

A Calculus of Challenges and Responses

Michael Backes
Saarland University
backes@cs.uni-sb.de

Agostino Cortesi
Ca' Foscari University
cortesi@dsi.unive.it

Riccardo Focardi
Ca' Foscari University
focardi@dsi.unive.it

Matteo Maffei
Saarland University
maffei@cs.uni-sb.de

Abstract

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

1 Introduction

Language-based security has proved to be a salient technique for formally analyzing security protocols, since Abadi's seminal work on secrecy by typing [1] up to modern techniques based on logics, model-checking and type systems (a far from being comprehensive list includes [3, 27, 23, 16, 25, 12, 18, 7]). Authentication protocols are known to require very subtle reasoning about encrypting and decrypting various related messages in the presence of a powerful adversary, with each of these encryptions being meant to contribute some part of the overall authentication guarantee. Thus they are strongly vulnerable to attacks originating by a certain degree of ambiguity in the protocol message, e.g., man-in-the-middle attacks or attacks where certain encryptions are re-used by the adversary in another protocol execution where they suddenly get a different semantics. This turns authentication protocols into particularly suitable targets of language-based security, and several current language-based static analysis techniques [25, 12, 18] for tackling this problem have been proposed, e.g., based on protocols narrations formalized in the spi-calculus [4], or in variants thereof such as Lysa [11] and the ρ -spi calculus [16]. Roughly speaking, these techniques typically rely on some static patterns, defined on the syntax of the calculus, which suffice

for showing that the run-time protocol execution respects certain intended challenge-response schemes which in turn imply the desired authentication guarantees.

For instance, the type system by Gordon and Jeffrey [25] relies on some type information provided by the user, which suffices for identifying nonces and formalizing to which extent their exchange contributes to achieving authentication. The generality of the analysis is sometimes paid by sophisticated type definitions that have to be defined manually and thus require a certain degree of expertise on the part of programmers. The type system by Bugliesi, Focardi and Maffei [18] relies on some dynamic information attached to ciphertexts, in the form of tags, which univocally determine the role of messages in the authentication task. A great advantage is that tag and type definitions are automatically inferred [29]. Furthermore, the use of tags makes the analysis compositional, thus naturally fitting the analysis of multi-protocol systems, where participants engage in different, and possibly unknown, protocols. Even if the analysis is general enough to cover several existing protocols, its scope is strictly constrained by the set of tagged ciphertexts.

This paper tackles the analysis of authentication protocols from a conceptually more abstract perspective. Starting from protocol narrations expressed in a dialect of the ρ -spi calculus, we abstract from the specific structure of messages by solely focussing on the challenge-response components that are inherent in the protocols. The resulting more abstract protocols are formalized in a new process calculus, the CR *calculus*, which is specifically tailored to reasoning about challenge-response mechanisms within authentication protocols. The CR calculus grants a more abstract view of authentication protocols, enables more abstract and general proofs, and enjoys some properties constituting a significant advantage over existing type-based approaches.

- Foremost, we prove a soundness theorem stating that abstractions preserve authentication properties, i.e., proving authentication for a protocol abstraction in the CR calculus suffices for obtaining an authentication proof for the actual ρ -spi protocol.
- The analysis is modular and compositional since each principal is independently validated and the parallel composition of successfully validated protocols yields a secure protocol again. This fits very well the analysis of multi-protocol systems.
- The abstraction from ρ -spi to CR calculus protocol descriptions and the effect system used for verifying the safety of the latter are completely independent. This independence constitutes a key feature of our approach since it entails that refining the abstraction does not affect the soundness of the analysis and vice-versa.
- The abstraction is extensible in that extensions to additional protocol classes immediately enjoy a soundness theorem provided that these extensions satisfy certain explicit, easily checkable conditions. This is a key difference with respect to other type-based approaches [2, 25, 18], where an extension of the analysis requires re-proving the soundness and the safety of the analysis from scratch.

Further related work. Strand spaces [26, 27, 21] and ProVerif [8, 3, 9] constitute effective and general frameworks for the analysis of security protocols: the analysis is automated, applies to several protocols and deals with different security properties.

In contrast to dynamic typing, these techniques prove safety results against participants running the protocol of interest. However, as discussed in [30], problems may arise when participants execute different protocols with the same cryptographic keys. The interaction among *different* protocols, possibly unknown or, by contrast, carrying out some common sub-tasks, is particularly interesting when several security services coexist and are possibly combined together. Interestingly, strand spaces offer some syntactic conditions on protocol specifications for guaranteeing the compositionality of the analysis. This still assumes that protocols interacting with each other are known and requires some constraints on the form of protocols, while our analysis guarantees that the parallel composition of successfully validated protocols is still safe, possibly relying on message tags. As opposed to our approach, both ProVerif and Strand Spaces do not enjoy guaranteed termination, although the analysis terminates for a large class of protocols [10].

The control-flow analysis for message authenticity proposed by Bodei *et al.* in [12] and recently extended in [24] to detect replay attacks is closely related to our approach: both of the techniques enjoy guaranteed termination but, due to the undecidability of authenticity, they perform an overapproximation that necessarily rules out safe protocols. The technique proposed in this paper exploits some syntactic patterns that suffice to guarantee the safety of the protocol and, notably, to ensure the compositionality of the analysis. The control flow analysis of [12, 24] works on an abstraction of the protocol semantics and, consequently, it is not constrained by the challenge-response paradigm. However, it does not enjoy immediate compositionality results.

Finally, Datta *et al.* [23, 20, 19] have recently proposed the protocol composition logic (PCL), an interesting logic-based approach to cryptographic protocol analysis. PCL is designed around a process calculus with actions for possible protocol steps including generating new random numbers, sending and receiving messages, and performing decryption and digital signature verification actions, much in the same style as ρ -spi calculus. By relying on protocol invariants, authors develop a modular way of reasoning about security protocols, ensuring that protocols that are proved to be individually secure do not interact insecurely when they are composed with other protocols. As opposed to our approach and similarly to [26], compositionality results require the knowledge of the protocols that are concurrently executed. However, PCL supports compositional reasoning about sequential composition of protocol steps, which we do not address in this paper. A formal comparison is thus interesting but, since the analysis technique and the underlying model are different, it is left as future work.

Outline. Section 2 reviews the ρ -spi calculus and introduces a small novel dialect thereof. Section 3 presents the new CR calculus. Section 4 introduces the abstraction of ρ -spi protocol descriptions into the CR calculus. Section 5 proposes an effect system for checking the safety of CR protocols. Section 6 concludes and outlines future work.

Note for reviewers. A preliminary version of this work has been presented at [6], a workshop with no formally published proceedings thus not precluding further conference publication. With respect to that version, we have extended the calculus and the abstraction so as to deal with hash functions, MACs and session key distribution and authentication.

