introduce two new atomic effects: $\mathsf{Chal}^{Un}_{N}()$ and $\mathsf{Resp}^{Un}_{N}()$. In fact, identity labels may be omitted as a nonce sent in clear does not convey any information regarding principals involved in the handshake. Type and effect equalities are used in the typing of messages and processes, respectively, by the following rules:

$$\textsc{Type Equality}\ \frac{\Gamma \vdash M : T' \qquad T' = T}{\Gamma \vdash M : T} \qquad \textsc{Effect Equality 1}\ \frac{\Gamma \vdash P : e' \qquad e' = e}{\Gamma \vdash P : e} \qquad \textsc{Effect Equality 2}\ \frac{I;\Gamma \vdash S : e' \qquad e' = e}{I;\Gamma \vdash S : e}$$

4.2 Typing Trusted Ciphertexts

In this section, we illustrate the typing rules of Table 5 regulating ciphertexts generated or decrypted by trusted principals. In the following, the order of the terms in a ciphertext is immaterial and we write _ to denote a tuple of untagged terms. Intuitively, when generating or decrypting messages of the form $\{\mathsf{Id}(I), R_H(N), \mathsf{Auth}(M), \_\}_K$, the tagged structure is "compiled" into static ciphertext types of the form $\mathsf{Enc}(e_C; e_R)$. This *dynamic typing* is defined by the function $\mathit{Typeofenc}(I, R_H, N, M, T_K)$ taking as input the entity I tagged by Id, the nonce tag R_H, the nonce N, the authenticated message M and the type of the encryption key T_K. This function yields the tuple (e_C, e_R, T_N), representing the challenge and response effects e_C and e_R associated to the ciphertext and the type of the nonce T_N.

Function *Typeofenc* is defined through a table whose rows are indexed by the kind of handshake H and columns by the key type T_K. Once a box is identified by the row and the column, i.e., by H and T_K, the value of R is checked and, based on it, a tuple (e_C, e_R, T_N) is returned. Notice that for shared keys, both cases for R, i.e., Verif and Claim, are covered. For public and private keys, instead, only one value of R is possible for each box. Intuitively, this is due to the fact that symmetric cryptography requires that either the claimant or the verifier is specified in the message in order to disambiguate the, intrinsically symmetric, use of the key. In asymmetric cryptography, instead, one of the principals is specified through the key and the other one is included in the message along with its specific role. If either the identified box is empty or the constraint on R does not hold, then *Typeofenc* is undefined.

CIPHERTEXT gives ciphertexts type $\mathsf{Enc}(e_C; e_R)$ if the function *Typeofenc* yields the tuple (e_C, e_R, T_N) and the nonce has type T_N. Next examples show the use of this rule and illustrate the intuition behind *Typeofenc*.

Example 4 Consider the protocols of Example 1. The first one can be typed as follows:

PC HANDSHAKE

A		B
	$\leftarrow n \text{—}$	
$\mathsf{begin}_n(A,B;m)$		
	$\text{—} \{\ \mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{PC}}(n)\ \}_{\mathsf{Priv}(k_A)} \rightarrow$	
		$\mathsf{end}_n(B,A;m)$

$$\Gamma \vdash \begin{array}{l} A : \mathsf{Un}, B : \mathsf{Un}, m : \mathsf{Un}, n : \mathsf{Un} \\ \mathsf{Priv}(k_A) : \mathsf{PrivateKey}(A) \\ \{|\mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{PC}}(n)|\}_{\mathsf{Priv}(k_A)} : \mathsf{Enc}([]; [\mathsf{Resp}_n^{Ciph}(A,B,m)]) \end{array}$$

Table 5 Typing rules for trusted ciphertexts

$\mathit{Typeofenc}(I, R_H, N, M, T_K)$

H \ T_K	$\mathsf{SharedKey}(I,J)$ *or* $\mathsf{SharedKey}(J,I)$	$\mathsf{PrivateKey}(J)$	$\mathsf{PublicKey}(J)$
PC	$\dfrac{\textit{if } R = \mathsf{Verif}}{[],\ [\mathsf{Resp}_N^{Ciph}(J,I,M)],\ \mathsf{Un}}$ | $\dfrac{\textit{if } R = \mathsf{Claim}}{[],\ [\mathsf{Resp}_N^{Ciph}(I,J,M)],\ \mathsf{Un}}$	$\dfrac{\textit{if } R = \mathsf{Verif}}{[],\ [\mathsf{Resp}_N^{Ciph}(J,I,M)],\ \mathsf{Un}}$	
CP	$\dfrac{\textit{if } R = \mathsf{Verif}}{[\mathsf{Chal}_N^{Ciph}(I,J,M)],\ [],\ \mathsf{Nonce}^{Ciph}(I,J)}$ | $\dfrac{\textit{if } R = \mathsf{Claim}}{[\mathsf{Chal}_N^{Ciph}(J,I,M)],\ [],\ \mathsf{Nonce}^{Ciph}(J,I)}$		$\dfrac{\textit{if } R = \mathsf{Verif}}{[\mathsf{Chal}_N^{Ciph}(I,J,M)],\ [],\ \mathsf{Nonce}^{Ciph}(I,J)}$
CC?	$\dfrac{\textit{if } R = \mathsf{Verif}}{[\mathsf{Chal}_N^{Priv}(I,J,M)],\ [],\ \mathsf{Nonce}^{Priv}(I,J)}$ | $\dfrac{\textit{if } R = \mathsf{Claim}}{[\mathsf{Chal}_N^{Priv}(J,I,M)],\ [],\ \mathsf{Nonce}^{Priv}(J,I)}$	$\dfrac{\textit{if } R = \mathsf{Claim}}{[\mathsf{Chal}_N^{Int}(J,I,M)],\ [],\ \mathsf{Nonce}^{Int}(J,I)}$	$\dfrac{\textit{if } R = \mathsf{Verif}}{[\mathsf{Chal}_N^{Priv}(I,J,M)],\ [],\ \mathsf{Nonce}^{Priv}(I,J)}$
CC!	$\dfrac{\textit{if } R = \mathsf{Verif}}{[],\ [\mathsf{Resp}_N^{Priv}(J,I,M)],\ \mathsf{Nonce}^{Priv}(I,J)}$ | $\dfrac{\textit{if } R = \mathsf{Claim}}{[],\ [\mathsf{Resp}_N^{Priv}(I,J,M)],\ \mathsf{Nonce}^{Priv}(J,I)}$	$\dfrac{\textit{if } R = \mathsf{Verif}}{[],\ [\mathsf{Resp}_N^{Int}(J,I,M)],\ \mathsf{Nonce}^{Int}(I,J)}$	$\dfrac{\textit{if } R = \mathsf{Claim}}{[],\ [\mathsf{Resp}_N^{Priv}(I,J,M)],\ \mathsf{Nonce}^{Priv}(J,I)}$

CIPHERTEXT

$$\frac{\Gamma \vdash M, \_ : \mathsf{Un} \qquad \Gamma \vdash N : T_N \qquad \Gamma \vdash K : T_K \qquad (e_C, e_R, T_N) = \mathit{Typeofenc}(I, R_H, N, M, T_K)}{\begin{array}{c}\Gamma \vdash \{\mathsf{Id}(I), R_H(N), \mathsf{Auth}(M), \_\}_K : \mathsf{Enc}(e_C; e_R) \\ \Gamma \vdash \{|\mathsf{Id}(I), R_H(N), \mathsf{Auth}(M), \_|\}_K : \mathsf{Enc}(e_C; e_R)\end{array}}$$

UNTAGGED CIPHERTEXT

$$\frac{\Gamma \vdash \_ : \mathsf{Un} \qquad \Gamma \vdash K : T}{\Gamma \vdash \{\_\}_K : \mathsf{Enc}([];[]) \qquad \Gamma \vdash \{|\_|\}_K : \mathsf{Enc}([];[])}$$

CHALLENGE-RESPONSE

$$\frac{H \in \{\mathsf{CP}, \mathsf{CC?}\} \qquad H' \in \{\mathsf{PC}, \mathsf{CC!}\} \qquad \Gamma \vdash \{\mathsf{Id}(I), \mathsf{Auth}(M), R_H(N), \_\}_K : \mathsf{Enc}(e_C; \cdot) \qquad \Gamma \vdash \{\mathsf{Id}(I), \mathsf{Auth}(M), R'_{H'}(N'), \_\}_K : \mathsf{Enc}(\cdot; e_R)}{\Gamma \vdash \{\mathsf{Id}(I), \mathsf{Auth}(M), R_H(N), R'_{H'}(N'), \_\}_K : \mathsf{Enc}(e_C; e_R)}$$

The key $\mathsf{Priv}(k_A)$ is A's private key, thus having type $\mathsf{PrivateKey}(A)$ (rule PRIVATE KEY, Table 3). As mentioned before, the nonce tag $\mathsf{Verif}_{\mathsf{PC}}$ indicates that the ciphertext is used as a response in a PC handshake. Typing rule CIPHERTEXT computes $\mathit{Typeofenc}(B, \mathsf{Verif}_{\mathsf{PC}}, n, m, \mathsf{PrivateKey}(A)) = ([], [\mathsf{Resp}_n^{\mathit{Ciph}}(A,B,m)], \mathsf{Un})$, according to the box in the first row (PC) and second column ($\mathsf{PrivateKey}(A)$), given that the role of B is Verif. Thus, assumed that the nonce n has type Un, the ciphertext is typed $\mathsf{Enc}([]; [\mathsf{Resp}_n^{\mathit{Ciph}}(A,B,m)])$ symbolizing that it is an encrypted response from A to B authenticating m and based on nonce n. Similarly, the second protocol can be typed as follows:

CP HANDSHAKE

A		B
	$\leftarrow \{\ \mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{CP}}(n)\ \}_{k_{AB}}$ —	
$\mathsf{begin}_n(A,B,m;)$		
	— n →	
		$\mathsf{end}_n(B,A,m;)$

$$\begin{aligned}\Gamma \vdash\ & A : \mathsf{Un}, B : \mathsf{Un}, m : \mathsf{Un} \\ & k_{AB} : \mathsf{SharedKey}(A,B) \\ & n : \mathsf{Nonce}^{\mathit{Ciph}}(B,A) \\ & \{\mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{CP}}(n)\}_{k_{AB}} : \mathsf{Enc}([\mathsf{Chal}_n^{\mathit{Ciph}}(B,A,m)]; [])\end{aligned}$$

The nonce tag $\mathsf{Verif}_{\mathsf{CP}}$ and the key type $\mathsf{SharedKey}(A,B)$ indicate that the ciphertext is used in a symmetric key CP challenge. $\mathit{Typeofenc}(B, \mathsf{Verif}_{\mathsf{CP}}, n, m, \mathsf{SharedKey}(A,B)) = ([\mathsf{Chal}_n^{\mathit{Ciph}}(B,A,m)], [], \mathsf{Nonce}^{\mathit{Ciph}}(B,A))$ can be computed by looking at the second row (CP) and first column ($\mathsf{SharedKey}(A,B)$), case $R = \mathsf{Verif}$. Thus the ciphertext has type $\mathsf{Enc}([\mathsf{Chal}_n^{\mathit{Ciph}}(B,A,m)]; [])$, given that n has type $\mathsf{Nonce}^{\mathit{Ciph}}(B,A)$. This type correctly identifies the ciphertext as an encrypted challenge from B to A authenticating m and based on nonce n. We finally give the typed version of the third protocol, leaving as an exercise to the interested reader the derivation of types.

CC HANDSHAKE

A		B
	$\leftarrow \{\mathsf{Id}(B), \mathsf{Auth}(m_B), \mathsf{Verif}_{\mathsf{CC?}}(n)\}_{\mathsf{Pub}(k_A)}$ —	
$\mathsf{begin}_n(A,B,m_B;m_A)$		
	— $\{\mathsf{Id}(A), \mathsf{Auth}(m_A), \mathsf{Claim}_{\mathsf{CC!}}(n)\}_{\mathsf{Pub}(k_B)}$ →	
		$\mathsf{end}_n(B,A,m_B;m_A)$

$$\begin{aligned}\Gamma \vdash\ & A : \mathsf{Un}, B : \mathsf{Un}, m_B : \mathsf{Un}, m_A : \mathsf{Un} \\ & \mathsf{Pub}(k_A) : \mathsf{PublicKey}(A), \\ & \mathsf{Pub}(k_B) : \mathsf{PublicKey}(B) \\ & n : \mathsf{Nonce}^{\mathit{Priv}}(B,A) \\ & \{|\mathsf{Id}(B), \mathsf{Auth}(m_B), \mathsf{Verif}_{\mathsf{CC?}}(n)|\}_{\mathsf{Pub}(k_A)} : \mathsf{Enc}([\mathsf{Chal}_n^{\mathit{Priv}}(B,A,m_B)]; []) \\ & \{|\mathsf{Id}(A), \mathsf{Auth}(m_A), \mathsf{Claim}_{\mathsf{CC!}}(n)|\}_{\mathsf{Pub}(k_B)} : \mathsf{Enc}([]; [\mathsf{Resp}_n^{\mathit{Priv}}(A,B,m_A)])\end{aligned}$$

The remaining rules of Table 5 regulate ciphertexts containing no tagged messages and ciphertexts used both as challenges and responses. The former are typed as $\mathsf{Enc}([];[])$, by UNTAGGED CIPHERTEXT, since they represent neither challenges nor responses. The latter, instead, are typed $\mathsf{Enc}(e_C;e_R)$ with both e_C and e_R non-empty, by the rule CHALLENGE-RESPONSE. Intuitively, ciphertexts of the form $\{\mathsf{Id}(I),\mathsf{Auth}(M),R_H(N),R'_{H'}(N'),\_\}_K$ with two nonces, the first used in an encrypted challenge (i.e., $H=\mathsf{CP}$ or $H=\mathsf{CC?}$) and the second used in an encrypted response (i.e., $H=\mathsf{PC}$ or $H=\mathsf{CC!}$), are projected into two submessages $\{\mathsf{Id}(I),\mathsf{Auth}(M),R_H(N),\_\}_K$ and $\{\mathsf{Id}(I),\mathsf{Auth}(M),R'_{H'}(N'),\_\}_K$ that are separately typechecked. The resulting effects, e_C from the challenge component and e_R from the response one, are finally collected into the type $\mathsf{Enc}(e_C;e_R)$.

We illustrate this typing rule with an example. Let us consider the following mutual-authentication protocol, where A sends B a message m encrypted with a long-term key shared between A and B.

$$\begin{array}{ccc} A & & B \\ & \leftarrow n_B \text{ —} & \\ & \text{— } \{A,n_B,n_A,m\}_{k_{AB}} \rightarrow & \\ & \leftarrow n_A \text{ —} & \end{array}$$

At the end of the protocol B authenticates m and A knows that B has received m. The second message is both the response of a PC handshake, based on n_B, and the challenge of a CP handshake, based on n_A. The protocol with types and tags is depicted below.