2 Review of the ρ -spi Calculus

The ρ -spi calculus [16] derives from the spi calculus [4] and inherits many of the features of *Lysa* [11], a dialect of the spi calculus specifically tailored to the analysis

Table 1 (Our Dialect of) the ρ -spi Calculus

Names		Terms	
$a ::=$	n, m (Msg) I, J, A, B, E (Id)	$T ::=$	a (Name) k (Keys) x, y, z (Vars) $?x, ?y, ?z$ (Input Vars)
Keys		$\text{Tag}(T)$ (Tag) (T, T) (Pair) $\{\!\{ T \}\!\}_K$ (Enc) $h(T)$ (Hash) $\text{MAC}_K(T)$ (MAC)	
$k ::=$	k_{IJ} (Sym) k_I^+, k_I^- (Asym)		
Processes			
$P, Q ::=$	$\text{new}(n).P$ (New Name) $\text{new}(k_{IJ}).P$ (Ses. Key) $\text{in}(T).P$ (In) $\text{out}(T).P$ (Out) $\text{begin}_N(A, I, M, M, K).P$ (Begin) $\text{end}_N(A, I, M, M, K).P$ (End) $A \triangleright P$ (Princ) $P Q$ (Par) $!P$ (Repl) $\mathbf{0}$ (Stop)		

– M, N denote terms without encryptions and tags. K denotes keys and key variables x_{IJ} .

of authentication protocols. The ρ -spi calculus differs from both calculi in several respects: it incorporates the notion of tagged message exchange from [15], associates principal identities to processes and syntactically binds keys to their owners. In this paper, we consider a novel dialect of ρ -spi in which encryptions and decryptions are performed on-the-fly when sending and receiving messages, respectively. This dialect in particular links protocol specifications more tightly to their informal “graphical” descriptions, which only depict sent and received messages without giving a precise semantics on how messages are parsed and constructed. Furthermore, the calculus is extended so to also deal with hash functions, message authentication codes (MAC) and session keys.

2.1 Syntax of our Dialect of the ρ -spi Calculus

The formal syntax of our dialect of the ρ -spi calculus is depicted in Table 1. We presuppose a countable set \mathcal{N} of *names* partitioned into two distinct categories: *messages* and *identities*. The set of identities ID , ranged over by I and J , is further partitioned into *trusted principals* $ID_{\mathcal{P}}$, ranged over by A and B , and *enemies* $ID_{\mathcal{E}}$, ranged over by E . *Keys* are partitioned into symmetric long-term keys k_{IJ} , shared between I and J , and asymmetric long-term keys k_I^+, k_I^- representing corresponding public and private keys belonging to I , and symmetric session keys k_{IJ} , shared between I and J . Long term keys are assumed to be known in advance by the respective users and, for this reason, occur free in the process, i.e., they are not explicitly generated through the command ‘new’; session keys, instead, are always freshly generated through the command ‘new(k_{IJ})’.

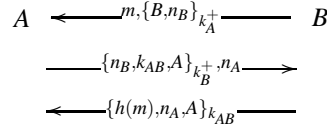
Well-formed processes (see below) are such that long-term keys are never transmitted on the network. For easing the presentation of the analysis, we isolate a subset of variables, called *key variables*: these variables represent session keys received from the network and are labelled by the identity of the owners: for instance, x_{IJ} denotes a variable that at run-time should be instantiated with a session key shared between I and J . Notice that such an ideal behavior is not checked by the semantics and, in fact, key variables have the same computational import as variables and can be thus instantiated with any term. However, we prove that for safe protocols, i.e., protocols successfully validated, key variables are only replaced at run-time by the intended session-keys. We also presuppose a set \mathcal{T} of tags and terms can be tagged using a $\text{Tag} \in \mathcal{T}$, thus determining their role in the authentication task. Moreover, terms contain pairs¹, encryptions, hashes and MACs. The special name $_$, which denotes the empty message, will be omitted when occurring at the end of a tuple, e.g., $\text{begin}_N(A, B, M_1, M_2, _)$ will be written as $\text{begin}_N(A, B, M_1, M_2)$.

Processes (or *protocols*), ranged over by P and Q , behave as follows: $\text{new}(n).P$ generates a fresh name n local to P and $\text{new}(k_{IJ}).P$ generates a fresh session key k_{IJ} local to P , whose intended scope is the pair of principals I and J . We presuppose a unique anonymous public channel, the network, from/to which all principals, including intruders, read and send messages. Similarly to *Lysa*, our input primitive may atomically test part of the read message, by employing pattern-matching. Notice that in is a binder for the variables preceded by ‘?’, called *input variables*, and we forbid the occurrence of input variables $?x$ anywhere else. The scope of an input variable is the right-hand side of the input pattern and the continuation process. If the input message matches the input pattern, then the input variables are bound to the corresponding sub-part of the input message; otherwise the message is not read at all. For example, process $\text{in}(?x, ?y).P$ reads any pair (G, G') and reduces into process $P[G/x, G'/y]$, where the free occurrences of x and y are replaced by G and G' , respectively; process $\text{in}(?x, x).P$ reads instead only pairs composed of identical messages, since the input variable $?x$ binds the second occurrence of x in the input pattern, which is thus pattern-matched. This mechanism is also used to decrypt received messages on-the-fly, and thus constitutes an important novelty compared to the ρ -spi calculus; of course, in order to immediately match a message encrypted with asymmetric cryptography, the correct decryption key has to be specified in the input pattern. For example, process $\text{in}(\{?x\}_{k_A^-}).P$ reads any message encrypted with A 's public key k_A^+ , i.e., messages of the form $\{G\}_{k_A^+}$, decrypts them on the fly, and binds all the free occurrences of x to G in process P . The semantics is formally defined in Section 2.2. Note that we distinguish between the *static* cryptographic terms T of the calculus, and the actual sent and received *messages* G . Encryption, hashing and MAC generation for terms is denoted $\{T\}_k, h(T)$ and $\text{MAC}_k(T)$, respectively, while encrypted messages, hashes and MACs are denoted $\{G\}_k, h(G)$ and $\text{MAC}_k(G)$, respectively. This is crucial in the calculus semantics to distinguish, e.g., between the simple reception and the reception-with-decryption of an encrypted message. For instance, $\text{in}(?x).P$ and $\text{in}(\{?y\}_{k_{AB}}).Q$ can both read message $\{G\}_{k_{AB}}$ but the first process just reads it, denoted $\text{in}(\{G\}_{k_{AB}})$, while the second one reads and decrypts it, denoted $\text{in}(\{G\}_{k_{AB}})$. In particular, x is bound to $\{G\}_{k_{AB}}$ while y is bound to G . This makes it possible to express protocols in which part of a message is unknown to the recipient, as also done, for example, in symbolic model checking and constraint solving [5, 13, 31].

¹For the sake of readability, in the rest of the paper we write pairs as tuples: for instance, the nested pair $(a, (b, k))$ is simplified in (a, b, k) .

The $\text{begin}_N(A, B, M_1, M_2, K)$ and $\text{end}_N(B, A, M_1, M_2, K)$ primitives are used to check the *correspondence assertions* [32] in a nonce handshake between A and B based on nonce N . The former declares that A is starting a protocol session with B , while the latter declares that B is ending a protocol session in which he believes to have correctly authenticated A . Messages M_1 and M_2 , when specified, represent messages exchanged respectively from B to A and from A to B during the protocol session; K is a session key sent by A to B and authenticated in the handshake. Finally, $A \triangleright P$ represents principal A executing process P ; process $P|Q$ is the parallel composition of P and Q ; process $!P$ indicates an arbitrary number of parallel instances of P , and $\mathbf{0}$ is the null process that does nothing. We always omit $\mathbf{0}$ from protocol specifications. Finally, we remark that the ρ -spi calculus comes with a notion of well-formedness checking that (i) identity declarations do not nest; (ii) the first identity in begin and end assertions refers to the principal running the process; (iii) principals only use their own long-term keys; (iv) session keys are the only keys that can be sent or received on the network; and (v) each key variable x_{IJ} is authenticated before being used for encryption, i.e., each input and output of terms of the form $\{T\}_{x_{IJ}}$ is preceded by an end assertion of the form $\text{end}_{n_i}(J, I, M_1, M_2, x_{IJ})$. This last condition requires the authentication of session keys before their actual use in the protocol. Well-formedness is trivially verifiable by a syntactic inspection of processes and, from now on, we will implicitly assume it.

Example 1 To illustrate, let us consider the following mutual authentication protocol, where B is willing to authenticate message m with A , through a session-key freshly generated by A .



In the first message, B encrypts a fresh nonce n_B and his own identifier with A 's public-key, while the message to authenticate is sent in clear. Since the attacker can manipulate messages in transit on the network, A cannot check the origin or the integrity of message m . A proves her identity by decrypting the ciphertext and replies by sending to B a fresh nonce n_A and another ciphertext, encrypted with B 's public-key and containing n_B , a fresh session-key k_{AB} and A 's identifier. After receiving the second message and checking the freshness of n_B , B authenticates A and k_{AB} . In the third message, B 's encrypts the hash of m , n_A and A 's identifier with k_{AB} : after decrypting the ciphertext, A authenticates B and message m . The protocol description in ρ -spi calculus is reported in Table 2. For the sake of readability, we have depicted message exchanges in between the corresponding inputs and outputs. Notice that B authenticates the session key received from A in the second message ($\text{end}_{n_B}(B, A, -, -, x)$) and, before sending the third message, he declares his intention to authenticate m with A ($\text{begin}_y(B, A, -_m)$). Dually, A starts the protocol to exchange an authenticated session key k_{AB} with B ($\text{begin}_y(A, B, -, -, k_{AB})$) and finally authenticates the message received from B ($\text{end}_{n_A}(A, B, -, -, x)$). \square

2.2 Operational Semantics of our Dialect of the ρ -spi Calculus

We define the operational semantics of our dialect of ρ -spi in terms of *traces*, following [13]. 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

Table 2 Example protocol in ρ -spi

$Resp \triangleq$		$Init \triangleq$
$in(?x, \{B, ?y\}_{k_A^-}).$ $new(n_A).new(k_{AB}).$ $begin_y(A, B, -, -, k_{AB}).$ $out(\{y, k_{AB}, A\}_{k_B^+}, n_A).$	$\longleftarrow m, \{B, n_B\}_{k_A^+} \longrightarrow$	$new(m).new(n_B).$ $out(m, \{B, n_B\}_{k_A^+}).$
$in(\{h(x), n_A, A\}_{k_{AB}}).$ $end_{n_A}(A, B, -, x).$	$\longleftarrow \{n_B, k_{AB}, A\}_{k_B^+}, n_A \longrightarrow$ $\longleftarrow \{h(m), n_A, A\}_{k_{AB}} \longrightarrow$	$in(\{n_B, ?x_{AB}, A\}_{k_B^-}, ?y).$ $end_{n_B}(B, A, -, -, x_{AB}).$ $begin_y(B, A, -, m).$ $out(\{h(m), y, A\}_{x_{AB}}).$
$System \triangleq !B \triangleright Init \mid !A \triangleright Resp$		

between *configurations*, i.e., pairs $\langle s, P \rangle$, where $s \in Act^+$ is a trace and P is a closed process. Each transition $\langle s, P \rangle \rightarrow \langle s :: \alpha, P' \rangle$ simulates one computation step in P and records the corresponding action α in the trace. We denote by \rightarrow^+ a finite non-empty sequence of computation steps. Principals do not directly synchronize with each other. Instead, they may receive from the unique channel an arbitrary message known to the environment, which models the Dolev-Yao intruder [22]: the knowledge of the environment is formalized by a set of deduction rules stating that the environment knows all the messages sent on the network, every name which is not restricted in the trace, all public keys, and every shared and private key with E in the subscript, e.g., k_{EI} and k_E^- . Moreover, the environment can tag and untag messages, construct or destruct pairs, apply hash functions, and, if the appropriate key is known, encrypt/decrypt messages and build MACs. The transition relation and the deductive system for the knowledge of the environment are formalized in Appendix A.

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

The notion of safety extends the *agreement* property of [32, 28] by distinguishing between received and sent messages and pointing out the nonce used in the handshake.

Definition 2 (Safety) *A trace s is safe if and only if whenever $s = s_1 :: end_n(A, B, G) :: s_2$, there exist s'_1, s''_1 such that $s_1 = s'_1 :: begin_n(B, A, G) :: s''_1$ and $s'_1 :: s''_1 :: s_2$ is safe. P is safe if s is safe for all $s \in Tr(P)$.*

A trace is safe if every $end_n(A, B, G)$ is preceded by a distinct $begin_n(B, A, G)$. Intuitively, this guarantees that whenever B authenticates A then A is willing to authenticate with B and the two principals agree on the authenticated term G^2 .

3 CR Calculus

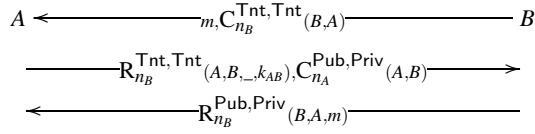
In the previous section, we presented a calculus for specifying and reasoning on cryptographic authentication protocols. Now, we want to abstract away from the specific structure of messages, and in particular from symbolic cryptography, and to reason on authentication protocols just in terms of challenges and responses. To illustrate,

²For convenience, we often write the triple (G_1, G_2, G_3) of authenticated messages as a unique term G .

Table 3 Protocol concretizations

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

let us consider again the protocol of Example 1. Intuitively, the first message is a “tainted” challenge, since A ’s public key is known to the attacker. The second message is composed of a tainted response and a public challenge directed to B . Finally, the third message is a private response, since it is encrypted with a symmetric session-key shared between A and B . We can depict the abstract protocol narration as follows:



The idea is that cryptographic challenges or responses are abstracted into two special messages, namely $C_N^{\ell, \ell'}(B, A, M_1, K_1)$ and $R_N^{\ell, \ell'}(A, B, M_2, K_2)$. The former may be read as “challenge sent by B to A for authenticating message M_1 and session-key K_1 with nonce N ” and the latter may be read as “response sent by A to B for authenticating message M_2 and session-key K_2 with nonce N ”. Labels $\ell, \ell' \in \{\text{Pub}, \text{Tnt}, \text{Int}, \text{Priv}\}$ specify the security level of the challenge and the response, respectively, with the following meaning:

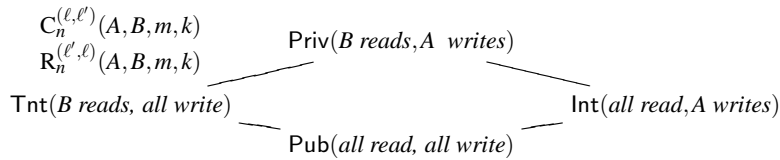
Pub : readable and writable by everyone; cleartexts fall in this class;

Tnt : writable by everyone but readable only by the intended receiver; public key cryptography is an example of this kind;

Int : readable by everyone but writable only by the sender (e.g., digital signatures);

Priv : readable only by the intended receiver and writable only by the sender; this is achievable through, e.g., “authenticated symmetric encryption” as soon as the “direction” is made explicit as previously explained.