PC/CP HANDSHAKES

$$\begin{array}{ccc} A & & B \\ & \leftarrow n_B \text{ —} & \\ & \text{— } \{\mathsf{Id}(A),\mathsf{Claim}_{\mathsf{PC}}(n_B),\mathsf{Verif}_{\mathsf{CP}}(n_A),\mathsf{Auth}(m)\}_{k_{AB}} \rightarrow & \\ & \leftarrow n_A \text{ —} & \end{array}$$

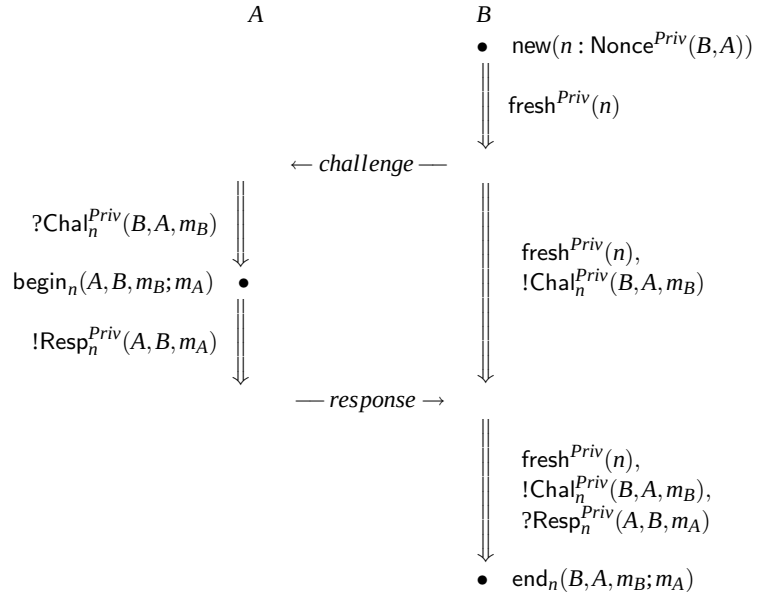
$$\begin{array}{ll} \Gamma\vdash & A:\mathsf{Un},B:\mathsf{Un},m:\mathsf{Un} \\ & k_{AB}:\mathsf{SharedKey}(A,B) \\ & n_B:\mathsf{Un} \\ & n_A:\mathsf{Nonce}^{Ciph}(A,B) \\ & \{\mathsf{Id}(A),\mathsf{Claim}_{\mathsf{PC}}(n_B),\mathsf{Auth}(m)\}_{k_{AB}}:\mathsf{Enc}([];[\mathsf{Resp}^{Ciph}_{n_B}(A,B,m)]) \\ & \{\mathsf{Id}(A),\mathsf{Verif}_{\mathsf{CP}}(n_A),\mathsf{Auth}(m)\}_{k_{AB}}:\mathsf{Enc}([\mathsf{Chal}^{Ciph}_{n_A}(A,B,m)];[]) \\ & \{\mathsf{Id}(A),\mathsf{Claim}_{\mathsf{PC}}(n_B),\mathsf{Verif}_{\mathsf{CP}}(n_A),\mathsf{Auth}(m)\}_{k_{AB}}: \\ & \qquad \mathsf{Enc}([\mathsf{Chal}^{Ciph}_{n_A}(A,B,m)];[\mathsf{Resp}^{Ciph}_{n_B}(A,B,m)]) \end{array}$$

Typing $\{\mathsf{Id}(A),\mathsf{Claim}_{\mathsf{PC}}(n_B),\mathsf{Verif}_{\mathsf{CP}}(n_A),\ \mathsf{Auth}(m)\}_{k_{AB}}$ requires to project the ciphertext into a response and challenge component as follows:

$$\{\mathsf{Id}(A),\mathsf{Claim}_{\mathsf{PC}}(n_B),\mathsf{Auth}(m)\}_{k_{AB}} \text{ and } \{\mathsf{Id}(A),\mathsf{Verif}_{\mathsf{CP}}(n_A),\mathsf{Auth}(m)\}_{k_{AB}}$$

The two ciphertexts differ only in the nonces and have type $\mathsf{Enc}([];[\mathsf{Resp}^{Ciph}_{n_B}(A,B,m)])$ and $\mathsf{Enc}([\mathsf{Chal}^{Ciph}_{n_A}(A,B,m)];[])$, respectively. The type of the whole protocol ciphertext

Table 6 Typing nonce handshakes: an example



$\Gamma \vdash$ $k_A : \mathsf{Key}(A)$
$k_B : \mathsf{Key}(B)$
$n : \mathsf{Nonce}^{Priv}(B,A)$
$\{|\mathsf{Id}(B), \mathsf{Auth}(m_B), \mathsf{Verif}_{\mathsf{CC?}}(n)|\}_{\mathsf{Pub}(k_A)} : \mathsf{Enc}([\mathsf{Chal}_n^{Priv}(B,A,m_B)];[])$
$\{|\mathsf{Id}(A), \mathsf{Auth}(m_A), \mathsf{Claim}_{\mathsf{CC!}}(n)|\}_{\mathsf{Pub}(k_B)} : \mathsf{Enc}([];[\mathsf{Resp}_n^{Priv}(A,B,m_A)])$

$\mathsf{Enc}([\mathsf{Chal}_{n_A}^{Ciph}(A,B,m)];[\mathsf{Resp}_{n_B}^{Ciph}(A,B,m)])$ collects the challenge and response effects obtained by separately typing the challenge and response components.

4.3 Typing Nonce Handshakes

In this section, we illustrate through a simple example how the analysis combines types and effects for type-checking nonce handshakes. Let us consider the CC handshake of Example 4, allowing the initiator B to authenticate the responder A receiving message m_B and sending message m_A. The narration of the protocol, decorated with correspondence assertions, types and effects, is depicted in Table 6.

The initiator B generates a fresh nonce n with type $\mathsf{Nonce}^{Priv}(B,A)$: the atomic effect $\mathsf{fresh}^{Priv}(n)$ tracks the freshness of the nonce. The nonce is encrypted together with message m_B in a challenge sent by B to A having type $\mathsf{Enc}([\mathsf{Chal}_n^{Priv}(B,A,m_B)];[])$, as discussed in Section 4.2. The atomic effect $!\mathsf{Chal}_n^{Priv}(B,A,m_B)$, extracted from the above type, tracks the generation of the challenge. The ciphertext is received and de-

crypted by the responder A, who tracks this event through the effect $?\mathsf{Chal}_n^{Priv}(B,A,m_B)$ extracted, again, from the ciphertext type. This shows how ciphertext types are used to remotely agree on the challenges (and responses) sent and received.

The following $\mathsf{begin}_n(A,B,m_B;m_A)$ assertion requires a challenge containing the message m_B (effect $?\mathsf{Chal}_n^{Priv}(B,A,m_B)$) and justifies a response for authenticating m_A (effect $!\mathsf{Resp}_n^{Priv}(A,B,m_A)$). This is the reason why effects having the form $!\mathsf{Resp}_n^{\ell}(\cdot)$ are used for *enabling* rather than tracking the generation of responses, as mentioned in Section 4.1.

The atomic effect $!\mathsf{Resp}_n^{Priv}(A,B,m_A)$ enables A to generate a ciphertext with type $\mathsf{Enc}([];[\mathsf{Resp}_n^{Priv}(A,B,m_A)])$. Hence this ciphertext is received and decrypted by B, tracked by $?\mathsf{Resp}_n^{Priv}(A,B,m_A)$ thanks to the ciphertext type. Since the nonce handshake has been successfully completed (effects $!\mathsf{Chal}_n^{Priv}(B,A,m_B)$ and $?\mathsf{Resp}_n^{Priv}(A,B,m_A)$) and the nonce is fresh (effect $\mathsf{fresh}^{Priv}(n)$), B can assert $\mathsf{end}_n(B,A,m_B;m_A)$.

4.3.1 Kinds of Handshakes

In the example above, both the challenge and the response are private. In general, challenge and response effects justifying an end assertion have the form $!\mathsf{Chal}_n^{\ell}(B,A,M_C)$ and $?\mathsf{Resp}_n^{\overline{\ell}}(A,B,M_R)$, respectively. The relation between ℓ and $\overline{\ell}$ is determined by the kind of handshake:

PC the nonce is sent out in clear and received encrypted, thus $\ell = Un$ and $\overline{\ell} = Ciph$;

CP the nonce is sent out encrypted and received in clear, thus $\ell = Ciph$ and $\overline{\ell} = Un$;

CC the nonce may be sent out and received either encrypted ($\ell = \overline{\ell} = Priv$) or signed ($\ell = \overline{\ell} = Int$).

If the nonce n is sent as challenge in clear, then the effect tracking such an output is $!\mathsf{Chal}_n^{Un}()$: notice that no message is associated with the challenge. Analogously, the input of a challenge in clear is tracked by $?\mathsf{Chal}_n^{Un}()$. Similar reasoning applies to responses that are sent and received in clear.

4.4 Typing Processes

Processes are typed according to the judgment $\Gamma \vdash P : e$, meaning that the process P can be typed under the typing environment Γ and the effect e. Judgment $I;\Gamma \vdash S : e$, for the sequential process S executed by the entity I, has the same intuitive meaning. For easing the presentation, we introduce some conventions: $\Gamma \vdash M_1,\ldots,M_n : T$ is an abbreviation of $\Gamma \vdash M_1 : T,\ldots,\Gamma \vdash M_n : T$ and we write $![t_1,\ldots,t_n]$ and $?[t_1,\ldots,t_n]$ to denote $[!t_1,\ldots,!t_n]$ and $[?t_1,\ldots,?t_n]$, respectively. Finally, $+$ 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.

Process judgments are reported in Table 7 and described below. SYMMETRIC KEY and ASYMMETRIC KEY type key declarations, binding the key with the appropriate

Table 7 Typing processes

$$\begin{array}{rcl}
\textsc{Encrypt} \\ \textsc{Decrypt} & : & \begin{cases} M = \{N\}_K \Rightarrow \Gamma \vdash K : T \in \{\mathsf{Un}, \mathsf{SharedKey}(A,I)\} \\ M = \{|N|\}_K \Rightarrow \Gamma \vdash K : T \in \{\mathsf{PrivateKey}(A), \mathsf{PublicKey}(I)\} \end{cases} \\
\textsc{Decrypt}: & & \{M\}_K^{-1} = \{M\}_K,\ \{|M|\}_{\mathsf{key}(K)}^{-1} = \{|M|\}_{\overline{\mathsf{key}}(K)} \\
\textsc{Begin and End}: & & (\ell, \overline{\ell}) \in \{(Un, Ciph), (Ciph, Un), (Priv, Priv), (Int, Int)\} \text{ (cf Section 4.3)}
\end{array}$$

$$\textsc{Symmetric Key}\ \frac{\Gamma, k : \mathsf{SharedKey}(I,J) \vdash P : []}{\Gamma \vdash \mathsf{let}\ k = \mathsf{sym\text{-}key}(I,J).P : []} \qquad \textsc{Asymmetric-Key}\ \frac{\Gamma, k : \mathsf{Key}(I) \vdash P : []}{\Gamma \vdash \mathsf{let}\ k = \mathsf{asym\text{-}key}(I).P : []}$$

$$\textsc{Replication}\ \frac{\Gamma \vdash A \triangleright S : []}{\Gamma \vdash A \triangleright !S : []} \qquad \textsc{Par}\ \frac{\Gamma \vdash P : e_P \qquad \Gamma \vdash Q : e_Q}{\Gamma \vdash P|Q : e_P + e_Q} \qquad \textsc{Identity}\ \frac{A; \Gamma \vdash S : e}{\Gamma \vdash A \triangleright S : e} \qquad \textsc{Nil}\ A; \Gamma \vdash \mathbf{0} : []$$

$$\textsc{New}\ \frac{\ell \in \{Un, Priv, Ciph, Int\} \qquad A; \Gamma, n : \mathsf{Nonce}^{\ell}(A,I) \vdash S : e}{A; \Gamma \vdash \mathsf{new}(n : \mathsf{Nonce}^{\ell}(A,I)).S : e - [\mathsf{fresh}^{\ell}(n)]}$$

$$\textsc{Input}\ \frac{A; \Gamma, vars(M) : \mathsf{Un} \vdash S : e}{A; \Gamma \vdash \mathsf{in}(M).S : e - [?\mathsf{Chal}_M^{Un}(), ?\mathsf{Resp}_M^{Un}()]} \qquad \textsc{Output}\ \frac{\Gamma \vdash M : \mathsf{Un} \qquad A; \Gamma \vdash S : e}{A; \Gamma \vdash \mathsf{out}(M).S : e - [!\mathsf{Chal}_M^{Un}()]}$$

$$\textsc{DownCast to Untrusted}\ \frac{A; \Gamma, x : \mathsf{Un} \vdash S : e \qquad \Gamma \vdash N : \mathsf{Nonce}^{Ciph}(I,A)}{A; \Gamma \vdash \mathsf{cast}\ N\ \mathsf{is}\ (x : \mathsf{Un}).S : e + [!\mathsf{Resp}_N^{Un}()]}$$

$$\textsc{Encrypt}\ \frac{\Gamma \vdash M : \mathsf{Enc}(e_C; e_R) \qquad A; \Gamma, z : \mathsf{Un} \vdash S : e + !e_C}{A; \Gamma \vdash \mathsf{encrypt}\ M\ \mathsf{as}\ z.S : e + !e_R}$$

$$\textsc{Decrypt}\ \frac{vars(M) = \overline{x} \qquad \Gamma, \overline{x} : \overline{T} \vdash M^{-1} : \mathsf{Enc}(e_C; e_R) \qquad \begin{array}{c} \Gamma \vdash M' : \mathsf{Un} \\ A; \Gamma, \overline{x} : \overline{T} \vdash S : e + ?e_C + ?e_R \end{array}}{A; \Gamma \vdash \mathsf{decrypt}\ M'\ \mathsf{as}\ M.S : e}$$

$$\textsc{Begin}\ \frac{A; \Gamma \vdash S : e + [!\mathsf{Resp}_N^{\overline{\ell}}(A,I,M_2)] \qquad \Gamma \vdash N : \mathsf{Nonce}^{\ell}(I,A) \qquad \Gamma \vdash M_2 : T}{A; \Gamma \vdash \mathsf{begin}_N(A,I,M_1;M_2).S : e + [?\mathsf{Chal}_N^{\ell}(I,A,M_1)]}$$

$$\textsc{End}\ \frac{A; \Gamma \vdash S : e \qquad \Gamma \vdash n : \mathsf{Nonce}^{\ell}(A,I)}{A; \Gamma \vdash \mathsf{end}_N(A,I,M_1;M_2).S : e + [!\mathsf{Chal}_n^{\ell}(A,I,M_1), ?\mathsf{Resp}_n^{\overline{\ell}}(I,A,M_2), \mathsf{fresh}^{\ell}(n)]}$$

type into the typing context. As we will see later on, if a process type-checks under empty effect, then that process is safe. Since we type processes for verifying their safety, and key declarations are at the beginning of each process, typing a key declaration is useful only if the effect is empty. For this reason, SYMMETRIC KEY and ASYMMETRIC KEY require the process effect to be empty. Similarly, REPLICATION types the replication of a principal under empty effect, if that principal is in turn typed under empty effect.

PAR types the parallel composition of two processes under the union of their effects. Intuitively, a parallel composition of two processes is safe if both of the processes are safe: since an effect represents the assumptions under which a process is safe, a parallel composition of two processes is safe if both of the assumptions required to type them hold.

IDENTITY makes the typing of a sequential process dependent on the identity label I of the principal running it through the judgement $I;\Gamma \vdash S : e$.

NIL types process $\mathbf{0}$ under empty effect since typing the null process does not require any specific assumption on the principal's behavior. Notice that the typing of a process is defined by induction on its structure and the null process is the base case: since each typing rule univocally determines an effect from the one of the continuation process, the effect of type-checked processes is unique.

Restriction is typed by NEW: the name n is inserted into the typing environment with the declared type. The restriction justifies, through the atomic effect $\mathsf{fresh}^{\ell}(n)$, at most one use of n as fresh nonce with integrity/secrecy property ℓ in the continuation process. Subtracting an atomic effect $\mathsf{fresh}^{\ell}(n)$ in the thesis allows the continuation process to type-check with or without $\mathsf{fresh}^{\ell}(n)$, thus admitting processes that make no use of the freshly generated nonce.

INPUT gives type Un to the read messages as they come from the untrusted network. Typing $\mathsf{in}(M)$ justifies in the continuation process that M has been received in clear either as challenge (effect $?\mathsf{Chal}_M^{Un}()$) or response (effect $?\mathsf{Resp}_M^{Un}()$).

OUTPUT says that a term can be sent on the network only if it has type Un. Typing $\mathsf{out}(M)$ justifies in the continuation process that M has been sent in clear as challenge (effect $!\mathsf{Chal}_M^{Un}()$).

As mentioned in Section 4.1, nonce types regulate secrecy and integrity properties of nonces. A type cast is needed only in CP handshakes as the nonce is sent back in clear and its secrecy is lost. DOWNCAST TO UNTRUSTED casts the type of N from $\mathsf{Nonce}^{Ciph}(I,A)$ to Un. This cast requires the atomic effect $!\mathsf{Resp}_N^{Un}()$, namely the permission to send the nonce in clear on the network. Notice the use of the effect union $+$ in the thesis for extending the assumptions required for type-checking the process; these assumptions have to be justified by previous actions.

ENCRYPT types the generation of a ciphertext M, relying on the judgement $\Gamma \vdash M : \mathsf{Enc}(e_C;e_R)$: the generation of a challenge is tracked in the continuation process (effect $!e_C$), while the generation of a response has to be justified in the process' effect (effect $!e_R$), as discussed in Section 4.3. Adding the effect $!e_C$ in the hypothesis forces the continuation process to use the generated challenge. The idea is to only type-check protocols that really need all the specified tagged messages for completing the authentication task; in other words, protocol specifications are required to be minimally tagged.