The capability of reading or writing challenges and responses induces a partial order \leq on labels:



Messages with $\ell \leq \text{Tnt}$ may be written by everyone, while messages with $\ell \leq \text{Int}$ may be read by everyone. Moreover, only B can read messages with $\ell \geq \text{Tnt}$ and only A can write messages with $\ell \geq \text{Int}$. There are two ways in which a principal B can authenticate himself: (i) by sending a response that only B can generate, i.e., with security level Int or Priv ; and (ii) by replying to a challenge that only B can read, i.e., Tnt or Priv . For example, $R_n^{\text{Pub}, \text{Int}}(A, B, m)$ represents an integer response to a public

Table 4 CR Calculus Names and Terms

CR Names		CR Terms	
$a ::=$	n, m (Msgs)	$T ::=$	a (Names)
	k_{IJ} (Ses Keys)		x, y, z (Vars)
	I, J, A, B, E (Identities)		$?x, ?y, ?z$ (Input Vars)
	\top (Any)		(T_1, T_2) (Pair)
Notation :	$\ell \in \{\text{Pub}, \text{Tnt}, \text{Priv}, \text{Int}\}$.		$C_N^{(\ell, \ell')}(I, J, M, K)$ (Chal)
	M, N denote terms with no challenges and responses.		$R_N^{(\ell, \ell')}(I, J, M, K)$ (Resp)
	K is either a session key k_{IJ} or a session key variable x_{IJ} .		

challenge $C_n^{\text{Pub}, \text{Int}}(B, A)$, which might be concretized as in protocol a (cf. Table 3): A proves her identity by signing the nonce together with ‘ B ’ and message m . As another example, consider $R_n^{\text{Tnt}, \text{Tnt}}(A, B, -, k_{AB})$ representing a tainted response to a tainted challenge $C_n^{\text{Tnt}, \text{Tnt}}(B, A, m)$, which might be concretized as in protocol b: A proves her identity by decrypting the tainted challenge using her private key. Attention should be paid when the security label of the challenge and the response is the same and symmetric key cryptography is used. For example, $R_n^{\text{Priv}, \text{Priv}}(A, B, m_2)$ with challenge $C_n^{\text{Priv}, \text{Priv}}(B, A, m_1)$ should never be concretized using messages that might be confused. For example, consider the worst case in which challenge and response are the same, as in protocol c. Here the enemy can trivially attack the protocol by replaying the received challenge back to B , authenticating as A . This is why the CR calculus explicitly distinguishes between challenges and responses and the abstraction from ρ -spi to the CR calculus guarantees that they remain distinguishable even when concretized through symbolic cryptography (see Section 4).

3.1 Syntax and semantics

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

Example 2 We show the CR calculus specification corresponding to Example 1. As before, we use arrows to point out synchronizing inputs and outputs.

$$\begin{array}{l}
\text{Resp} \triangleq \\
\text{in}(\text{?}x, C_{?y}^{\text{Tnt}, \text{Tnt}}(B, A)) \\
\text{new}(n_A). \text{new}(k_{AB}). \\
\text{begin}_y(A, B, -, -, k_{AB}). \\
\text{out}(R_y^{\text{Tnt}, \text{Tnt}}(A, B, -, k_{AB}), \\
C_{n_A}^{\text{Pub}, \text{Priv}}(A, B)). \\
\text{in}(R_{n_A}^{\text{Pub}, \text{Priv}}(B, A, x)). \\
\text{end}_{n_A}(A, B, -, x). \\
\end{array}
\quad
\begin{array}{l}
\text{Init} \triangleq \\
\text{new}(m). \text{new}(n_B). \\
\text{out}(m, C_{n_B}^{\text{Tnt}, \text{Tnt}}(B, A)). \\
\text{in}(R_{n_B}^{\text{Tnt}, \text{Tnt}}(A, B, -, ?x_{AB}), C_{?y}^{\text{Pub}, \text{Priv}}(A, B)). \\
\text{end}_{n_B}(B, A, -, -, x_{AB}). \\
\text{begin}_y(B, A, -, m). \\
\text{out}(R_y^{\text{Pub}, \text{Priv}}(B, A, m)) \\
\end{array}$$

$$\text{System} \triangleq !B \triangleright \text{Init} \mid !A \triangleright \text{Resp} \quad \square$$

As the ρ -spi calculus, the CR calculus comes with a notion of well-formedness for processes ruling out undesired process behaviors. In particular, as for ρ -spi (i) identity declarations do not nest; (ii) the first identity in begin and end assertions refers to the principal running the process. Moreover, (iii) the identities in challenges and responses refers to the sender and the receiver, respectively; (iv) nonces, i.e., the subscripts of begin and end assertions, can occur in input and output patterns only as subscripts of challenge and response terms and (v) session keys and key variables can occur in input and output patterns only as last component of challenge and response terms. The last two conditions ensure that nonces and session keys always appear in a precise position within exchanged challenges and responses. This is crucial to avoid they are leaked or used in an uncontrolled way. These conditions are easily verifiable by an inspection of the process syntax and in the following we shall always consider well-formed processes.

We define CR calculus operational semantics in terms of *traces*. Recall that principals do not directly synchronize with each other. Instead, they may receive from the unique channel an arbitrary message known to the environment, which models the Dolev-Yao intruder. Here, instead of symbolic representations of cryptographic terms, the intruder can manipulate challenges and responses but we assume that security labels are respected: the intruder can forge messages with labels less than or equal to Tnt ; moreover, the intruder can read messages with labels less than or equal to Int . Finally, if the attacker gets the knowledge of a session-key k_{IJ} , then it can also write and read challenges and responses sent by I to J or vice-versa, regardless of their security level, thus abstracting session-key corruption. In the following, we write \rightarrow_a and \rightarrow_a^+ to denote one-step and multi-step reduction of abstract processes, respectively. For more detail on the transition relation and the deductive system for the knowledge of the environment, we refer to Appendix B.

4 Abstraction from ρ -spi to CR

The abstraction from ρ -spi to CR calculus is defined on ρ -spi terms, traces and processes. The idea is to abstract away from the specific structure of messages, focusing instead on their challenge-response role in the authentication task. The abstraction, given a ρ -spi protocol narration, yields a CR protocol that is proved to be a faithful abstraction of the former. The abstraction of terms, reported in Table 5, allows for abstracting ciphertexts into challenges and responses: the abstraction of traces and processes is conceptually simpler and amounts to abstract every term occurring therein.

The abstraction of names and variables is straightforward and amounts to map them into their direct CR counterparts. The abstraction of a pair yields the pair composed of the abstractions of each component. Finally, tags and hashes are abstracted away. The abstraction of encryptions performed by trusted principals is parameterized and ruled by a partial function f from encryption patterns to abstract challenge and response terms. This function works on syntactic terms in which there are no input variables and all the variables are distinct. In the actual protocol specification variables may be different, preceded by ‘?’ or even replaced by run-time terms: the abstraction of an encryption is thus given by the closure of f , denoted by \underline{f} .

Function \underline{f} is defined by closing f under all the possible *valid* variable instantiations: a valid variable instantiation on syntactic terms maps (i) all the key variables x_{IJ} into different key variables y_{IJ} (possibly prefixed by ‘?’) or actual session keys k_{IJ} of the same pair I, J of principals, and (ii) the remaining concrete variables into different

Table 5 Abstraction of run-time terms, terms and processes

Run Time Terms

$$\begin{aligned}
\alpha(a) &= a \\
\alpha(R_1, R_2) &= (\alpha(R_1), \alpha(R_2)) \\
\alpha(\text{Tag}(R)) &= \alpha(R) \\
\alpha(h(R)) &= \alpha(R) \\
\alpha(\llbracket R \rrbracket_K) &= \begin{cases} \underline{f}(\llbracket R \rrbracket_K) & \text{if } \llbracket R \rrbracket_K \in \text{dom}(\underline{f}) \\ \underline{\alpha}(R) & \text{if } \llbracket R \rrbracket_K \notin \text{dom}(\underline{f}) \wedge \\ & \nexists \text{ valid } \sigma \text{ and } R' \text{ s.t.} \\ & [\llbracket R \rrbracket_K] \sigma = [R'] \wedge R' \in \text{dom}(\underline{f}) \\ \perp & \text{otherwise} \end{cases} \\
\alpha(\text{MAC}_K(\llbracket R \rrbracket)) &= \begin{cases} \underline{f}(\text{MAC}_K(\llbracket R \rrbracket)) & \text{if } \text{MAC}_K(\llbracket R \rrbracket) \in \text{dom}(\underline{f}) \\ \underline{\alpha}(R) & \text{if } \text{MAC}_K(\llbracket R \rrbracket) \notin \text{dom}(\underline{f}) \wedge \\ & \nexists \text{ valid } \sigma \text{ and } R' \text{ s.t.} \\ & [\text{MAC}_K(\llbracket R \rrbracket)] \sigma = [R'] \wedge R' \in \text{dom}(\underline{f}) \\ \perp & \text{otherwise} \end{cases} \\
\alpha(G) &= \top \quad G \text{ in } \{\{G\}_k, h(G), \text{MAC}_k(G)\}
\end{aligned}$$

Processes

$$\begin{aligned}
\alpha(\mathbf{0}) &= \mathbf{0} \\
\alpha(\text{new}(n).P) &= \text{new}(\alpha(n)).\alpha(P) \\
\alpha(\text{in}(\bar{R}).P) &= \text{in}(\alpha(R)).\alpha(P) \\
\alpha(\text{out}(R).P) &= \text{out}(\alpha(R)).\alpha(P) \\
\alpha(\text{begin}_G(A, I, G').P) &= \text{begin}_{\alpha(G)}(\alpha(A), \alpha(I), \alpha(G')).\alpha(P) \\
\alpha(\text{end}_G(A, I, G').P) &= \text{end}_{\alpha(G)}(\alpha(A), \alpha(I), \alpha(G')).\alpha(P) \\
\alpha(A \triangleright P) &= \alpha(A) \triangleright \alpha(P) \\
\alpha(P | Q) &= \alpha(P) | \alpha(Q) \\
\alpha(!P) &= !\alpha(P)
\end{aligned}$$

Traces

$$\begin{aligned}
\alpha(\varepsilon) &= \varepsilon \\
\alpha([A \triangleright] \text{new}(n) :: t) &= \text{new}(\alpha(n)) :: \alpha(t) \\
\alpha(\text{in}(R) :: t) &= \text{in}(\alpha(R)) :: \alpha(t) \\
\alpha(\text{out}(R) :: t) &= \text{out}(\alpha(R)) :: \alpha(t) \\
\alpha(\text{begin}_G(A, I, G') :: t) &= \text{begin}_{\alpha(G)}(\alpha(A), \alpha(I), \alpha(G')) :: \alpha(t) \\
\alpha(\text{end}_G(A, I, G') :: t) &= \text{end}_{\alpha(G)}(\alpha(A), \alpha(I), \alpha(G')) :: \alpha(t) \\
\alpha(A \triangleright t) &= \alpha(A) \triangleright \alpha(t)
\end{aligned}$$

- \bar{R} replaces each decryption key in R with the corresponding encryption one.
- σ is valid if the following conditions hold:
 - (i) $\sigma : x \mapsto a \mid \{G\}_a \mid \text{MAC}_a(G) \mid h(G) \mid ?x$
 - (ii) $x_{IJ} \in \text{dom}(\sigma) \Rightarrow \sigma(x_{IJ}) \in \{?y_{IJ}, y_{IJ}, k_{IJ}, \text{ for some } k, y\}$
- For every $T \in \text{dom}(f)$ and valid σ , $\underline{f}(T\sigma) \stackrel{\text{def}}{=} f(T)\sigma^\sharp$

where σ^\sharp is defined as follows:

$$\begin{aligned}
(i) \quad \sigma^\sharp : x \mapsto \top \\
(ii) \quad x \in \text{dom}(\sigma^\sharp) \Leftrightarrow x \in \text{dom}(\sigma) \\
(iii) \quad \sigma^\sharp(x) = \begin{cases} a & \text{if } \sigma(x) = a \\ y & \text{if } \sigma(x) = y \\ ?y & \text{if } \sigma(x) = ?y \\ \top & \text{otherwise} \end{cases}
\end{aligned}$$

Table 6 Encryption Abstraction

A partial function $f : \llbracket T \rrbracket_K \mid \text{MAC}_K(\llbracket T \rrbracket) \mapsto \mathcal{C}/\mathcal{R}_N^{\ell, \ell'}(I, J, M, K)$ is an *encryption abstraction* iff

Format - $\text{names}(T) \subseteq \mathcal{ID}$ and $\text{names}(N) = \text{names}(M) = \text{names}(K) = \emptyset$;
 - $x \in \text{vars}(T)$ iff $x \in \text{vars}(N) \cup \text{vars}(M) \cup \text{vars}(K)$;
 - $\forall x, y \in \text{vars}(M)$, x precedes y in T iff x precedes y in M .
 - T does not contain input variables

Unique Abstraction if $\exists T', T' \in \text{dom}(f)$, valid σ, σ' s.t. $\llbracket T \rrbracket \sigma = \llbracket T' \rrbracket \sigma'$,
 then $T = T'$;

Nesting if $\exists T' \in \text{dom}(f)$ and $\llbracket T \rrbracket_K \in \text{terms}(T')$ (resp. $\text{MAC}_K(\llbracket T \rrbracket) \in \text{terms}(T')$),
 then $\alpha(\llbracket T \rrbracket_K) = \alpha(T)$ (resp. $\alpha(\text{MAC}_K(\llbracket T \rrbracket)) = \alpha(T)$) (nested terms are neither challenges nor responses);

Encryption If $\llbracket T \rrbracket_K \in \text{dom}(f)$ or $\text{MAC}_K(\llbracket T \rrbracket) \in \text{dom}(f)$ then

- $K = k_T^+$ implies $f(\llbracket T \rrbracket_K) \in \{\mathcal{C}_N^{\ell, \ell'}(J, I, M, K), \mathcal{R}_N^{\ell', \ell}(J, I, M, K)\}$, with $l \leq \text{Tnt}$;
- $K = k_T^-$ implies $f(\llbracket T \rrbracket_K) \in \{\mathcal{C}_N^{\ell, \ell'}(I, J, M, K), \mathcal{R}_N^{\ell', \ell}(I, J, M, K)\}$, with $l \leq \text{Int}$;
- $K \in \{k_{IJ}, x_{IJ}\}$ implies $f(\llbracket T \rrbracket_K) \in \{\mathcal{C}/\mathcal{R}_N^{\ell, \ell'}(I, J, M, K), \mathcal{C}/\mathcal{R}_N^{\ell', \ell}(J, I, M, K)\}$;

variables or run-time terms different from pairs and tagged terms that are never substituted to variables at run-time. Function \underline{f} maps the abstract variables correspondingly apart from non-atomic terms which are abstracted into \top . With non-atomic terms we mean terms that are not names, keys, variables and input variables

If the encryption is in the domain of \underline{f} , then the abstraction is defined accordingly. If the encryption is not defined in \underline{f} and there is no variable instantiation allowing the corresponding run-time ciphertext to be interpreted as a challenge-response term, then the abstraction is simply defined as the abstraction of the content in that the outer encryption does not provide any authentication guarantee. Otherwise the abstraction fails since the challenge-response interpretation is not statically predictable. Similar reasoning applies to MACs.

Example 3 Suppose that $f(\llbracket A, x, z \rrbracket_{k_{AB}}) = \mathcal{R}_x^{\text{Pub,Priv}}(A, B, z)$: then we

$$\underline{f}(\llbracket A, n_A, \{n\}_{k_{BC}} \rrbracket_{k_{AB}}) = \mathcal{R}_{n_A}^{\text{Pub,Priv}}(A, B, \top)$$

, since $\{n\}_{k_{BC}}$ is non-atomic. However, we have that

$$\underline{f}(\llbracket A, n_A, m \rrbracket_{k_{AB}}) = \mathcal{R}_{n_A}^{\text{Pub,Priv}}(A, B, m)$$

given that m is, instead, atomic.