DECRYPT types the received messages according to the type of the ciphertext. In practice, finding these types is trivial, since it amounts to giving type Un to all the variables except the one with tag R_H which is given type $\mathsf{Nonce}^{\ell}(I,J)$ returned by *Typeofenc* (cf. Section 4.2). For instance, typing decrypt z as $\{\mathsf{Id}(A),\mathsf{Verif}_{\mathsf{CP}}(x),\mathsf{Auth}(y)\}_k.P$, with k of type $\mathsf{SharedKey}(A,B)$, gives x the type $\mathsf{Nonce}^{Ciph}(A,B)$ and y the type Un as $\mathit{Typeofenc}(A,\mathsf{Verif}_{\mathsf{CP}},x,y,\mathsf{SharedKey}(A,B)) = ([\mathsf{Chal}_x^{Ciph}(A,B,y)],[],\mathsf{Nonce}^{Ciph}(A,B))$. Notice that in the case of ciphertexts decrypted through asymmetric cryptography, in order to type the ciphertext that one expects to decrypt at run-time, we need to substitute the decryption key with the corresponding encryption one. For example, typing decrypt z as $\{|\mathsf{Id}(A),\mathsf{Verif}_{\mathsf{CP}}(N),\mathsf{Auth}(M)|\}_{\mathsf{Priv}(k)}$ requires to type the ciphertext $\{|\mathsf{Id}(A),\mathsf{Verif}_{\mathsf{CP}}(N),\mathsf{Auth}(M)|\}_{\mathsf{Pub}(k)}$. This is formalized through the function $\cdot^{-1}$, inverting the top-level asymmetric key. Once the ciphertext is typed, the reception of challenges and responses is tracked in the continuation process (effects $?e_C$ and $?e_R$).

As illustrated in Section 4.3, a $\mathsf{begin}_N(A,I,M_1;M_2)$ by A (rule BEGIN) requires the reception of a challenge from I, containing the nonce N and the message M_1 (effect $?\mathsf{Chal}_N^{\ell}(I,A,M_1)$), and justifies the generation of a response to I, containing the same nonce and the message M_2 (effect $!\mathsf{Resp}_N^{\overline{\ell}}(A,I,M_2)$). Finally, an $\mathsf{end}_N(A,I,M_1;M_2)$ by A (rule END) requires the freshness of the nonce n (effect $\mathsf{fresh}^{\ell}(n)$), the generation of a challenge to I, containing the same nonce and the message M_1 (effect $!\mathsf{Chal}_n^{\ell}(A,I,M_1)$), and the reception of a response from I, containing the same nonce and the message M_2 (effect $?\mathsf{Resp}_n^{\overline{\ell}}(I,A,M_2)$).

4.5 Safety Theorem

Our main result states that if a process can be typed with empty effect and empty typing environment, then every trace generated by that process is safe.

Theorem 1 (Safety) *Let P be a process. If $\overline{I} : \mathsf{Un} \vdash P : []$, where $\overline{I}$ are the identities in P, then P is safe.* ■

Interestingly, our analysis is strongly compositional, as stated by the following theorem. Let $\mathbf{keys}(k_1,\ldots,k_n)$ denote a sequence of key declarations.

Theorem 2 (Strong Compositionality) *Let P be the process $\mathbf{keys}(k_1,\ldots,k_n).(I_1\triangleright!S_1|\ldots|I_m\triangleright!S_m)$ and $I_1,\ldots,I_m$ be the identities in P. Then $I_1,\ldots,I_m : \mathsf{Un} \vdash P : []$ if and only if $I_1,\ldots,I_m : \mathsf{Un} \vdash \mathbf{keys}(k_1,\ldots,k_n).I_i\triangleright!S_i : [], \forall i \in [1,m]$.* ■

Intuitively, a protocol is safe if so are all the protocol participants. In addition, judging a participant safe only requires knowledge of the long-term keys it shares with other participants. This is a fairly mild assumption as the information conveyed by the keys is relative to identities of the parties sharing them, not to the protocol they are running. This flexibility is paid in ρ-spi in terms of the dynamic checks required on the structure of messages to validate the tags used in the different handshakes. Indeed, we do not regard the dynamic checks as limiting or inconvenient: instead, our contention is

that the ρ-spi tagging of ciphertexts is a good specification practice which helps document the intended semantics of the ciphertexts and yields strong compositionality to the handshakes. We remark that tagged messages generated by trusted principals are protected by cryptography while being in transit on the network, so that their integrity is always guaranteed with the exception of public-key encrypted messages that might be generated even by the enemy; this is handled by means of the subtyping relation discussed in Section 4.1.

5 Example

We illustrate our system with a slightly modified version of the SPLICE/AS Protocol (see the remark at the end of this section):

	A	B
(Msg 1)	$\leftarrow B, n_B$ —	
(Msg 2)	— $A, B, \{B, n_B, \{n_A\}_{\mathsf{Pub}(k_B)}\}_{\mathsf{Priv}(k_A)} \rightarrow$	
(Msg 3)	$\leftarrow \{B, n_A\}_{\mathsf{Pub}(k_A)}$ —	

This protocol should mutually authenticate A and B. In the first two messages, B exploits a PC nonce handshake to authenticate A, as n_B is sent in clear and received encrypted. In the second message, A sends a (nested) challenge to B using a CC scheme. However, this second part is flawed as shown by the following (man-in-the-middle) attack sequence:

	A	$(B)E$	B
1.b		|	$\leftarrow B, n_B$ —
1.a	$\leftarrow B, n_B$ —	|	
2.a	— $A, B, \{B, n_B, \{n_A\}_{\mathsf{Pub}(k_B)}\}_{\mathsf{Priv}(k_A)} \rightarrow$	|	
2.b		|	— $E, B, \{B, n_B, \{n_A\}_{\mathsf{Pub}(k_B)}\}_{\mathsf{Priv}(k_E)} \rightarrow$
3.b		|	$\leftarrow \{B, n_A\}_{\mathsf{Pub}(k_E)}$ —
3.a	$\leftarrow \{B, n_A\}_{\mathsf{Pub}(k_A)}$ —	|	

There are two parallel sessions: a and b. Session b is between E and B, and E exploits such a session to impersonate B with A in session a. In particular, the enemy uses B as an oracle to decrypt message $\{n_A\}_{\mathsf{Pub}(k_B)}$ (messages 2.b and 3.b). The projection on input, output and correspondence assertions of the trace corresponding to the attack is

$$
\begin{array}{ll}
1.b & \mathit{out}(B, n_B) :: \\
1.a & \mathit{in}(B, n_B) :: \\
2.a & \mathit{begin}_{n_B}(\mathbf{A}, B) :: \mathit{out}(A, B, \{|B, n_B, \{|n_A|\}_{\mathsf{Pub}(k_B)}|\}_{\mathsf{Priv}(k_A)}) :: \\
2.b & \mathit{in}(E, B, \{|B, n_B, \{|n_A|\}_{\mathsf{Pub}(k_B)}|\}_{\mathsf{Priv}(k_E)}) :: \mathit{end}_{n_B}(B, \mathbf{E}) :: \\
3.b & \mathit{begin}_{n_A}(B, \mathbf{E}) :: \mathit{out}(\{|B, n_A|\}_{\mathsf{Pub}(k_E)}) :: \\
3.a & \mathit{in}(\{|B, n_A|\}_{\mathsf{Pub}(k_A)}) :: \mathit{end}_{n_A}(\mathbf{A}, B)
\end{array}
$$

Looking at correspondence assertions we have an $end_{n_A}(A,B)$ (session a) which does not match $begin_{n_A}(B,E)$ (session b). Notice that this attack is similar to the well known one on the Needham-Schroeder public-key protocol [28]. In ρ-spi calculus, we capture these attack sequences thanks to the possibility of providing the enemy with long-term keys. Indeed, the environment decrypts the ciphertext sent by A in $2.a$ as it knows A's public key and in $2.b$ generates a new ciphertext by its own private key. Similarly, the environment decrypts by its own private key the ciphertext sent by B in $3.b$ and in $3.a$ generates a new ciphertext encrypted with A's public key.
Since the protocol is unsafe it cannot be type-checked. We give the intuition of why this happens. A may be specified as follows:

$$\begin{array}{ll}
(\text{Msg } 1) & \mathsf{in}(B,x)\ . \\
(\text{Msg } 2) & \mathsf{new}(n_A)\ .\ \mathsf{encrypt}\ \{|n_A|\}_{\mathsf{Pub}(k_B)}\ \mathsf{as}\ z\ . \\
 & \quad \mathsf{encrypt}\ \{|B,x,z|\}_{\mathsf{Priv}(k_A)}\ \mathsf{as}\ w\ .\ \mathsf{out}(A,B,w)\ . \\
(\text{Msg } 3) & \mathsf{in}(y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{|B,n_A|\}_{\mathsf{Priv}(k_A)}.\mathsf{end}_{n_A}(A,B)
\end{array}$$

There is no tagging that makes the sequential process above type-check. Indeed, the challenge effect $!\mathsf{Chal}^{Priv}_{n_A}(A,B)$, required by $\mathsf{end}_{n_A}(A,B)$, is not justified by any encryption. In order to justify that effect, the identity label A has to be inserted in the first ciphertext. As a matter of fact, by changing message 2 into

$$(\text{Msg } 2) \qquad A \ \rightarrow\ B: \quad A,B,\{B,n_B,\{\mathbf{A},n_A\}_{\mathsf{Pub}(k_B)}\}_{\mathsf{Priv}(k_A)}$$

we obtain the repaired protocol suggested by Gavin Lowe in [30]. To see how this protocol is checked using our type and effect system, consider the following tagged version:

$$\begin{array}{lcc}
 & A & B \\
(\text{Msg } 1) & \leftarrow B,n_B\ \text{—} & \\
(\text{Msg } 2) & \text{—}\ A,B,\{\mathsf{Id}(B),\mathsf{Verif}_{\mathsf{PC}}(n_B),\{\mathsf{Id}(A),\mathsf{Verif}_{\mathsf{CC?}}(n_A)\}_{\mathsf{Pub}(k_B)}\}_{\mathsf{Priv}(k_A)} \rightarrow & \\
(\text{Msg } 3) & \leftarrow \{\mathsf{Id}(B),\mathsf{Claim}_{\mathsf{CC!}}(n_A)\}_{\mathsf{Pub}(k_A)}\ \text{—} &
\end{array}$$

In (Msg 2), A communicates to B that B is the verifier of the current authentication session by a PC handshake on n_B and asks B to confirm whether or not he is willing to start an authentication session with her using a CC handshake on n_A. Thus, we tag n_B by $\mathsf{Verif}_{\mathsf{PC}}$ and n_A by $\mathsf{Verif}_{\mathsf{CC?}}$. In the second ciphertext, B completes the CC handshake and n_A is tagged by $\mathsf{Claim}_{\mathsf{CC!}}$.

The proof derivation for the ρ-spi calculus specification of the protocol is reported in Table 8 (protocol definition), Table 9 (initiator) and Table 10 (responder): for each primitive π followed by the process P, we indicate both the effect e (right column) and the environment Γ (left column) for typing the process $\pi.P$. (in Table 9 and Table 10 we omit the principal I executing the sequential process.) The environment is specified incrementally, by just indicating the type binding added by the current primitive π. Notation '…' is a shorthand for the environment used in the previous typing judgement (previous row in the table).

Notice that we analyze an unbounded number of sessions, where A and B play both the initiator and the responder role.

Table 8 Proof Derivation for the protocol definition of the SPLICE/AS protocol

$A:\mathsf{Un}$, $B:\mathsf{Un}$, $E:\mathsf{Un}$	$\vdash$	$Protocol_{Splice/AS} \triangleq$	:	$[]$
$\ldots$	$\vdash$	let $k_A = \mathsf{asym\text{-}key}(A)$.	:	$[]$
$\ldots, k_A:\mathsf{Key}(A)$	$\vdash$	let $k_B = \mathsf{asym\text{-}key}(B)$.	:	$[]$
$\ldots, k_B:\mathsf{Key}(B)$	$\vdash$	let $k_E = \mathsf{asym\text{-}key}(E)$.	:	$[]$
$\ldots, k_E:\mathsf{Key}(E)$	$\vdash$	$(\mid \begin{smallmatrix}\forall I\in\{A,B\}\\ \forall J\in\{A,B,E\}\end{smallmatrix}$ $I\triangleright !Initiator_{Splice/AS}(I,J,\mathsf{Priv}(k_I),\mathsf{Pub}(k_J)) \mid$ $I\triangleright !Responder_{Splice/AS}(I,J,\mathsf{Priv}(k_I),\mathsf{Pub}(k_J))$ $)$	:	$[]$

The proof for initiator and responder processes is shown for two arbitrary principals I and J. Notice, however, that typing rules are defined on arbitrary identities with the only constraint that sequential processes are executed by trusted principals. Thus, typability of sequential processes is closed under identity substitutions as far as the executing principal is trusted. This is why we can analyze an unbounded number of sessions where A and B play both the initiator and the responder role, possibly communicating with E. This is achieved, in Table 8, by simply instantiating I to A or B and J to A, B or E. Since $A,B,E:\mathsf{Un} \vdash Protocol_{Splice/AS}:[]$, by Theorem 1 the protocol is safe.
We remark that the specification can be easily generalized to an arbitrary number m of entities, acting both as initiator and as responder and arbitrarily replicated.

$$\begin{aligned} Protocol_{Splice/AS} \triangleq\ & \mathsf{let}_{i=1}^{m} k_{I_i} = \mathsf{asym\text{-}key}(I_i). \\ & (\mid_{\substack{i,j=1 \\ I_i \notin I\mathcal{D}_{\mathcal{I}}}}^{m} I_i \triangleright !Initiator_{Splice/AS}(I_i,I_j,\mathsf{Priv}(k_{I_i}),\mathsf{Pub}(k_{I_j}))\mid \\ & \qquad I_i \triangleright !Responder_{Splice/AS}(I_i,I_j,\mathsf{Priv}(k_{I_i}),\mathsf{Pub}(k_{I_j}))) \end{aligned}$$

Here k_{I_i} represents the asymmetric key owned by entity I_i. Typability is proved, as before, by observing that the running principal I_i is only instantiated to trusted identities.

Remark 1 In the original version of the SPLICE/AS protocol, the format of (Msg 2) is

$$A, B, \{A, n_B, \{n_A\}_{\mathsf{Pub}(k_B)}\}_{\mathsf{Priv}(k_A)}$$

with A, rather than B, as the entity identifier of the top-level encrypted packet. With our present system, there is no tagging for this message enabling a successful validation of the protocol. We believe the problem can be circumvented, and the protocol verified, by enhancing the system to allow (messages encrypted by) public keys to be treated as entity identifiers, and to be tagged as such.