Although we can define a unique function f covering several interesting protocol classes, as discussed in the conclusion, our framework allows the programmer to extend such a function when needed. The abstraction is sound as far as f is an *encryption abstraction*, i.e., it respects the conditions reported in Table 6:

Format The only names occurring in T are identities; N , M and K are abstract terms only composed of the (abstraction of) variables occurring in T . It is important that abstractions preserve all of the encrypted variables so that, when decryption is performed, we can recover those (abstract) values from the corresponding challenge-response. We also require that the order of the abstract variables in the authenticated message M corresponds to the order in the encryption pattern: this

avoids that variables occurring free in the encryption pattern get bound in the abstract message or vice-versa. Notice that the sorting of abstract variables does not regard N and K , since in well-formed processes nonces and session-keys occur only once in the same abstract term. For the sake of readability, we have not considered the case of ciphertexts being both a challenge and a response: the extension is straightforward and is reported in Appendix D.

Unique Abstraction The encryption (and MAC) patterns in f give rise to disjoint classes of ground terms. This means that any ground run-time message G has a unique possible challenge-response interpretation. When composing different protocols, this guarantees that if two ciphertexts have the same structure, and may be thus exploited by the attacker to interleave different protocol sessions, then they also share the same challenge-response interpretation: this property is crucial for preserving the compositionality of the analysis and can be enforced by the use of message tags [18]. We verify this condition by applying a standard most general unifier.

Nesting Nested encryptions and MACs represent neither challenges nor responses and this property holds for any variable instantiation. This prevents the uncontrolled leakage of challenges and responses. As future work, we plan to take into account a more sophisticated definition of encryption abstraction so as to identify the security level and role of nested terms and to abstract them accordingly.

Encryption The security level of CR terms is consistent with the security level of the encryption keys. Public key encryption is of security level at most Tnt , digital signature at most Int and symmetric key encryption has no constraints as it is of the highest level $Priv$. Notice that it is allowed to abstract a message to a lower level of security, which is sound but could make the abstract protocol insecure even if the concrete one is safe, thus losing precision in the abstraction.

Finally, notice that names sent or received in clear do not contain enough information for telling whether they represent challenges or responses or messages without a specific role in the authentication task. Since messages circulating in clear on the network have the same computational import as public challenges and public responses (i.e., they can be derived from each other by the environment), terms of the form v , $C_v^{Pub,\ell}(I,J)$ or $R_v^{\ell,Pub}(I,J)$, where v is either a name or a variable or an input variable, are in fact freely exchangeable within the process when occurring as plaintext in input or output patterns: in fact, we identify processes differing because of the above described replacement.

Example 4 Let us consider the protocol of Example 1 and formulate the following abstraction function:

$$\begin{aligned}
 f: \quad \{B,x\}_{k_A^+} &\mapsto C_x^{Tnt,Tnt}(B,A), \\
 \{x,y_{AB},A\}_{k_B^+} &\mapsto R_x^{Tnt,Tnt}(A,B,_,y_{AB}), \\
 \{h(x),y,A\}_{x_{AB}} &\mapsto R_y^{Pub,Priv}(B,A,x)
 \end{aligned}$$

By applying the abstraction on terms defined in Table 5, and in particular the closure of the encryption abstraction defined above, we can abstract the protocol of Example 1 into the protocol of Example 2. It is easy to see that function f satisfies the conditions of Table 6 and is thus an encryption abstraction. \square

Before stating the soundness of the abstraction, we introduce the notion of authenticated key variable. At run-time, key variables x_{IJ} must be bound to actual session keys k_{IJ} so to avoid “fake” session keys sent by the enemy. This is crucial to safely abstract ciphertexts encrypted with session keys into private challenges or private responses. As stated in Section 5, validated abstract processes always guarantee this property.

Definition 3 (Authentication of key variables) *An abstract process P guarantees authentication of key variables if and only if in every reduction of P key variables of the form x_{IJ} are only instantiated with session keys of the form k_{IJ} . Similar definition applies to ρ -spi processes.*

The main result states that every concrete computation has a direct counterpart in the abstract model, given that the abstract process guaranteed authentication of key variables.

Theorem 1 (Soundness) *If $\alpha(P)$ is well-formed and guarantees authentication of key variables, then $\langle s, P \rangle \rightarrow^+ \langle s', P' \rangle$ implies $\langle \alpha(s), \alpha(P) \rangle \rightarrow_a^+ \langle \alpha(s'), \alpha(P') \rangle$.*

For easing the readability of the paper, we postpone the proofs to Appendix C.

5 Effect System

Our use of effects for locally checking the correct behavior of principals is inspired from [18]. Superseding [18] however, we do not need any typing environment as CR syntax makes explicit the role of terms, which considerably simplifies the analysis. The effect system checks the safety of nonce handshakes by relying on the following intuitive principles: the *verifier* should authenticate through an end assertion only after the successful completion of a suitable nonce handshake based on a fresh nonce and the *claimant* should respond to a received challenge only after a suitable begin assertion. For instance, A should assert $\text{end}_n(A, I, M_1, M_2, K)$ only after the output of $C_n^{\ell, \ell'}(A, I, M_1)$ (i.e., a challenge with nonce n for authenticating message M_1 , the input of $R_n^{\ell, \ell'}(I, A, M_2, K)$ (i.e., a response with the same nonce for authenticating message M_2 and session-key K) and the check on the freshness of n . Dually, B should respond to challenge $C_n^{\ell, \ell'}(I, B, M_1)$ with response $R_n^{\ell, \ell'}(B, I, M_2, K)$, only after having asserted $\text{begin}_n(B, I, M_1, M_2, K)$. For easing the analysis, we assume that session-keys circulate only within responses: this is the common way to distribute session-keys as sending them within challenges requires a further nonce handshake for verifying their freshness. Although the analysis and the abstraction can be extended to deal with such a kind of handshakes similarly to [17, 29], this is left as future work.

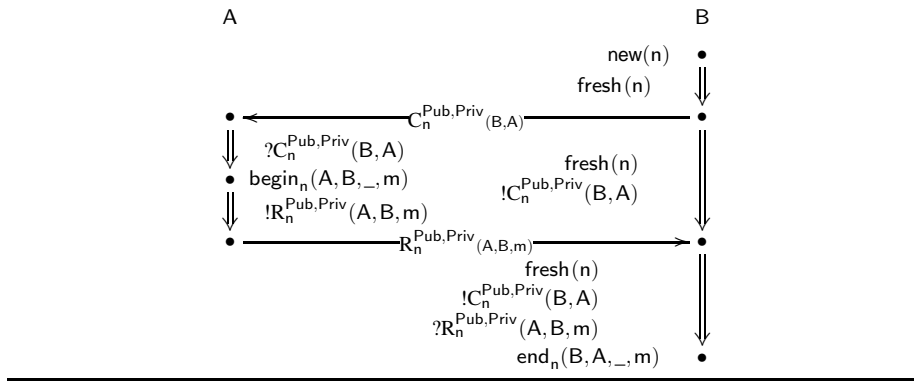
5.1 Effects and their Usefulness for Proving Safety

The link between correspondence assertions and the nonce handshake followed by principals can be achieved by tracking the generation-reception of challenges and responses and the freshness of nonces. This tracking can be conducted by means of *effects*. Formally, effects are multisets of atomic effects, as formalized by the following grammar:

$$\begin{aligned} (\text{at. eff.}) \quad f & ::= \text{fresh}(n) \mid [!]?C_N^{\ell, \ell'}(I, J, M) \mid [!]?R_N^{\ell, \ell'}(I, J, M, K) \\ (\text{eff.}) \quad e & ::= [f_1, \dots, f_n] \end{aligned}$$

As mentioned before, we assume that session-keys only circulate within responses and, consequently, the last field of the challenge effect is empty.

Table 7 CR protocol with effects



- The atomic effect $\text{fresh}(n)$ tracks the freshness of nonce n , allowing for a successive $\text{end}_n(\dots)$ on the same nonce: a new name n is fresh until it is used in an $\text{end}_n(\dots)$.
- The atomic effect $?C_N^{\ell,\ell'}(I,J,M)$ (resp. $!C_N^{\ell,\ell'}(I,J,M)$) tracks the reception (resp. generation) of a challenge with nonce N sent by I to J for authenticating message M , thus allowing the occurrence of a successive $\text{begin}_N(J,I,M,\dots,\dots)$ (resp. $\text{end}_N(I,J,M,\dots,\dots)$) assertion. The security levels ℓ and ℓ' have the same semantics as in Section 3.
- The atomic effect $?R_N^{\ell,\ell'}(I,J,M,K)$ tracks the reception of a response with nonce N sent by I to J for authenticating M and session-key K , thus allowing a successive $\text{end}_N(J,I,\dots,M,K)$.
- The atomic effect $!R_N^{\ell,\ell'}(I,J,M,K)$ has different semantics as it *enables* I to generate a response for J with nonce N for authenticating M and session-key K . This atomic effect is justified by a $\text{begin}_N(I,J,\dots,M,K)$ assertion. This asymmetry is discussed below.

5.2 Effects for a Simple CR Protocol

To illustrate the use of effects, let us consider the protocol depicted in Table 7. along with the effects used in the analysis. B generates a nonce n , whose freshness is tracked by the atomic effect $\text{fresh}(n)$, and sends it in a challenge to A: $!C_n^{\text{Pub,Priv}}(B,A)$ tracks on the verifier's code the generation of the challenge. The reception of this message is tracked by $?C_n^{\text{Pub,Priv}}(B,A)$. The following $\text{begin}_n(A,B,m)$ requires the reception of a challenge from B (effect $?C_n^{\text{Pub,Priv}}(B,A)$) and justifies the generation of a response for authenticating m (effect $!R_n^{\text{Pub,Priv}}(A,B,m)$). This is the reason why effects of the form $!R_N^{\ell,\ell'}(\dots)$ are used for *enabling* rather than tracking the generation of responses, as mentioned above. Hence $!R_n^{\text{Pub,Priv}}(A,B,m)$ enables A to generate the corresponding response, which is eventually received by B as tracked by $?R_n^{\text{Pub,Priv}}(A,B,m)$. After completing the handshake (effects $!C_n^{\text{Pub,Priv}}(B,A)$ and $?R_n^{\text{Pub,Priv}}(A,B,m)$) and checking the freshness of n (effect $\text{fresh}(n)$), B can assert $\text{end}_n(B,A,-,m)$.

Table 8 Effect System (Terms and Processes)

$\frac{\text{CHAL} \quad \ell_C \geq \text{Tnt} \vee \ell_R \geq \text{Int}}{\vdash C_N^{\ell_C, \ell_R}(I, J, M) : ([C_N^{\ell_C, \ell_R}(I, J, M)], [])}$		$\frac{\text{RESP} \quad \ell_C \geq \text{Tnt} \vee \ell_R \geq \text{Int} \quad K \neq \varepsilon \Rightarrow \ell_R \geq \text{Tnt}}{\vdash R_N^{\ell_C, \ell_R}(I, J, M, K) : ([], [R_N^{\ell_C, \ell_R}(I, J, M, K)])}$		
$\text{ATOM} \quad \vdash a : ([], [])$		$\frac{\text{PAIR} \quad \vdash T : (e_C, e_R) \quad \vdash T' : (e'_C, e'_R)}{\vdash (T, T') : (e_C + e'_C, e_R + e'_R)}$		
$\text{NIL} \quad \vdash \mathbf{0} : []$	$\frac{\text{REPL} \quad \vdash P : []}{\vdash !P : []}$	$\frac{\text{PAR} \quad \vdash P : e_P \quad \vdash Q : e_Q}{\vdash P Q : e_P + e_Q}$	$\frac{\text{ID} \quad \vdash P : e}{\vdash A \triangleright P : e}$	$\frac{\text{NEW} \quad \vdash P : e}{\vdash \text{new}(n).P : e - \text{fresh}(n)}$
$\frac{\text{NEW KEY} \quad \vdash P : e}{\vdash \text{new}(k_{AI}).P : e}$		$\frac{\text{IN} \quad \vdash P : e + ?e_C + ?e_R \quad \vdash T : (e_C, e_R)}{\vdash \text{in}(T).P : e}$		
$\frac{\text{OUT} \quad \vdash P : e + !e_C \quad \vdash T : (e_C, e_R)}{\vdash \text{out}(T).P : e + !e_R}$		$\frac{\text{BEGIN} \quad \vdash P : e + [!R_N^{\ell_C, \ell_R}(A, I, M_2, k_{AI})] \quad M_2 \text{ ground}}{\vdash \text{begin}_N(A, I, M_1, M_2, k_{AI}).P : e + [?C_N^{\ell_C, \ell_R}(I, A, M_1)]}$		
$\frac{\text{END} \quad \vdash P : e \quad M_1, \text{ground}}{\vdash \text{end}_n(A, I, M_1, M_2, K).P : e + [!C_n^{\ell_C, \ell_R}(A, I, M_1), ?R_n^{\ell_C, \ell_R}(I, A, M_2, K), \text{fresh}(n)]}$				

An abstract term is ground if it contains neither variables nor \top

5.3 Effect rules

For simplifying the presentation, we introduce some additional conventions. We write $![f_1, \dots, f_n]$ and $?[f_1, \dots, f_n]$ to denote $![f_1, \dots, !f_n]$ and $?[f_1, \dots, ?f_n]$, respectively. Furthermore, $+$ and $-$ are the usual union and subtraction operators on multisets: $e_1 + e_2$ yields the effect composed of all the atomic effects in e_1 plus the ones in e_2 , while $e_1 - e_2$ yields the effect obtained by removing, if present, an occurrence of each atomic effect in e_2 from e_1 . If an atomic effect of e_2 does not occur in e_1 then the subtraction of that atomic effect leaves e_1 unchanged.