■

Table 9 Proof derivation for $Initiator_{Splice/AS}$

$I : \mathsf{Un}$, $J : \mathsf{Un}$, $k_I : \mathsf{Key}(I)$ $k_J : \mathsf{Key}(J)$	$\vdash$	$Initiator_{Splice/AS}(I, J, \mathsf{Priv}(k_I), \mathsf{Pub}(k_J)) \triangleq$	:	$[]$
$\ldots$	$\vdash$	$\mathsf{new}(n_I : \mathsf{Nonce}^{Un}(I,J))$.	:	$[]$
$\ldots, n_I : \mathsf{Nonce}^{Un}(I,J)$	$\vdash$	$\mathsf{out}(I, n_I)$.	:	$[\mathsf{fresh}^{Un}(n_I)]$
$\ldots$	$\vdash$	$\mathsf{in}(J, I, y)$.	:	$\left[\begin{array}{c}\mathsf{fresh}^{Un}(n_I),\\ !\mathsf{Chal}^{Un}_{n_I}()\end{array}\right]$
$\ldots, y : \mathsf{Un}$	$\vdash$	$\mathsf{decrypt}\ y\ \mathsf{as}\ \{\lvert \mathsf{Id}(I), \mathsf{Verif}_{\mathsf{PC}}(n_I), z' \rvert\}_{\mathsf{Pub}(k_J)}$	:	$\left[\begin{array}{c}\mathsf{fresh}^{Un}(n_I),\\ !\mathsf{Chal}^{Un}_{n_I}()\end{array}\right]$
$\ldots, z' : \mathsf{Un}$	$\vdash$	$\mathsf{end}_{n_I}(I, J)$.	:	$\left[\begin{array}{c}\mathsf{fresh}^{Un}(n_I),\\ !\mathsf{Chal}^{Un}_{n_I}(),\\ ?\mathsf{Resp}^{Ciph}_{n_I}(J, I)\end{array}\right]$
$\ldots$	$\vdash$	$\mathsf{decrypt}\ z'\ \mathsf{as}\ \{\lvert \mathsf{Id}(J), \mathsf{Verif}_{\mathsf{CC?}}(x) \rvert\}_{\mathsf{Priv}(k_I)}$.	:	$[]$
$\ldots, x : \mathsf{Nonce}^{Priv}(J,I)$	$\vdash$	$\mathsf{begin}_x(I, J)$.	:	$[?\mathsf{Chal}^{Priv}_x(J,I)]$
$\ldots$	$\vdash$	$\mathsf{encrypt}\ \{\lvert \mathsf{Id}(I), \mathsf{Claim}_{\mathsf{CC!}}(x) \rvert\}_{\mathsf{Pub}(k_J)}\ \mathsf{as}\ z''$.	:	$[!\mathsf{Resp}^{Priv}_x(I,J)]$
$\ldots, z'' : \mathsf{Un}$	$\vdash$	$\mathsf{out}(z'')$.	:	$[]$
$\ldots$	$\vdash$	$\mathbf{0}$	:	$[]$

Table 10 Proof derivation for $Responder_{Splice/AS}$

$I : \mathsf{Un}$, $J : \mathsf{Un}$ $k_I : \mathsf{Key}(I)$ $k_J : \mathsf{Key}(J)$	$\vdash$	$Responder_{Splice/AS}(I,J,\mathsf{Priv}(k_I),\mathsf{Pub}(k_J)) \triangleq$	:	$[]$
$\ldots$	$\vdash$	$\mathsf{new}(n_I : \mathsf{Nonce}^{Priv}(I,J))$.	:	$[]$
$\ldots$, $n_I : \mathsf{Nonce}^{Priv}(I,J)$	$\vdash$	$\mathsf{in}(J,x)$.	:	$[\mathsf{fresh}^{Priv}(n_I)]$
$\ldots, x : \mathsf{Un}$	$\vdash$	$\mathsf{begin}_x(I,J)$.	:	$\left[\begin{array}{c}\mathsf{fresh}^{Priv}(n_I),\\ ?\mathsf{Chal}_x^{Un}()\end{array}\right]$
$\ldots$	$\vdash$	encrypt $\{\lvert \mathsf{Id}(I), \mathsf{Verif}_{\mathsf{CC?}}(n_I)\rvert\}_{\mathsf{Pub}(k_J)}$ as z.	:	$\left[\begin{array}{c}\mathsf{fresh}^{Priv}(n_I),\\ !\mathsf{Resp}_x^{Ciph}(I,J)\end{array}\right]$
$\ldots, z : \mathsf{Un}$	$\vdash$	encrypt $\{\lvert \mathsf{Id}(J), \mathsf{Verif}_{\mathsf{PC}}(x), z\rvert\}_{\mathsf{Priv}(k_I)}$ as z'	:	$\left[\begin{array}{c}\mathsf{fresh}^{Priv}(n_I),\\ !\mathsf{Chal}_{n_I}^{Priv}(I,J),\\ !\mathsf{Resp}_x^{Ciph}(I,J)\end{array}\right]$
$\ldots, z' : \mathsf{Un}$	$\vdash$	$\mathsf{out}(I,J,z')$.	:	$\left[\begin{array}{c}\mathsf{fresh}^{Priv}(n_I),\\ !\mathsf{Chal}_{n_I}^{Priv}(I,J)\end{array}\right]$
$\ldots$	$\vdash$	$\mathsf{in}(y)$.	:	$\left[\begin{array}{c}\mathsf{fresh}^{Priv}(n_I),\\ !\mathsf{Chal}_{n_I}^{Priv}(I,J)\end{array}\right]$
$\ldots, y : \mathsf{Un}$	$\vdash$	decrypt y as $\{\lvert \mathsf{Id}(J), \mathsf{Claim}_{\mathsf{CC!}}(n_I)\rvert\}_{\mathsf{Priv}(k_I)}$.	:	$\left[\begin{array}{c}\mathsf{fresh}^{Priv}(n_I),\\ !\mathsf{Chal}_{n_I}^{Priv}(I,J)\end{array}\right]$
$\ldots$	$\vdash$	$\mathsf{end}_{n_I}(I,J)$	:	$\left[\begin{array}{c}\mathsf{fresh}^{Priv}(n_I),\\ !\mathsf{Chal}_{n_I}^{Priv}(I,J),\\ ?\mathsf{Resp}_{n_I}^{Priv}(J,I)\end{array}\right]$
$\ldots$	$\vdash$	$\mathbf{0}$	:	$[]$

6 Multi-Protocol Systems

Theorem 2 may be directly applied to the verification of multi-protocol systems. The interaction among *different* protocols, possibly carrying out some common sub-tasks, is considered an open issue [32], and it is particularly interesting in a global computing setting, where several security services coexist and are possibly combined together. However, as discussed by Syverson and Meadows in [31], "problems can arise when a protocol is interacting with another protocol that does not use a tagging scheme, or tags data in a different way". As an example, let us consider the two simple protocols below:

	A	B
(msg 1.a)	$\leftarrow n$ —	
(msg 2.a)	— $\{A, m, n\}_{k_{AB}} \rightarrow$	

	A	B
(msg 1.b)	$\leftarrow n$ —	
(msg 2.b)	— $\{B, m, n\}_{k_{AB}} \rightarrow$	

The two protocols exploit PC hand-shakes based on symmetric cryptography. They differ from each other because of the identity label inside the ciphertext: in the protocol on the left-side (protocol a) it is A, i.e. the claimant identity, while in the protocol on the right-side (protocol b) it is B, i.e. the verifier's one. At the end of the protocol B authenticates the message m from A. The two protocols are safe when all participants run the same protocol. Unfortunately, safety is broken when participants run both the protocols. Specifically, the following attack can be mounted by the enemy when B is running protocol a as initiator and protocol b as responder:

	E	B
(msg 1.b)	$\leftarrow n$ —	
(msg 1.a)	— $n \rightarrow$	
(msg 2.a)	$\leftarrow \{B, m, n\}_{k_{AB}}$ —	
(msg 2.b)	— $\{B, m, n\}_{k_{AB}} \rightarrow$	

Interestingly, the attack above cannot be mounted on the tagged version of the protocols, reported below.

	A	B
(msg 1.a)	$\leftarrow n$ —	
(msg 2.a)	— $\{\mathsf{Id}(A), \mathsf{Auth}(m), \mathsf{Claim}_{\mathsf{PC}}(n)\}_{k_{AB}} \rightarrow$	

	A	B
(msg 1.b)	$\leftarrow n$ —	
(msg 2.b)	— $\{\mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif}_{\mathsf{PC}}(n)\}_{k_{AB}} \rightarrow$	

Notice that the two ciphertexts differ because of the nonce tag that disambiguates whether the identity label should be intended as claimant or verifier of the authentication session. In other analyses (e.g. [21]) bad interactions among different protocols are prevented assuming ciphertexts belonging to a protocol to be tagged differently from ciphertexts belonging to any other protocol. The drawback of this approach is to prevent *all* the interactions among different protocols. Our tagging mechanism is less

demanding as ciphertexts conveying the same authentication guarantees, even if originated by different protocols, are tagged in the same way. Theorem 2 proves that any interaction among different protocols does not break safety. This result makes our system suitable for the analysis of complex protocols, possibly composed of sub-protocols interacting one with each other: each of the sub-components can be analyzed in isolation and, by Theorem 2, the safety of all the sub-components implies the safety of the protocol as a whole.

7 Conclusion and Related Work

We have proposed a type and effect system for authentication protocols. We have tested our analysis on several protocols and some results are reported in Table 11: in all cases, the analysis provide safety proofs for the correct versions of the protocols, while it consistently fails to validate the flawed versions. The main advantages of our proposal are

- scalability: since the authentication guarantees are local, safe sequential processes (possibly modeling different protocols) may be safely composed together;
- limited human effort: tagging is simple as it just requires to disambiguate the meaning of identifiers and nonces in some encrypted messages;
- simplicity: the type and effect system is simple yet expressive enough to verify many existing protocols.

The set of rules presented here and in [14] is general enough for analyzing many of the authentication protocols presented in literature. Specifically, we can analyze protocols combining the three kinds of nonce handshakes discussed in Section 2. Notice that the tagged ciphertexts that are typable, i.e., the ones in the domain of *Typeofenc*, strictly determine the class of protocols that the type and effect system can analyze. These ciphertexts cover many kind of commonly used handshakes, but of course, they are not the only way to safely achieve authentication. For instance, the fixed version of the SPLICE/AS authentication protocol, proposed by Gavin Lowe and discussed in Section 5, does not type-check because of the use of public keys in place of identities. Another class of protocols being out of the scope of the present system is the one of key-exchange protocols, using the exchanged session key to authenticate other messages. In [15] we discuss how the type and effect system might be extended to also cover this class of protocols. Indeed, the class of typable ciphertexts is, in principle, extendable to new cases of handshakes, but this requires to reprove the safety theorem. We are thus currently studying easy-to-verify conditions on such class, that imply the safety with no need of reproving the whole theorem.

Since the definition of tags and types may require some expertise on the part of programmers, we have developed a type-checker providing tag and type inference [19], thus further reducing the human effort required by the analysis and leaving to the user only the job to translate the protocol in ρ-spi calculus .

Related Work Tagging is not a new idea and it is proposed and used for verification purposes in [6, 7, 20, 21, 27]. Typically, tagging amounts to add a different label to

Table 11 Some Case Studies

Protocols	**Correct**	**Flawed**
CCITT X.509 (3)		X
Ban Modified Version of CCITT X.509 (3)	X	
SPLICE/AS		X
Lowe's fixed version of SPLICE/AS	X	
Needham-Schroeder Public Key Protocol		X
Lowe's fixed version of Needham-Schroeder Public Key Protocol	X	
ISO's Symmetric Key Three Pass Mutual Authentication Protocol	X	
Private Authentication Protocols [4]	X	

each encrypted protocol message, so that ciphertexts cannot be confused. Our tagging is less demanding, as we do not require that every message is unambiguously tagged since we tag only certain components. In particular, for protocols implemented with stronger tagging techniques, our tags can be safely removed without compromising the protocols' safety.

Our analysis can verify authentication in presence of a fixed, yet unbounded, number of enemies provided with long-term keys regarded as trusted by honest principals; in [3], Abadi and Blanchet prove a similar result for a type system for secrecy protocols. In a recent paper [22], Gordon and Jeffrey propose a type and effect system to check conditional secrecy, namely a refinement of secrecy where a message is unknown to the adversary unless particular messages or principals are compromised. Interestingly, the population of enemies is not fixed in advance and may dynamically grow. Authors claim that it should be possible to combine that type and effect system with the one for authentication [21] in order to check authentication even in presence of a dynamically growing set of enemies engaging as regular participants in protocol sessions.

The Strand Spaces formalism [23, 24, 25, 34] is an interesting framework for studying authentication. There are interesting similarities between our analysis and the way the three kinds of nonce-handshakes are checked in Strand Spaces. It would be interesting to explore how our type system could be applied in such a framework, in order to provide mechanical proofs of safety.

The recent work by Bodei *et al.* on a control-flow analysis for message authentication in *Lysa* [8, 9] is also strongly related to our present approach. The motivations and goals, however, are different, since message authentication concerns the origin of a message while agreement provides guarantees about the presence in the current session of the claimant and its willingness to authenticate with the verifier.

As far as compositionality is concerned, we find interesting to mention a logic-based approach [16, 17] to cryptographic protocol analysis proposed by Datta and others: relying on some protocol invariants, authors develop a modular reasoning for secure protocol composition, ensuring that protocols that are proved to be individually secure do not interact insecurely when they are composed with other protocols. In [26], Hasebe and Okada refine this approach for proving that a certain ordering among actions performed by principals holds in every protocol execution. A comparison with

these works is left as future work.

In [15], we compare our type and effect system with the one by Gordon and Jeffrey, drawing on a translation of tagged protocols, validated by our system, into protocols that type check with Gordon and Jeffrey's system; this work shows that our tags can be compiled even in the static types of [21]. This allows the tag inference procedure of [19] to be exploited for inferring such types.

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. *Theor. Comput. Sci.*, 298(3):387–415, 2003.

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

[4] M. Abadi and C.Fournet. Private authentication. In *Proceedings of the 2002 Workshop on Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 27–40. Springer-Verlag, 2003.

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

[6] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[7] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proceedings of Foundations of Software Science and Computation Structures*, pages 136–152, 2003.

[8] 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.

[9] 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)*, ENTCS. Elsevier, 2004.

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

[11] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Logic in Computer Science*, pages 157–166, 1999.

[12] 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.

[13] M. Bugliesi, R. Focardi, and M. Maffei. Authenticity by tagging and typing. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 1–12, New York, NY, USA, 2004. ACM Press.

[14] 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.

[15] M. Bugliesi, R. Focardi, and M. Maffei. Analysis of typed-based analyses of authentication protocols. In *Proceedings of 18th IEEE Computer Security Foundations Workshop (CSFW 2005)*, pages 112–125, 2005.

[16] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *CSFW*, pages 109–125, 2003.

[17] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 11–23, New York, NY, USA, 2003. ACM Press.

[18] 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.

[19] R. Focardi, M. Maffei, and F. Placella. Inferring authentication tags. In *WITS '05: Proceedings of the 2005 workshop on Issues in the theory of security*, pages 41–49, New York, NY, USA, 2005. ACM Press.

[20] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *Journal of Computer Security*, 11(4):451–519, 2004.

[21] A. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3/4):435–484, 2004.

[22] A. Gordon and A. Jeffrey. Secrecy despite compromise: types, cryptography, and the pi-calculus. In *Proceedings of Concur 2005*, pages 186–201. Springer-Verlag, September 2005.

[23] J. D. Guttman and F. J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[24] J.D. Guttman, F.J. Thayer, J.A. Carlson, J.C.Herzog, J.D. Ramsdell, and B.T. Sniffen. Trust management in strand spaces: a rely-guarantee method. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag, 2004.

[25] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Computer Society Press, July 2000.

[26] K. Hasebe and M. Okada. Non-monotonic properties for proving correctness in a framework of compositional logic. In *Proceedings of Foundations of Computer Security Workshop (FCS'04)*, pages 97–113, July 2004.

[27] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 255–268. IEEE Computer Society Press, July 2000.

[28] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[29] 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.

[30] G. Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[31] C. Meadows and P. Syverson. Formal specification and analysis of the group domain of intrepretation protocol using npatrl and the nrl protocol analyzer, 2003. to appear in Journal of Computer Security.

[32] Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. *Lecture Notes in Computer Science*, 2052:21–36, 2001.

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

[34] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. Journal of Computer Security, 1999. 7(2/3).

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

[36] 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 The ρ-Spi Semantics

The dynamics of ρ-spi is formalized by means of a transition relation between *configurations*, i.e., pairs $\langle s, P\rangle$, where $s \in Act^*$ 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. The transitions involving a sequential process preserve the identity identifiers associated with the process, as in $\langle s, I \triangleright \pi.S\rangle \vdash \langle s :: \alpha, I \triangleright S\rangle$, where α is the action corresponding to the primitive π.

Table 12 Transition System for ρ-spi

Transition rules: We omit the symmetric rule of PAR.
M,N,K ranges over terms: $M ::= n,x,(M,N),\{M\}_K,\{|M|\}_{\mathsf{Key}(K)},\mathsf{key}(M),TAG(M)$

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

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

OUTPUT
$$\langle s,I \triangleright \mathsf{out}(M).S\rangle \rightarrow \langle s :: out(M),I \triangleright S\rangle$$

BEGIN
$$\langle s,I \triangleright \mathsf{begin}_N(I,J,M_1;M_2).S\rangle \rightarrow \langle s :: begin_N(I,J,M_1;M_2),I \triangleright S\rangle$$

END
$$\langle s,I \triangleright \mathsf{end}_N(I,J,M_1;M_2).S\rangle \rightarrow \langle s :: end_N(I,J,M_1;M_2),I \triangleright S\rangle$$

INPUT
$$\frac{s \vdash M' \qquad \sigma = m.g.u.(M',M) \neq \uparrow}{\langle s,I \triangleright \mathsf{in}(M).S\rangle \rightarrow \langle s :: in(M'),I \triangleright S\sigma\rangle}$$

DECRYPT
$$\frac{\sigma = m.g.u.(M',M) \neq \uparrow}{\langle s,I \triangleright \mathsf{decrypt}\ M'\ \mathsf{as}\ M.S\rangle \rightarrow \langle s :: decrypt\ M',I \triangleright S\sigma\rangle}$$

ENCRYPT
$$\langle s,I \triangleright \mathsf{encrypt}\ M\ \mathsf{as}\ x.S\rangle \rightarrow \langle s :: encrypt\ M,I \triangleright S[M/x]\rangle$$

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

SYMMETRIC KEY
$$\frac{k \notin \mathrm{bn}(s) \cup \mathrm{fn}(s)}{\langle s,\mathsf{let}\ k = \mathsf{sym\text{-}key}(I_1,I_2).P\rangle \rightarrow \langle s :: sym-key(k,I_1,I_2),P\rangle}$$

ASYMMETRIC KEY
$$\frac{k \notin \mathrm{bn}(s) \cup \mathrm{fn}(s)}{\langle s,\mathsf{let}\ k = \mathsf{asym\text{-}key}(I).P\rangle \rightarrow \langle s :: asym-key(k,I),P\rangle}$$

Table 13 Most General Unifier

M is a ground term, while N may be composed of variables.

Most General Unifier

$$m.g.u.(M,M) = []$$
$$m.g.u.(TAG(M),TAG(N)) = m.g.u.(M,N) \quad \text{if } M \neq N$$
$$m.g.u.(\{M\}_K,\{N\}_K) = m.g.u.(M,N) \quad \text{if } M \neq N$$
$$m.g.u.(\{|M|\}_{\mathsf{key(K)}},\{|N|\}_{\overline{\mathsf{key}}(k)}) = m.g.u.(M,N)$$
$$m.g.u.(M,x) = [M/x] \quad \text{if } M = n, \{M'\}_K, \{|M'|\}_K, \mathsf{Pub}(K), \mathsf{Priv}(K)$$
$$m.g.u.((M,N),(M',N')) = m.g.u.(M,M') \uplus m.g.u.(N,N')$$
$$m.g.u.(M,N) = \uparrow \quad \text{otherwise}$$

Union

$$\sigma_1 \uplus \sigma_2 = \sigma_1 \cup \sigma_2 \quad x \in dom(\sigma_1) \cap dom(\sigma_2) \Rightarrow \sigma_1(x) = \sigma_2(x)$$
$$\sigma_1 \uplus \sigma_2 = \uparrow \quad \text{otherwise}$$

Table 14 Message Manipulation Rules

OUT
$$\frac{out(M) \in s}{s \vdash M}$$

PAIR
$$\frac{\forall i \in [1,2] \qquad s \vdash M_i}{s \vdash (M_1,M_2)}$$

PAIR DES
$$\frac{s \vdash (M_1,M_2)}{s \vdash M_i \qquad \forall i \in [1,2]}$$

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

TAG
$$\frac{s \vdash M}{s \vdash TAG(M)}$$

TAG DES
$$\frac{s \vdash TAG(M)}{s \vdash M}$$

KEY PAIR
$$\frac{s \vdash n}{s \vdash \mathsf{Pub}(n), \mathsf{Priv}(n)}$$

SYMMETRIC ENCRYPTION
$$\frac{s \vdash M \qquad s \vdash K}{s \vdash \{M\}_K}$$

ASYMMETRIC ENCRYPTION
$$\frac{s \vdash M \qquad s \vdash \mathsf{Key}(K)}{s \vdash \{|M|\}_{\mathsf{Key}(K)}}$$

SYMMETRIC DECRYPTION
$$\frac{s \vdash \{M\}_K \qquad s \vdash K}{s \vdash M}$$

ASYMMETRIC DECRYPTION
$$\frac{s \vdash \{|M|\}_{\mathsf{key}(K)} \qquad s \vdash \overline{\mathsf{key}}(K)}{s \vdash M}$$

PUBLIC KEYS
$$\frac{asym - key(k,I) \in s}{s \vdash \mathsf{Pub}(k)}$$

ENEMY KEYS
$$\frac{asym - key(k,E) \in s \vee sym - key(k,E,I) \in s}{s \vdash k}$$

The set of all possible actions, noted Act, includes the action $out(M)$ generated by output, $begin_N(I,J,M_1;M_2)$ and $end_N(I,J,M_1;M_2)$ by 'begin' and 'end', $in(M)$ by input, *decrypt M* by decryption, *encrypt M* by encryption, $fresh(n)$ by restriction and *sym-key*(k,I_1,I_2) and *asym-key*(k,I) by symmetric and asymmetric key assignment.

Some transitions apply substitutions to processes: formally, a substitution $\sigma : x \mapsto M$ is a function from variables to ground terms. Often substitutions are written explicitly by $[N_1/x_1,\ldots,N_n/x_n]$. The application of the substitution σ to the process S is denoted by $S\sigma$ and applies only to free occurrences of the variables in S.

The transitions, in Table 12, are mostly standard: PAR interleaves two different protocol executions, REPLICATION arbitrarily replicates a principal and OUTPUT allows a process to send a message on the network. BEGIN and END are self-explanatory. INPUT requires messages 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). Moreover, input messages are either bound to variables or tested by pattern-matching (explained below), a capability that is also available upon decryption (cf. DECRYPT). ENCRYPT behaves as expected. Notice that the generation of both names (RES) and keys (SYMMETRIC and ASYMMETRIC KEY) are formalized as semantic transitions in which the freshly generated name/key is required to be different from all the already used names.

Pattern-matching is formalized by the notion of most general unifier, defined in Table 13: the most general unifier takes as input a ground term and a term possibly containing variables and yields a substitution. Specifically, the most general unifier yields the empty substitution when applied to equal terms and the most general unifier between M and N when applied to $TAG(M)$ and $TAG(N)$, where the tag TAG is the same: the side condition $M \neq N$ makes the function $m.g.u.$ deterministic.

The most general unifier between two ciphertexts is more elaborate, as pattern-matching means decrypting and this requires the decryption key in the second ciphertext to correspond to the encryption key in the first one. If this happens, then the result is the most general unifier between the two ciphertext contents; otherwise, the most general unifier is undefined. It is worth noting that in ρ-spi calculus signature and encryption are the inverse function of each other.

The most general unifier between a ground term M and a variable x yields the substitution $[M/x]$. Notably, we prevent variables to be substituted by tagged terms, since tags are crucial patterns for authentication and we require them to be explicitly indicated. Also key-pairs cannot be bound to variables as to prevent private keys to be deduced by the corresponding public key. The most general unifier between the pairs (M,N) and (M',N') returns the union $\uplus$ of the most general unifier between M and N and the one between M' and N'. In all the other cases the most general unifier is undefined. The union between two most general unifiers is just the union of the substitutions yielded by each of them if the two most general unifiers substitute the same variable with the same term, otherwise their union is undefined.

We use a number of notation conventions. The restriction operator $\mathsf{new}(n).S$ is a binder for name n, the key declarations $\mathsf{let}\ k = \mathsf{sym\text{-}key}(I_1,I_2)$ and $\mathsf{let}\ k = \mathsf{asym\text{-}key}(I)$ are binders for k, while the input and decryption primitives are binders for the variables that occur in components M_i; finally, encryption is a binder for variable x. In all cases the scope of the binders is the continuation process. Similarly, $fresh(n)$, $sym-key(k,I)$

and $asym-key(k,I_1,I_2)$ are binders for names and their scope is the continuation trace. The notions of free/bound names and variables, both for processes and traces, arise as expected. As in companion transition systems, see, e.g. [11], we identify processes up to renaming of bound variables and names, i.e., up to α-equivalence.

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. By PAIR and PAIR DES, the environment may construct and destruct pairs. ENV allows the environment to generate a new bound name, not occurring in the trace. TAG and TAG DES allow the environment to tag and untag messages. By KEY PAIR, given a term M, the environment can build the private and public component of M. By SYMMETRIC ENCRYPTION, if the environment knows M and K, then it can encrypt M with K; by ASYMMETRIC ENCRYPTION, the environment may encrypt messages using asymmetric cryptography. SYMMETRIC DECRYPTION and ASYMMETRIC DECRYPTION formalize the capability of the environment to decrypt ciphertexts, whose decryption key is known. By PUBLIC KEYS, all the public keys are known by the environment. Moreover, by ENEMY KEYS, the environment may be provided with own key pairs and with long-term keys shared among the other 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. with the corresponding public one and vice-versa As an example, let us consider the following transitions :

$$\begin{array}{l}\langle s, A \triangleright \mathsf{in}(y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{|n|\}_{\mathsf{Priv}(k_A)}.\mathbf{0}\rangle \rightarrow \\ \langle s :: in(\{|n|\}_{\mathsf{Pub}(k_A)}), A \triangleright \mathsf{decrypt}\ \{|n|\}_{\mathsf{Pub}(k_A)}\ \mathsf{as}\ \{|n|\}_{\mathsf{Priv}(k_A)}.\mathbf{0}\rangle \rightarrow \\ \langle s :: in(\{|n|\}_{\mathsf{Pub}(k_A)}) :: decrypt\{|n|\}_{\mathsf{Pub}(k_A)}, \mathbf{0}\rangle\end{array}$$

where $asym-key(k_A,A) \in s$. A reads the message $\{|n|\}_{\mathsf{Pub}(k_A)}$ from the environment and the most general unifier for input is $[\{|n|\}_{\mathsf{Pub}(k_A)}/y]$. Hence A decrypts the ciphertext by her own private key $\mathsf{Priv}(k_A)$. The operation is successful only if the ciphertext is encrypted with the corresponding public-key and the encrypted message is equal to n (by pattern-matching).

A.1 Typed ρ-spi calculus

As discussed in Section 4, ρ-spi calculus is equipped with two typed primitives, namely a typed restriction and a cast operator. These primitives are needed by the type and effect system but have no significant computational import:

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

CAST
$$\langle s, I \triangleright \mathsf{cast}\ M\ \mathsf{is}\ (x:T).S\rangle \rightarrow \langle s :: cast(M), S[M/x]\rangle$$

TYPED RES behaves as RES while the process $\mathsf{cast}\ M\ \mathsf{is}\ (x:T).S$ reduces to $S[M/x]$.

B Soundness of the ρ-Spi Type and Effect System

In Section B.1 and Section B.2, we prove the soundness and the safety, respectively, of the ρ-spi type and effect system.

B.1 ρ-Spi Soundness

In the following, the substitution $\Gamma[n : T/n : T']$ denotes the cast of n's type in Γ from T' to T. If n is not present in Γ or it has type different from T', then such a substitution does not influence Γ. Moreover, $\jmath$ ranges over $\{\diamond, M : T, P : e, S : e\}$ so that $[I;]\Gamma \vdash \jmath$ denotes an arbitrary judgement.

The Weakening Lemma states that if a typing environment Γ proves a judgement $\jmath$, then every well-formed extension of Γ still proves $\jmath$. Intuitively, if Γ proves $\jmath$, then Γ contains all the typing assumptions needed for proving $\jmath$: hence extending the set of typing assumptions means extending the set of provable judgements.

Lemma 1 (Weakening) *If* $[I;]\Gamma, \Gamma'' \vdash \jmath$ *and* $\Gamma, \Gamma', \Gamma'' \vdash \diamond$, *then* $[I;]\Gamma, \Gamma', \Gamma'' \vdash \jmath$.

Proof. Trivial as the only typing rule checking the absence of a term in the typing environment is GOOD ENV and, by hypothesis, $\Gamma, \Gamma', \Gamma'' \vdash \diamond$. The proof is by straightforward induction on the length of the typing derivation of $[I;]\Gamma, \Gamma'' \vdash \jmath$. ■

The Substitution Lemma is a standard tool for proving the preservation of types at runtime. In fact, as long as a process evaluates, variables are instantiated to dynamic terms: the Substitution Lemma states that typing judgements are preserved by the substitution of variables with terms having the same type.

Lemma 2 (Substitution) *If* $[I;]\Gamma, x : T \vdash \jmath$ *and* $\Gamma \vdash M : T$, *then* $[I;]\Gamma[M/x] \vdash \jmath[M/x]$

Proof. Trivial as typing rules rely on judgements having the form $\Gamma \vdash M : T$, where M is an arbitrary term (not necessarily a variable) and T is the type derived for M (not necessarily the exact type of M in Γ). The proof is by straightforward induction on the length of the typing derivation of $[I;]\Gamma, x : T \vdash \jmath$. ■

In the following, we write $names(N)$ to denote the names in the term N. Similarly, we write $names(e)$ and $var(e)$ to denote the set of names and variables in the effect e, respectively. The following rule says that an effect e is well-formed according to a typing environment Γ, if the names and the variables which e depends on are in the domain of Γ.

$$\frac{\text{GOOD EFFECT} \quad names(e) \cup var(e) \subseteq dom(\Gamma)}{\Gamma \vdash e}$$

The following lemma says that if a judgement can be proved according to a typing environment, then the same judgment can be proved even if a cast from $\mathsf{Nonce}^{Ciph}(I, J)$ to Un occurs into the typing environment.

Lemma 3 (Typability under DownCasting) *If* $[I;]\Gamma, n : \mathsf{Nonce}^{Ciph}(I,J), \Gamma' \vdash \mathcal{J}$*, then* $[I;]\Gamma, n : \mathsf{Un}, \Gamma' \vdash \mathcal{J}$.

Proof. Trivial, by SUBSUMPTION and by straightforward induction on the length of the typing derivation of $[I;]\Gamma, n : \mathsf{Nonce}^{Ciph}(I,J), \Gamma' \vdash \mathcal{J}$. ∎

Moreover, we write $N \in terms(M)$ if M is built upon N: for instance, $\mathsf{Auth}(m) \in terms(\{\mathsf{Auth}(m), M\}_k)$, $n \in terms(\mathsf{Verif}(n))$ and $k \in terms(\{M\}_k)$. Similarly, we write $terms(s)$ and $terms(P)$ to denote the set of terms in the trace s and the process P, respectively. The only subtlety regards encryptions and decryptions: indeed, we define $terms(\mathsf{encrypt}\ \{M\}_K\ \mathsf{as}\ z.S)$ and $terms(\mathsf{decrypt}\ z\ \mathsf{as}\ \{M\}_K.S)$ as $terms(M) \cup terms(K) \cup terms(S)$, thus not considering $\{M\}_K$ as a term. Similar reasoning applies to encryptions and decryptions through asymmetric cryptography. There is a bit of overloading in the notation $\{M\}_K$: in encryptions and decryptions it should be understood as a piece of syntax specifying the ciphertext content M and the decryption key K, while in all the other contexts it is a previously generated dynamic term. For instance, $\{n\}_k \in terms(s :: encrypt\{n\}_k)$ but $\{n\}_k \notin terms(\mathsf{encrypt}\ \{n\}_k\ \mathsf{as}\ z.\mathbf{0})$. For easing the presentation we bear this overloading as the context makes clear the meaning of $\{M\}_K$.

Definition 3 (Traces and Typing Environment Consistency) *Let s be an execution trace and Γ a typing environment. We say that s and Γ are consistent, written $s \prec \Gamma$, if the following conditions hold:*

1. $dom(\Gamma) = bn(s) \cup fn(s) \cup \{I_1, \ldots, I_m\}$
2. $n \in fn(s) \cup \{I_1, \ldots, I_m\} \Rightarrow \Gamma(n) = \mathsf{Un}$
3. $s \vdash N \Rightarrow \Gamma, \overline{m} : \mathsf{Un} \vdash N : \mathsf{Un}$, *with* $\overline{m} = names(N) \setminus dom(\Gamma)$.

Intuitively, an execution trace s and a typing environment Γ are consistent if:

1. The domain of Γ is composed of the names in the trace s plus some identity labels.
2. Both the free names in s and the identity labels have type Un.
3. Every term in the knowledge of the environment (namely the set of terms derived from s according to the rules in Table 14) has type Un once untagged. This item says that the environment may guess only untrusted terms, i.e., the secrets in Γ are not leaked out.

Notice that, for typing a term N in the knowledge of the environment, Γ is extended with a set of untrusted names. Indeed, the environment may generate fresh names and use free names not in s, thus not belonging to the domain of Γ.

The following lemma states that if s and Γ are consistent, then they are consistent even after the output of untrusted terms.

Lemma 4 (Output and Environment Knowledge) *Let Γ be a typing environment, s a trace and M a term such that $s \prec \Gamma$ and $\Gamma \vdash M : \mathsf{Un}$. Then $s :: out(M) \prec \Gamma$.*

Proof. For proving $s :: out(M) \prec \Gamma$, we need to prove the third condition, namely $s :: out(M) \vdash N \Rightarrow \Gamma, \overline{m} : \mathsf{Un} \vdash N : \mathsf{Un}$, with $\overline{m} = fn(N) \setminus dom(\Gamma)$. Indeed, the first and second conditions are trivially satisfied. Let $s' = s :: out(M)$ and $\Gamma' = \Gamma, \overline{m} : \mathsf{Un}$. The proof follows by induction on the length of the derivation of $s' \vdash N$. We divide the base case according to the message manipulation rule applied:

Env Hence $N = n$, i.e., N is a name. By ENV, $s' \vdash n$ and $n \notin bn(s')$. Thus $n \notin bn(s)$ and, by ENV, $s \vdash n$. By induction hypothesis, $\Gamma' \vdash n : \mathsf{Un}$. The proof for PUBLIC KEYS and ENEMY KEYS follows from a similar reasoning.

Out Let us suppose by contradiction that $\Gamma' \nvdash N : \mathsf{Un}$. Since $s \prec \Gamma$, we get $N = M$, i.e., N is the last name sent on the network. By hypothesis, $\Gamma \vdash M : \mathsf{Un}$ and, by Lemma 1, $\Gamma' \vdash M : \mathsf{Un}$. Since $N = M$, $\Gamma' \vdash N : \mathsf{Un}$, giving a contradiction.

The proof of the induction step proceeds by cases according to the last message manipulation rule applied (we just discuss some interesting cases):

Asymmetric Decryption We reason by contradiction: let us suppose

$$s' \vdash \{|M|\}_{\mathsf{Key}(K)} \quad s' \vdash \overline{\mathsf{Key}}(K) \quad \Gamma' \nvdash M : \mathsf{Un}$$

The judgement $\Gamma' \vdash \{|M|\}_{\mathsf{Key}(K)} : \mathsf{Un}$ may be proved either by CIPHERTEXT (via SUBSUMPTION) or by UNTAGGED CIPHERTEXT (via SUBSUMPTION) or by UN ASYMM CIPH. Only the former typing rule is interesting, as the other ones require $\Gamma' \vdash M : \mathsf{Un}$, giving a contradiction.

By CIPHERTEXT, $M = (R_H(N), M')$, where M' has type Un but N might have not such type. Let us suppose $H = \mathsf{CC?}$: the other cases are similar. We distinguish two cases depending on whether the encryption key is public or private.

$\mathsf{Key} = \mathsf{Pub}$ According to CIPHERTEXT, $\Gamma' \vdash N : \mathsf{Nonce}^{Priv}(I,J)$ and $\Gamma' \vdash \mathsf{Key}(K) : \mathsf{PublicKey}(J)$. Let us suppose $I, J \notin \mathcal{ID}_{\mathcal{E}}$, otherwise $\Gamma' \vdash N : \mathsf{Un}$ by SUBSUMPTION (Public Names). Since $\Gamma' \vdash \overline{\mathsf{Key}}(K) : \mathsf{PrivateKey}(J)$, by induction hypothesis, $s' \nvdash \overline{\mathsf{Key}}(K)$, giving a contradiction.

$\mathsf{Key} = \mathsf{Priv}$ By CIPHERTEXT, $\Gamma' \vdash N : \mathsf{Nonce}^{Int}(I,J)$ and $\Gamma' \vdash N : \mathsf{Un}$ by SUBSUMPTION (Public Nonce). By PAIR, $\Gamma' \vdash M : \mathsf{Un}$, giving a contradiction.

Pair Des Let us suppose by contradiction that $s' \vdash (M,N)$ and $\Gamma' \nvdash N : \mathsf{Un}$. Indeed, the reasoning for $\Gamma' \nvdash M : \mathsf{Un}$ is the same. By induction hypothesis $\Gamma' \vdash (M,N) : \mathsf{Un}$ and, by the typing rule UNTRUSTED PAIR, $\Gamma' \vdash N : \mathsf{Un}$, thus giving a contradiction. ∎

Intuitively, the following lemma says that if the environment knows a term containing a ciphertext that cannot be generated by the environment, then such a ciphertext appears in the execution trace. For sake of readability, in the rest of the proof we do not distinguish between ciphertexts originated through symmetric and asymmetric cryptography, namely we write $\{M\}_K$ to denote both the kinds of ciphertexts.

Lemma 5 (Ciphertexts and Traces) *If $s \vdash M$, $\{N\}_K \in terms(M)$ and either $s \nvdash N$ or $s \nvdash K$, then $\{N\}_K \in terms(s)$.*

Proof. By induction on the length of the derivation of $s \vdash M$. We divide the base case according to the message manipulation rule applied:

Env Trivial, as $M = n$ and $\nexists N, K$ such that $\{N\}_K \in terms(n)$. The reasoning for PUBLIC and ENEMY KEYS is the same.

Out s contains the action $out(M)$ and, since $\{N\}_K \in terms(M)$, $\{N\}_K \in terms(s)$ as desired.

The inductive step proceeds by cases, according to the last message manipulation rule applied. We just discuss the most interesting cases:

Pair The rule proves $s \vdash M = (M_1, M_2)$. Let us suppose that $\{N\}_K \in terms(M_1)$. If $\{N\}_K \in terms(M_2)$, then the reasoning is the same. By PAIR, $s \vdash M_1$ and, by induction hypothesis, $\{N\}_K \in terms(s)$.

Symmetric Decryption The rule proves $s \vdash N'$, given that $s \vdash K'$ and $s \vdash \{N'\}_{K'}$. Since either $s \nvdash N$ or $s \nvdash K$, $\{N\}_K \neq \{N'\}_{K'}$. Let us suppose that $\{N\}_K \in terms(N')$ (if $\{N\}_K \in terms(K')$, then the reasoning is similar). Thus $\{N\}_K \in terms(\{N'\}_{K'})$ and, by induction hypothesis, $\{N\}_K \in terms(s)$.

Symmetric Encryption The rule proves $s \vdash \{N'\}_{K'}$, if $s \vdash N'$ and $s \vdash K'$. Hence, we get $\{N\}_K \neq \{N'\}_{K'}$. Thus either $\{N\}_K \in terms(N')$ or $\{N\}_K \in terms(K')$: by induction hypothesis, $\{N\}_K \in terms(s)$. ∎

Here and for the rest of the paper, we write $|\alpha|_s$ to denote the number of occurrences of the action α in the trace s. Similarly, we write $|t|_e$ to denote the number of occurrences of the atomic effect t in the effect e.
The following definition provides some invariants on traces, processes, typing environments and effects. Theorem 3 (Subject Reduction) proves that the evaluation of well-typed processes preserve these invariants and Theorem 1 (Safety) exploits this result in order to prove that well-typed processes are safe.

Definition 4 (Balanced Configuration) *Let I, J be identity labels in $\mathcal{ID}_{\mathcal{P}}$ and s be an execution trace. Let Γ be a typing environment, P a process and e an effect such that $\Gamma \vdash P : e$. We say that s, P, Γ and e are balanced iff the following conditions hold:*

1. $s \prec \Gamma$
2. $|\mathsf{fresh}^{\ell}(n)|_e + |end_n(\cdot)|_s \leq 1$
3. *If N is either $\{M\}_K$ or $\{|M|\}_K$ and $N \in terms(P)$, then $N \in terms(s)$*
4. $P = P_1|P_2 \Rightarrow \exists e_1, e_2$ *s.t.* $e_1 + e_2 = e \ \wedge\ s, P_i, \Gamma, e_i$ *are balanced,* $\forall i \in [1,2]$
5. *If* $?\mathsf{Resp}_N^{Un}() \in e$, *then* $\Gamma \vdash N : \mathsf{Un}$.

6. $[?|!]e_R \in e \ \vee \exists M_R \in terms(s)\ s.t.\ \Gamma \vdash M_R : \mathsf{Enc}(\cdot; e_R)$
$\Rightarrow$
$\wedge \left(\begin{array}{l} begin_N(J,I,M_1;M_2) \in s \\ \exists M_C \in terms(s)\ s.t.\ \Gamma \vdash M_C : \mathsf{Enc}(e_C;\cdot) \end{array} \right)$

$\Gamma(N) = \mathsf{Nonce}^{\ell}(I,J)$
$\ell \in \{Priv, Int\}$
$e_R = \mathsf{Resp}^{\overline{\ell}}_N(J,I,M_2)$
$e_C = \mathsf{Chal}^{\ell}_N(I,J,M_1)$

7. $\vee \left(\begin{array}{l} \mathsf{fresh}^{Ciph}(N) \in e \wedge \Gamma \vdash N : \mathsf{Un} \\ !e_R \in e \end{array} \right)$
$\Rightarrow$
$\wedge \left(\begin{array}{l} begin_N(J,I,M) \in s \\ \exists M_C \in terms(s)\ s.t.\ \Gamma \vdash M_C : \mathsf{Enc}(e_C;\cdot) \end{array} \right)$

$\Gamma \vdash N : \mathsf{Nonce}^{Ciph}(I,J)$
$e_C = \mathsf{Chal}^{Ciph}_N(I,J,M)$
$e_R = \mathsf{Resp}^{Un}_N()$

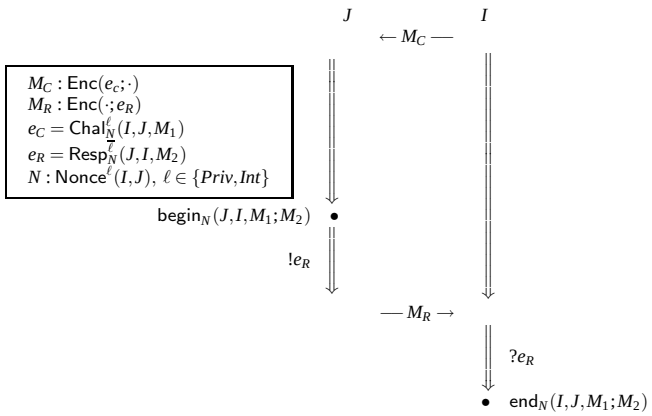
8. $[!|?]e_R \in e \vee \exists M_R \in terms(s)\ s.t.\ \Gamma \vdash M_R : \mathsf{Enc}(\cdot; e_R)$
$\Rightarrow$ $begin_N(J,I;M) \in s$

$\Gamma \vdash N : \mathsf{Nonce}^{Un}(I,J)$
$e_R = \mathsf{Resp}^{Ciph}_N(J,I,M)$

9. $?e_C \in e$
$\Rightarrow$ $\exists M_C \in terms(s)\ s.t.\ \Gamma \vdash M_C : \mathsf{Enc}(e_C;\cdot)$
$\mathsf{fresh}^{\ell}(N) \in e \ \wedge \ \exists M_C \in terms(s)\ s.t.\ \Gamma \vdash M_C : \mathsf{Enc}(e_C;\cdot)$
$\Rightarrow$ $!e_C \in e$

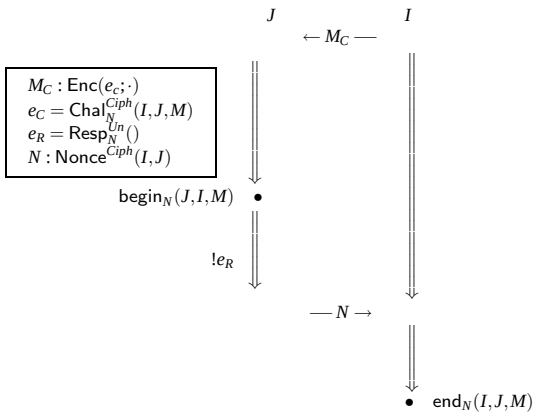
$\Gamma(N) = \mathsf{Nonce}^{\ell}(I,J)$
$\ell \in \{Ciph, Priv, Int\}$
$e_C = \mathsf{Chal}^{\ell}_N(I,J,M)$

The intuitive reading of the balancing conditions is as follows:

1. The execution trace and the typing environment are consistent.
2. Either a nonce is fresh or one $end(\cdot)$ has been asserted by checking its freshness.
3. Every ciphertext in the terms of P appears in the trace.
4. Balancing is propagated through parallel composition.
5. If a term has been received in clear, then it has type Un.
6. 

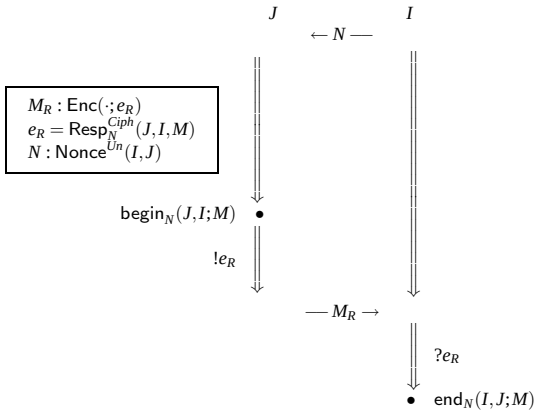
This balancing condition provides an invariant on CC handshakes. If a response with nonce N and message M_2 is either received ($?e_R \in e$), or justified ($!e_R \in e$), or sent on the network ($M_R \in terms(s)$), then $begin_N(J,I,M_1;M_2)$ has been asserted and a challenge with nonce N and message M_1 has been sent on the network ($M_C \in terms(s)$).

7.



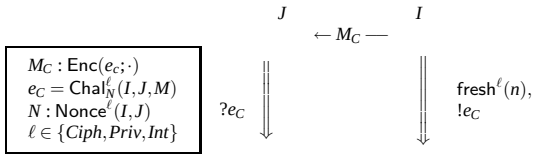
This balancing condition provides an invariant on CP handshakes. If either the nonce is fresh ($\mathsf{fresh}^{Ciph}(N) \in e$) and has type Un ($\Gamma \vdash N : \mathsf{Un}$) or the output in clear of the nonce N is justified ($!e_R \in e$), then $begin_N(J,I,M_1)$ has been asserted and a challenge with nonce N and message M_1 has been sent on the network ($M_C \in terms(s)$).

8.



This balancing condition provides an invariant on PC handshakes. If a response with nonce N and message M is either received ($?e_R \in e$), or justified ($!e_R \in e$), or sent on the network ($M_R \in terms(s)$), then $begin_N(J,I;M)$ has been asserted.

9.



This balancing condition is composed of two implications. The former states that if a challenge with nonce N and message M has been received ($?e_C \in e$), then such a challenge has been sent on the network ($M_C \in terms(s)$). The latter states that if the nonce N is fresh ($\mathsf{fresh}^{\ell}(N) \in e$) and a challenge with nonce N

and message M is sent on the network ($M_C \in terms(s)$), then such a challenge has been generated by an honest principal ($!e_C \in e$).

Condition 2 is used in the proof of Theorem 1 for verifying the freshness of end assertions, while conditions 5, 6, 7, 8 and 9 are used for proving that each end assertion is preceded by the corresponding begin assertion. The other conditions are needed in the proof of Theorem 3 in order to show that the previous ones are preserved during process reduction.

The Subject Reduction Theorem proves that typability and balancing conditions are preserved at run-time.

Theorem 3 (Subject Reduction) *Let P and Q be processes, Γ a typing environment, s a trace and e an effect such that*

- $\Gamma \vdash \diamond$ *and* $\Gamma \vdash e$
- $\langle s, Q\rangle \rightarrow \langle s :: \alpha, P\rangle$
- $\Gamma \vdash Q : e$
- s, Q, Γ, e *are balanced*

then it is possible to find an environment Γ' and an effect e' such that

- $\Gamma' \vdash \diamond$ *and* $\Gamma' \vdash e'$
- $\Gamma' = \Gamma, names(\alpha) \setminus names(s) : T$ *or* $\Gamma[n : \mathsf{Un}/n : \mathsf{Nonce}^{Ciph}(I,J)]$
- $\Gamma' \vdash P : e'$
- $s :: \alpha, P, \Gamma', e'$ *are balanced*

Proof. We reason by induction on the length of the derivation for $\langle s, Q\rangle \rightarrow \langle s :: \alpha, P\rangle$ (if Q is a parallel composition of processes, then such a derivation is composed of more than one step). The proof of the base case proceeds by cases, according to the semantic rule applied.

REPLICATION $\boxed{\Gamma \vdash I \triangleright !S : [] \quad \rightarrow \quad \Gamma \vdash I \triangleright !S | I \triangleright S : []}$

REPLICATION proves $\Gamma \vdash I \triangleright !S : []$ only if $\Gamma \vdash I \triangleright S : []$. Since the type and effect system is syntax directed, typing rules can be read upside-down. Thus, from $\Gamma \vdash I \triangleright !S : []$ we derive $\Gamma \vdash I \triangleright S : []$ and, by PAR, $\Gamma \vdash I \triangleright !S | I \triangleright S : []$.

SYMMETRIC KEY $\boxed{\begin{array}{l}\Gamma \vdash \mathsf{let}\ k = \mathsf{sym-key}(I_1, I_2).P : [] \quad \rightarrow \\ \Gamma, k : \mathsf{SharedKey}(I_1, I_2) \vdash P : []\end{array}}$

The typed reduction depicted above follows from SYMMETRIC KEY.

ASYMMETRIC KEY

$$\Gamma \vdash \mathsf{let}\ k = \mathsf{asym-key}\ (I).P : [] \quad \rightarrow \quad \Gamma, k : \mathsf{Key}(I) \vdash P : []$$

The typed reduction depicted above follows from ASYMMETRIC KEY.

NEW

$$\Gamma \vdash A \triangleright \mathsf{new}(n : \mathsf{Nonce}^{\ell}(A,I)).S : e - [\mathsf{fresh}^{\ell}(n)] \quad \rightarrow \quad \Gamma, n : \mathsf{Nonce}^{\ell}(A,I) \vdash A \triangleright S : e$$

By IDENTITY and NEW, $A;\Gamma \vdash \mathsf{new}(n : \mathsf{Nonce}^{\ell}(A,I)).S : e - [\mathsf{fresh}^{\ell}(n)]$ and $A;\Gamma, n : \mathsf{Nonce}^{\ell}(A,I) \vdash S : e$. Applying IDENTITY, $\Gamma, n : \mathsf{Nonce}^{\ell}(A,I) \vdash A \triangleright S : e$.

Let $e' \in \{e, e + [\mathsf{fresh}^{\ell}(n)]\}$. We need to prove that the target configuration composed of $s :: fresh(n), A \triangleright S, \Gamma, n : \mathsf{Nonce}^{\ell}(A,I), e'$ is still balanced.

Conditions 1 We need to prove that $s :: fresh(n) \prec \Gamma, n : \mathsf{Nonce}^{\ell}(A,I)$. We prove each of the conditions in Definition 3.

1. By hypothesis $s \prec \Gamma$. Hence $dom(\Gamma) = fn(s) \cup bn(s) \cup \{I_1, \ldots, I_m\}$. The result follows from observing that $n \in bn(s :: fresh(n))$.
2. Trivial, by observing that $s \prec \Gamma$ and $n \notin fn(s :: fresh(n)) \cup \{I_1, \ldots, I_m\}$.
3. Trivial, by the balancing of the source configuration and by observing that $s :: fresh(n) \vdash N \Rightarrow s \vdash N$.

Condition 2 By the semantic rule RES, $n \notin fn(s) \cup bn(s)$ and $|end_n(\cdot)|_s = 0$. Since $\Gamma \vdash e$, $n \notin names(e)$ and $|\mathsf{fresh}^{\ell}(n)|_e = 0$. Thus we obtain the equality $|\mathsf{fresh}^{\ell}(n)|_{e'} + |end_n(\cdot)|_{s :: fresh(n)} = 1$, as desired.

Condition 9 Let us suppose by contradiction that condition 9 does not hold on the target configuration, i.e., let us suppose that $\exists M_C \in terms(s)$ s.t. $\Gamma \vdash M_C : \mathsf{Enc}(\mathsf{Chal}^{\ell}_n(I,J,M);)$ and $!\mathsf{Chal}^{\ell}_n(I,J,M) \notin e$. Thus $n \in fn(s) \cup bn(s)$. By the semantic rule RES, $n \notin fn(s) \cup bn(s)$, giving a contradiction.

BEGIN

$$\Gamma \vdash A \triangleright \mathsf{begin}_N(A,I,M_1;M_2).S : e + [?\mathsf{Chal}^{\ell}_N(I,A,M_1)] \quad \rightarrow \quad \Gamma \vdash A \triangleright S : e + [!\mathsf{Resp}^{\overline{\ell}}_N(A,I,M_2)]$$

By IDENTITY and BEGIN, $A;\Gamma \vdash \mathsf{begin}_N(A,I,M_1;M_2).S : e + [?\mathsf{Chal}^{\ell}_N(I,A,M_1)]$ and $A;\Gamma \vdash S : e + [!\mathsf{Resp}^{\overline{\ell}}_N(A,I,M_2)]$. The result follows from IDENTITY.

We need to check that conditions 6, 7 and 8 still hold on the target configuration composed of $s :: begin_N(A,I,M_1;M_2), A \triangleright S, \Gamma, e + [!\mathsf{Resp}^{\overline{\ell}}_N(A,I,M_2)]$. (the proof for the other ones follows directly from observing that the source configuration is balanced.)

Conditions 6 and 7 Since the source configuration is balanced and the effect in the source configuration contains $?\mathsf{Chal}^{\ell}_N(I,A,M_1)$, condition 9 proves that $\exists M \in terms(s)$ such that $\Gamma \vdash M : \mathsf{Enc}([\mathsf{Chal}^{\ell}_N(I,A,M_1)];\cdot)$, as desired.

Condition 8 Let us suppose by contradiction that condition 8 does not hold on the target configuration. Thus $begin_N(A,I;M_2) \notin s :: begin_N(A,I,M_1;M_2)$, with $\Gamma(N) = \mathsf{Nonce}^{Un}(I,A)$. Hence, $M_1 \neq ()$. The contradiction arises as, by syntactic restriction on the challenge effect, $M_1 = ()$.

INPUT

$$\Gamma \vdash A \triangleright \mathsf{in}(M).S : e - [?\mathsf{Chal}_M^{Un}(), ?\mathsf{Resp}_M^{Un}()] \quad \rightarrow$$
$$\Gamma, \overline{m} : \mathsf{Un} \vdash A \triangleright S[U] : e[U]$$
$$\text{where } \sigma = m.g.u.(M', M),\ \alpha = in(M') \text{ and } \overline{m} = fn(M') \setminus dom(\Gamma)$$

IDENTITY and INPUT prove $A; \Gamma \vdash \mathsf{in}(M).S : e - [?\mathsf{Chal}_M^{Un}(), ?\mathsf{Resp}_M^{Un}()]$ and $A; \Gamma$, $vars(M) : \mathsf{Un} \vdash S : e$. By the semantic rule INPUT, $s \vdash M'$. By the balancing condition 1, $s \prec \Gamma$. Thus $\Gamma, \overline{m} : \mathsf{Un} \vdash M' : \mathsf{Un}$, where $\overline{m}$ are the names originated by the enemy by the message manipulation rule ENV (in Table 14), namely $\overline{m} = fn(M') \setminus dom(\Gamma)$. By Lemma 2 (the Substitution Lemma) and IDENTITY, $\Gamma, \overline{m} : \mathsf{Un} \vdash A \triangleright S\sigma : e\sigma$.

We need to prove that the target configuration is still balanced. The insertion of both $?\mathsf{Chal}_M^{Un}()$ and $?\mathsf{Resp}_M^{Un}()$ may break only condition 5. Furthermore, the trace and the process are modified and conditions 3, 6, 8 and 9 might fail.

Condition 3 The input may increase the set of terms in S. Since they appear in the action $in(M')$, condition 3 holds.

Condition 5 Trivial as every input message has type Un.

Conditions 6 and 8 Let us suppose by contradiction that condition 6 does not hold on the target configuration, i.e., $\exists M_R \in terms(s :: in(M'))$ such that $\Gamma \vdash M_R : \mathsf{Enc}(\cdot; [\mathsf{Resp}_N^{\overline{\ell}}(J,I,M_2)])$, with $\Gamma(N) = \mathsf{Nonce}^{\ell}(I,J)$ and $\ell \in \{Priv, Int\}$, and $\nexists M_C, M_1$ such that $begin_N(I,J,M_1;M_2) \in s$, $M_C \in terms(s)$ and $\Gamma \vdash M_C : \mathsf{Enc}([\mathsf{Chal}_N^{\ell}(I,J,M_1)]; \cdot)$. Since, by hypothesis, condition 6 holds on the source configuration, $M_R \notin terms(s)$.

By an inspection of the typing rules for ciphertexts $M_R = \{R_H(N), M'\}_K$. We prove that either $s \nvdash N$ or $s \nvdash K$: by Lemma 5, this implies $M_R \in terms(s)$, giving a contradiction. We distinguish two cases:

- $\Gamma(K) \in \{\mathsf{SharedKey}(I,J), \mathsf{PrivateKey}(I)\}$ By hypothesis, $s \prec \Gamma$ and, by the third condition of Definition 3, $s \nvdash K$.
- $\Gamma \vdash K : \mathsf{PublicKey}(J)$ By an inspection of *Typeofenc*, $\ell = Priv$. Hence $\Gamma(N) = \mathsf{Nonce}^{Priv}(I,J)$. By hypothesis, $s \prec \Gamma$ and, by the third condition of Definition 3, $s \nvdash N$.

The reasoning for condition 8 is the same.

Condition 9 Let us suppose by contradiction that $\exists M_C \in terms(s :: in(M'))$ such that $\Gamma \vdash M_C : \mathsf{Enc}([\mathsf{Chal}_N^{\ell}(I,J,M)]; \cdot)$, $\mathsf{fresh}^{\ell}(N) \in e$ and $!\mathsf{Chal}_N^{\ell}(I,J,M) \notin e'$, with $\Gamma(N) = \mathsf{Nonce}^{\ell}(I,J)$ and $\ell \in \{Ciph, Priv, Int\}$. Since condition 9 holds on the source configuration, $M_C \notin terms(s)$.

By an inspection of the typing rules for ciphertexts, $M_C = \{R_H(N), M'\}_K$. As in the previous item, it is easy to prove that either $s \nvdash N$ or $s \nvdash K$. By

Lemma 5, this implies $M_C \in terms(s)$, giving a contradiction.

OUTPUT $\Gamma \vdash A \triangleright \mathsf{out}(N).S : e - [!\mathsf{Chal}_N^{Un}()] \quad \rightarrow \quad \Gamma \vdash A \triangleright S : e$

By IDENTITY and OUTPUT, $A;\Gamma \vdash \mathsf{out}(N).S : e - [!\mathsf{Chal}_N^{Un}()]$ and $A;\Gamma \vdash S : e$. By IDENTITY, $\Gamma \vdash A \triangleright S : e$. The only interesting condition for the balancing of the target configuration is condition 1, which is trivially proved by Lemma 4.

DECRYPTION
$\Gamma \vdash A \triangleright \mathsf{decrypt}\ \{|M'|\}_{\mathsf{key}(K)}\ \mathsf{as}\ \{|M|\}_{\overline{\mathsf{key}(K)}}.S : e \quad \rightarrow$
$\Gamma \vdash A \triangleright S\sigma : e + ?e_C + ?e_R[U]$ where $U = m.g.u.(M,M')$

Let us suppose the decryption to be performed by asymmetric cryptography. (the reasoning for symmetric cryptography is similar.) Let $vars(M) = \overline{x}$. By IDENTITY and DECRYPT, $A;\Gamma,\overline{x} : \overline{T} \vdash S : e + ?e_C + ?e_R$, where $\Gamma,\overline{x} : \overline{T} \vdash M^{-1} : \mathsf{Enc}(e_C;e_R)$. Types are assigned according to function *Typeofenc* in Table 5. We show that Lemma 2 (the Substitution Lemma) can be applied in the case of ciphertexts originated in CC challenges. (the reasoning for ciphertexts originated in other kinds of nonce handshakes is similar.)

By CIPHERTEXT, $M' = R_{\mathsf{CC?}}(N), \ldots$. We distinguish two cases, depending on whether the encryption key is public or private:

- $\mathsf{Key} = \mathsf{Pub}$ and $\Gamma \vdash \mathsf{Pub}(K) : \mathsf{PublicKey}(A)$ All the free variables in M are typed by Un, but the one tagged by $R_{\mathsf{CC?}}$ which is typed by $\mathsf{Nonce}^{Priv}(I,A)$. By DECRYPT, the received ciphertext has type Un. The judgment can be proved either by CIPHERTEXT (via SUBSUMPTION) (and N has type $\mathsf{Nonce}^{Priv}(I,A)$) or by UN ASYMM CIPH as, by SUBSUMPTION (Public Key), the encryption key is untrusted, i.e., $\Gamma \vdash \mathsf{Pub}(K) : \mathsf{Un}$. In this case, $\Gamma \vdash N : \mathsf{Un}$ and, by the subtyping rule (Tainted Name), we get $\Gamma \vdash N : \mathsf{Nonce}^{Priv}(I,A)$. All the other terms in the received ciphertext, once untagged, have type Un as well as the corresponding variables in M.
- $\mathsf{Key} = \mathsf{Priv}$ and $\Gamma \vdash \mathsf{Priv}(K) : \mathsf{PrivateKey}(I)$ The free variables in M are typed by Un, but the one tagged by $R_{\mathsf{CC?}}$ which is typed by $\mathsf{Nonce}^{Int}(I,A)$. By DECRYPT, the received ciphertext has type Un. If $I \notin \mathcal{ID}_{\mathcal{E}}$, then the judgment is proved by CIPHERTEXT (via SUBSUMPTION) and, according to this rule, N has type $\mathsf{Nonce}^{Int}(I,A)$. If $I \in \mathcal{ID}_{\mathcal{E}}$, then the judgement can be proved even by UN ASYMM CIPH and N might have type Un: by SUBSUMPTION (Public Names), if $I \in \mathcal{ID}_{\mathcal{E}}$, then $\mathsf{Un} <: \mathsf{Nonce}^{Int}(I,A)$. All the other terms in the received ciphertext, once untagged, have type Un as well as the corresponding variables in M.

Hence Lemma 2 can be applied and $\Gamma \vdash A \triangleright S\sigma : e + ?e_C + ?e_R\sigma$.

The application of σ to S does not break the balancing condition 3 as every ground term substituted to variables in S appears in α. We need to prove that adding challenge and response effects keeps balanced the target configuration.

$e_C \neq []$ We need to check that condition 9 still holds on the target configuration. Let us suppose by contradiction that it does not. Since the source configuration is balanced, $\{|M'|\}_{\mathsf{key}(K)} \notin terms(s)$. Since condition 3 holds on the source configuration, $\{|M'|\}_{\mathsf{key}(K)} \in terms(s)$, thus giving a contradiction.

$e_R \neq []$ We need to check that condition 6 and 8 hold on the target configuration. Let us suppose by contradiction that condition 8 does not hold. (the reasoning for condition 6 is the same.) Thus $begin_N(I,J;M_R) \notin s$, with $e_R = \mathsf{Resp}_N^{Ciph}(I,J,M_R)$. By hypothesis, condition 3 holds on the source configuration. Hence $\{|M'|\}_{\mathsf{key}(K)} \in terms(s)$. Since $\Gamma \vdash \{|M'|\}_{\mathsf{key}(K)} : \mathsf{Enc}(\cdot; e_R)$, by condition 8, $begin_N(I,J;M_R) \in s$, thus giving a contradiction.

ENCRYPTION $\boxed{\Gamma \vdash A \triangleright \mathsf{encrypt}\, M \;\mathsf{as}\; z.S : e + !e_R \quad \rightarrow \quad \Gamma \vdash A \triangleright S[M/z] : e + !e_C}$

The judgement $\Gamma \vdash A \triangleright S[M/z] : e + !e_C$, where $\Gamma \vdash M : \mathsf{Enc}(e_C; e_R)$, is proved applying the subtyping rule (Trusted Ciphertext), Lemma 2 (the Substitution Lemma) and IDENTITY. The only interesting point is the balancing of the target configuration:

$e_C \neq []$ We need to check that condition 9 still holds on the target configuration. Let us suppose that $e_C = \mathsf{Chal}_N^{\ell}(A,I,M_C)$ and $\Gamma(N) = \mathsf{Nonce}^{\ell}(A,I)$. The proof for the first implication is trivial and follows from the balancing of the source configuration. Since $\Gamma \vdash M : \mathsf{Enc}(e_C; e_R)$, $M \in terms(s :: encrypt\ M)$ and $!e_C$ belongs to the effect in the target configuration, the second one holds as well.

$e_R \neq []$ We need to check that conditions 6 and 8 still hold on the target configuration. Let us suppose by contradiction that condition 8 does not hold: then $begin(A,I;M_R) \notin s$ with $e_R = \mathsf{Resp}_N^{Ciph}(A,I,M_R)$. By hypothesis, condition 8 holds on the source configuration. Since $!e_R$ belongs to the effect in the source configuration, $begin(A,I;M_R) \in s$, giving a contradiction. The proof for condition 6 is similar.

CAST $\boxed{\Gamma \vdash A \triangleright \mathsf{cast}\, N \;\mathsf{is}\; (x : \mathsf{Un}).S : e + [!\mathsf{Resp}_N^{Un}()] \quad \rightarrow \quad \Gamma \vdash A \triangleright S[N/x] : e}$

We distinguish two cases: if $\Gamma \vdash N : \mathsf{Un}$ (and $\Gamma \vdash N : \mathsf{Nonce}^{Ciph}(I,A)$ by SUBSUMPTION (Tainted Nonce)), then the result follows from IDENTITY and Lemma 2 (the Substitution Lemma). If $\Gamma(N) = \mathsf{Nonce}^{Ciph}(I,A)$, then the result follows applying IDENTITY, Lemma 3 and Lemma 2.

END $\boxed{\begin{array}{l}\Gamma \vdash A \triangleright \mathsf{end}_n(A,I,M_1;M_2).S : e + e' \quad \rightarrow \quad \Gamma \vdash A \triangleright S : e \\ \qquad\qquad\qquad\qquad\qquad\qquad e' = [\mathsf{fresh}^{\ell}(n), !\mathsf{Chal}_n^{\ell}(A,I,M_1), ?\mathsf{Resp}_n^{\overline{\ell}}(I,A,M_2)]\end{array}}$

By IDENTITY and END, $A;\Gamma \vdash \mathsf{end}_n(A,I,M_1;M_2).S : e+e'.$ and $A;\Gamma \vdash S : e$. The result follows from IDENTITY. We need to check that the target configuration is still balanced.

Condition 2 Trivial, by the balancing of the source configuration and an inspection of the target one.

Condition 9 The second implication might fail if $\mathsf{fresh}^{\ell}(n) \in e$. However, by condition 2, $\mathsf{fresh}^{\ell}(n) \notin e$.

The inductive step regards the parallel composition of processes:

PAR $\boxed{\Gamma \vdash P_1|P_2 : e_1+e_2 \quad \rightarrow \quad \Gamma' \vdash P_1'|P_2 : e_1'+e_2}$

By PAR, $\Gamma \vdash P_i : e_i$, with $i \in [1,2]$. Let us suppose $\langle s, P_1\rangle \rightarrow \langle s :: \alpha, P_1'\rangle$. (the symmetric case is analogous.) By inductive hypothesis, we can find an environment Γ' and an effect e_1' such that $\Gamma' \vdash P_1' : e_1'$.

If $\Gamma' = \Gamma, \mathit{names}(\alpha) \setminus \mathit{names}(s) : T$, then, by Lemma 1 and PAR, $\Gamma' \vdash P_1'|P_2 : e_1' + e_2$. If $\Gamma' = \Gamma[n : \mathsf{Un}/n : \mathsf{Nonce}^{Ciph}(I,J)]$, then, by Lemma 3 and PAR, $\Gamma' \vdash P_1'|P_2 : e_1' + e_2$.

We need to prove that the target configuration is balanced. By hypothesis the source configuration is balanced and, by condition 4, s, P_i, Γ, e_i are balanced, $\forall i \in [1,2]$. By induction hypothesis, even $s :: \alpha, P_1', \Gamma', e_1'$ are balanced. It remains to prove that $s :: \alpha, P_1'|P_2', \Gamma', e_1' + e_2$ are balanced.

Condition 1 Since $\Gamma', s :: \alpha, P_1', e_1'$ are balanced, $s :: \alpha \prec \Gamma'$ as desired.

Condition 2 We just discuss the two interesting cases, i.e., $\alpha = \mathit{fresh}(n)$ and $\alpha = \mathit{end}_n(\cdot)$.

- $\alpha = \mathit{fresh}(n)$ By the semantic rule RES, $n \notin \mathit{names}(s)$ and, by condition 1, $n \notin \mathit{dom}(\Gamma)$. Since $\Gamma \vdash e$, $n \notin \mathit{names}(e_1) \cup \mathit{names}(e_2)$ and the inequality trivially holds.
- $\alpha = \mathit{end}_n(\cdot)$ By the typing rule END, $\mathsf{fresh}^{\ell}(n) \in e_1$. By condition 2, we get $\mathit{end}_n(\cdot) \notin s$ and $\mathsf{fresh}^{\ell}(n) \notin e_2$.
  By induction hypothesis $s :: \mathit{end}_n(\cdot), P_1', \Gamma', e_1'$ are balanced: by condition 2 and the typing rule END, $\mathsf{fresh}^{\ell}(n) \notin e_1'$.
  Hence $|\mathsf{fresh}^{\ell}(n)|_{e_1'+e_2} = 0$ and $|\mathit{end}_n(\cdot)|_{s::\mathit{end}_n(\cdot)} = 1$, getting $0+1 \leq 1$ as desired.

Condition 3, 6, 7 and 8 Trivial, by the balancing of $s, P_1|P_2, \Gamma, e_1+e_2$, condition 4 and induction hypothesis.

Condition 4 Trivial, by the balancing of the source configuration and by induction hypothesis.

Condition 9 The first implication is trivially proved by the balancing of the source configuration and induction hypothesis. Let us suppose by contradiction that the second implication does not hold, i.e., $\mathsf{fresh}^{\ell}(n) \in e_1' +$

e_2 and $\exists M \in terms(s :: \alpha)$ such that $\Gamma \vdash M : \mathsf{Enc}(\mathsf{Chal}_n^{\ell}(I,J,M_1);\cdot)$ and $!\mathsf{Chal}_n^{\ell}(I,J,M_1) \notin e_1' + e_2$. The only interesting case is when $\alpha = end_n(I,J,M_1;M_2)$ with $\mathsf{fresh}^{\ell}(n) \in e_2$ as the atomic effect $!\mathsf{Chal}_n^{\ell}(I,J,M_1)$ is removed by the typing rule END, i.e., $e_1 = e_1' + [\mathsf{fresh}^{\ell}(n), !\mathsf{Chal}_n^{\ell}(I,J,M_1), ?\mathsf{Resp}_n^{\overline{\ell}}(J,I,M_2)]$. Since the source configuration is balanced and $\mathsf{fresh}^{\ell}(n)$ belongs to e_1, by condition 2, $\mathsf{fresh}^{\ell}(n) \notin e_1' + e_2$, giving a contradiction.

B.2 ρ-Spi Safety

In the following, we formalize the concept of proof-tree for process judgements. Intuitively, a proof-tree for $\Gamma \vdash P : e$ has the form

$$\dfrac{\dfrac{\dfrac{\Gamma_m \vdash \diamond}{I;\Gamma_m \vdash \mathbf{0} : []}\ \textsc{Nil}}{\vdots}}{\dfrac{\Gamma_1 \vdash P_1 : e_1 \qquad \text{side conditions}}{\Gamma \vdash P : e}\ \textsc{Rule}}$$

Each leaf is labeled by NIL as it is the only rule whose hypothesis does not contain any process judgement. Notice that branching in the proof tree may happen because of PAR, whose hypothesis is composed of two process judgements. More formally, we define a proof-tree τ according to the following grammar:

$$\begin{array}{lcl} \tau & = & \rho.\tau \mid (\tau_1|\tau_2) \\ \rho & = & (\textsc{Rule},\Gamma \vdash P : e) \mid (\textsc{Rule},I;\Gamma \vdash S : e) \end{array}$$

We denote a path λ in τ from a node ρ_1 to a descendant ρ_n by $\rho_1 \rightarrow \ldots \rightarrow \rho_n$ or by $\rho_1 \rightarrow^* \rho_n$ when we are not interested in showing the intermediate nodes.

Let us consider the path $\lambda = \rho_1 \rightarrow^* \rho_i \rightarrow \rho_{i+1} \rightarrow^* \rho_n$, with $\rho_i \in \{(\textsc{Rule}_i,\Gamma_i \vdash P_i : e_i),(\textsc{Rule}_i,I;\Gamma_i \vdash S_i : e_i)\}$ and $\rho_{i+1} \in \{(\textsc{Rule}_{i+1},\Gamma_{i+1} \vdash P_{i+1} : e_{i+1}),(\textsc{Rule}_{i+1},I;\Gamma_{i+1} \vdash S_{i+1} : e_{i+1})\}$.

We say that the atomic effect t is inserted along λ if $t \notin e_i$ and $t \in e_{i+1}$. Similarly, we say that t is removed along the path λ if $t \in e_i$ and $t \notin e_{i+1}$. Analogously, we say that the name n is inserted into the typing environment along λ if $n \notin dom(\Gamma_i)$ and $n \in dom(\Gamma_{i+1})$.

Lemma 6 (Proof Trees and Effects) *Let τ be a proof tree for $\Gamma \vdash P : e$ and λ be a path in τ leading from the root to a leaf. Then every atomic effect in e is removed along the path λ.*

Proof. Trivial as the null process is type-checked only under empty effect. ∎

The following lemma says that the domain of the typing environment monotonously increases along a path in a proof-tree.

Lemma 7 (Environment Monotonicity) *Let λ be the path $(\textsc{Rule},\Gamma \vdash P : e) \rightarrow^* \rho'$, with $\rho' \in \{(\textsc{Rule'},\Gamma' \vdash P' : e'),(\textsc{Rule'},I;\Gamma' \vdash S : e')\}$, in the proof-tree τ. Then $dom(\Gamma) \subseteq dom(\Gamma')$.*

Proof. Trivial by an inspection of the typing rules in Table 7. ∎

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

Lemma 8 (Uniqueness of Challenges) *Let P be a process, Γ a typing environment and e an effect s.t.*

1. $\Gamma \vdash P : e$
2. $e = e' + [!\mathsf{Chal}^{\ell}_{N}(I,J,M)]$
3. $\Gamma \vdash e$
4. $e = e'' + |\mathsf{fresh}^{\ell}(N)| \Rightarrow \mathsf{fresh}^{\ell'}(N) \notin e$.

Then $\nexists \ell'$ s.t. $!\mathsf{Chal}^{\ell'}_{N}(\cdot) \in e'$.

Proof. Let τ be the proof tree for $\Gamma \vdash P : e$ and λ be a path leading from the root to a leaf. By Lemma 6, the atomic effect $!\mathsf{Chal}^{\ell}_{N}(\cdot)$ is removed along λ. The only typing rule achieving this task is END, which removes $\mathsf{fresh}^{\ell}(N)$ as well.

Let us suppose by contradiction that $\exists \ell'$ such that $!\mathsf{Chal}^{\ell'}_{N}(\cdot) \in e'$. Thus either the atomic effects $\mathsf{fresh}^{\ell}(N)$ and $\mathsf{fresh}^{\ell'}(N)$ are in e, or at least one of them is inserted in λ by RES. The former case is not possible because of condition 4 in the hypothesis. Even the latter case gives rise to a contradiction as typing $\mathsf{new}(n:T)$ requires that N is not in the domain of the typing environment and, by Lemma 7 , $N \notin dom(\Gamma)$. Since $\Gamma \vdash e$, $N \in dom(\Gamma)$ giving a contradiction. ∎

The Safety Theorem exploits the preservation of the balancing conditions under process evaluation for proving the safety of well-typed processes.

Theorem 1 (Safety) Let P be a process. If $\overline{I} : \mathsf{Un} \vdash P : []$, where $\overline{I}$ are the identities in 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 ε is safe.

Let $\langle \varepsilon, P\rangle \rightarrow^{*} \langle s, I \triangleright \mathsf{end}_n(I,J,M_1;M_2).S|Q\rangle \rightarrow \langle s :: end_n(I,J,M_1;M_2), I \triangleright S|Q\rangle$ be a semantic derivation leading to an *end* assertion.

The proof is divided in two steps. First, we prove that $begin_n(J,I,M_1;M_2) \in s$, thus showing that every $end_n(I,J,M_1;M_2)$ in s is preceded by a $begin_n(J,I,M_1;M_2)$ (Non-Injective Agreement). Then, we prove that s does not contain actions having the form $end_n(\cdot)$: this implies that every $end_n(I,J,M_1;M_2)$ in s is preceded by a *distinct* $begin_n(J,I,M_1;M_2)$ (Agreement).

Non-Injective Agreement By Theorem 3, $\exists \Gamma, e$ such that $\Gamma \vdash I \triangleright \mathsf{end}_n(I,J,M_1;M_2).S|Q : e$. By an inspection of the typing rules, that judgement is proved by PAR and END. Hence, $\mathsf{fresh}^{\ell}(n) \in e, !\mathsf{Chal}^{\ell}_{n}(I,J,M_1) \in e$ and $?\mathsf{Resp}^{\overline{\ell}}_{n}(J,I,M_2) \in e$. We proceed by cases according to ℓ: each of these cases correspond to a kind of nonce handshake.

$\ell = Un$ (PC handshakes) By syntactic restriction, $M_1 = ()$. By the balancing condition 8, we get $begin_n(J,I;M_2) \in s$, as desired.

$\ell = Ciph$**,** (CP handshakes) By syntactic restriction, $M_2 = ()$. Since $?\mathsf{Resp}_n^{Un}(J,I) \in e$, by the balancing condition 5, $\Gamma \vdash n : \mathsf{Un}$. Since $\mathsf{fresh}^{Ciph}(n) \in e$, by the balancing condition 7, $begin_n(J,I,M_1') \in s$ and $\exists M \in terms(s)$ such that $\Gamma' \vdash M : \mathsf{Enc}([\mathsf{Chal}_n^{Ciph}(I,J,M_1')];\cdot)$. By condition 9, $!\mathsf{Chal}_n^{Ciph}(I,J,M_1') \in e$ and by Lemma 8, $M_1 = M_1'$. Hence $begin_n(J,I,M_1) \in s$, as desired.

$\ell \in \{Priv, Int\}$**,** (CC handshakes) Since $\mathsf{fresh}^{\ell}(n) \in e$, n has been generated with type $\mathsf{Nonce}^{\ell}(I,J)$ and, by Theorem 3, the type of n does not change, i.e., $\Gamma(n) = \mathsf{Nonce}^{\ell}(I,J)$. By condition 6, $begin_n(J,I,M_1';M_2) \in s$ and $\exists M \in terms(s)$ such that $\Gamma \vdash M : \mathsf{Enc}([\mathsf{Chal}_n^{\ell}(I,J,M_1')];\cdot)$. By condition 9, we get $!\mathsf{Chal}_n^{\ell}(I,J,M_1') \in e$ and, by Lemma 8, $M_1 = M_1'$. Hence we have that $begin_n(J,I,M_1;M_2) \in s$, as desired.

Non-Injective Agreement By condition 2, $end_n(\cdot) \notin s$. ∎

Theorem 2 (Strong Compositionality) Let P be the process $\mathbf{keys}(k_1,\ldots,k_n).(I_1 \triangleright !S_1 | \ldots | I_m \triangleright !S_m)$ and $I_1,\ldots,I_m$ be the identities in P. Then $I_1,\ldots,I_m : \mathsf{Un} \vdash P : []$ if and only if $I_1,\ldots,I_m : \mathsf{Un} \vdash \mathbf{keys}(k_1,\ldots,k_n).I_i \triangleright !S_i : [], \forall i \in [1,m]$.

Proof. Trivial, by an inspection of rule PAR. ∎