The binding between abstract messages and their effects is formalized in Table 8 by the judgement $\vdash M : (e_C, e_R)$, read as “M has challenge effect e_C and response effect e_R ”. Notice that we check that the security levels of challenges and responses are consistent, namely either the challenge security level is greater than Tnt or the response security level is greater than Int , as discussed in Section 3. The main judgement in our analysis is $\vdash P : e$, read as “P has effect e ”, meaning that process P is safe under the conditions expressed by effect e . For instance, $\vdash P : [\text{fresh}(n)]$ means that P is safe if n is fresh. In the following we shall give informal explanations on the process judgements of Table 8. The validation of a process is defined by induction on its structure and the null process is the base case. NIL validates process $\mathbf{0}$ under empty effect since the null process is always safe. REPL validates the replication of a process under empty effect $[\]$, if that process is in turn validated under empty effect. Requiring the empty effect is necessary in order to preserve the safety; e.g., replicating a process with effect $\text{fresh}(n)$ may generate an infinite number of processes exploiting the freshness of the same nonce n to complete authentication sessions; this, of course, is

not safe as a nonce should be used only once. PAR validates the parallel composition of two processes under the union of their effects, stating that the parallel composition of two processes is safe if both of the processes are safe. ID skips identity declarations as they are not relevant in the analysis. NEW justifies, through the atomic effect $\text{fresh}(n)$, at most one use of n as fresh nonce in the continuation process. Indeed, the deletion of one occurrence of $\text{fresh}(n)$ in the thesis allows the continuation process to exploit the nonce for asserting one end assertion (cf. rule END). NEW KEY type-checks the generation of a new session key: we do not need to track the freshness of the session key since the current dialect of ρ -spi calculus does not deal with session-key corruption. The insertion of session-key corruption in this framework can be achieved similarly to [29]. IN justifies in the continuation process the reception of the challenges and responses composing the received message. OUT justifies in the continuation process the reception of the challenges and requires the permission to generate the responses composing the message sent on the network. As discussed in Section 5.2, the assertion $\text{begin}_N(A, l, M_1, M_2, K)$ (rule BEGIN) requires the reception of $C_N^{\ell_C, \ell_R}(l, A, M_1)$ and justifies the generation of $R_N^{\ell_C, \ell_R}(A, l, M_2, K)$. Similarly, the assertion $\text{end}_n(A, l, M_1, M_2, K)$ (rule END) requires the freshness of n , the generation of $C_n^{\ell_C, \ell_R}(A, l, M_1)$ and the reception of $R_n^{\ell_C, \ell_R}(l, A, M_2, K)$. Notice that we check on the syntax of begin and end assertions that messages sent in the response by the claimant are ground and so are the ones sent in the challenge by the verifier. This suffices for proving that \top never occurs within authenticated messages at run time. This is crucial for carrying safety properties from the abstract semantics to the concrete one, since \top might otherwise be instantiated differently in the matching begin and end assertions. Finally, notice that validation rules univocally determine the process effect from the one of the continuation process, with the exception of BEGIN and END which have to guess the security level of challenges and responses. However, this nondeterminism can be easily solved in the prevalidation by labelling end assertions with the security level of challenges and responses based on the same nonce, thus making the analysis fully deterministic.

5.4 Safety Results

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

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

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

Theorem 3 (Safety) *If $\alpha(P)$ is well-formed and $\vdash \alpha(P) : []$, then P is safe.*

This theorem can be easily proved by exploiting abstraction soundness of Theorem 1 and by applying the following lemma:

Lemma 1 (Key Variables) *If $\vdash P : []$, then P guarantees authentication of key variables.*

For example, the protocol of Example 2 type-checks with empty effect and it is well-formed. By Theorem 2 it is safe and, by Lemma 1, it guarantees authentication of key variables. By Theorem 3, the protocol of Table 2 is safe as well. The last theorem states that the effect system is modular and the analysis compositional.

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

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

6 Conclusion

We have presented a novel approach for concisely abstracting authentication protocols and for subsequently analyzing those abstractions in a sound manner, i.e., deriving authentication guarantees for protocol abstractions suffices for proving these guarantees for the actual protocols. The approach enjoys compositionality guarantees, allows for a modular analysis of abstractions, and ensures that many authentication protocols share a common abstraction so that validating it entails security guarantees for all these protocols and all compositions thereof. Moreover, such a validation can be performed automatically using an effect system proposed in this paper. We remark that our approach is extensible in that extensions to abstractions of additional protocol classes immediately enjoy a soundness theorem provided that these extensions satisfy the conditions on the encryption abstraction.

As future work, we plan to exploit recent results on linking symbolic cryptography with actual cryptographic algorithms to verify truly abstract authentication protocols in a way that ensures strong authentication guarantees even for concrete, cryptographic implementations.

References

- [1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.
- [2] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In *Proc. 4th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 2030 of *Lecture Notes in Computer Science*, pages 25–41. Springer-Verlag, 2001.
- [3] M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proc. 29th Symposium on Principles of Programming Languages (POPL)*, pages 33–44. ACM Press, 2002.
- [4] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.
- [5] R. M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2003.
- [6] M. Backes, A. Cortesi, R. Focardi, and M. Maffei. A calculus of challenges and responses. In *WITS '07: Proceedings of the 7th Workshop on Issues in the theory of security*, pages 101–116. Informal publication, 2007.

- [7] M. Backes, A. Cortesi, and M. Maffei. Causality-based abstraction of multiplicity in cryptographic protocols, 2007. To appear in Proc. 20th IEEE Symposium on Computer Security Foundations (CSF).
- [8] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE Computer Society Press, 2001.
- [9] B. Blanchet. From secrecy to authenticity in security protocols. In *Proc. 9th International Static Analysis Symposium (SAS)*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359. Springer-Verlag, 2002.
- [10] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proc. 6th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, pages 136–152, 2003.
- [11] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Nielson. Automatic validation of protocol narration. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 126–140. IEEE Computer Society Press, 2003.
- [12] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.
- [13] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *28rd International Colloquium on Automata, Languages and Programming (ICALP 2001)*, Lecture Notes in Computer Science, pages 667–681. Springer-Verlag, 2001.
- [14] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 157–166, 1999.
- [15] 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.
- [16] M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *Proc. 13th European Symposium on Programming (ESOP)*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer-Verlag, 2004.
- [17] M. Bugliesi, R. Focardi, and M. Maffei. Analysis of typed-based analyses of authentication protocols. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 112–125. IEEE Computer Society Press, 2005.
- [18] M. Bugliesi, R. Focardi, and M. Maffei. Dynamic types for authentication, 2007. To appear in *Journal of Computer Security*.
- [19] A. Datta, A. Derek, J.C. Mitchell, and A. Roy. Protocol composition logic (pcl). *Electronic Notes on Theoretical Computer Science*, 172:311–358, 2007.
- [20] Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.

- [21] S.F. Doghmi, J.D. Guttman, and F.J. Thayer. Searching for shapes in cryptographic protocols. In *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 523–538. Springer-Verlag, 2007.
- [22] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
- [23] 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.
- [24] H. Gao, P. Degano, C. Bodei, and H. R. Nielson. Detecting replay attacks by freshness annotations. In *WITS '07: Proceedings of the 7th Workshop on Issues in the theory of security*, pages 85–100. Informal publication, 2007.
- [25] A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3):435–484, 2004.
- [26] J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 24–34. IEEE Computer Society Press, 2000.
- [27] J. D. Guttman and F. J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.
- [28] G. Lowe. “A Hierarchy of Authentication Specification”. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE Computer Society Press, 1997.
- [29] M. Maffei. *Dynamic Typing for Security Protocols*. PhD thesis, Ca’ Foscari University, 2006.
- [30] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proc. 2000 DARPA Information Survivability Conference and Exposition (DISCEX)*, pages 237–250, 2000.
- [31] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 166–175. ACM Press, 2001.
- [32] T. Y. C. Woo and S. S. Lam. A lesson on authentication protocol design. *Operation Systems Review*, 28(3):24–37, 1994.

A ρ – spi calculus 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 $\langle s, P \rangle \rightarrow \langle s :: \alpha, P' \rangle$ simulates one computation step in P and records the corresponding action in the trace. At run-time, term variables can be bound to run-time messages producing special terms containing a mixture of $\{ \dots \}_k$ and $\{ \dots \}^k$. We call these extended terms *run-time terms*, ranged over by R . Run-time terms can occur in input and output primitives only, given that calculus syntax forbids the use of cryptography in the other primitives. For this reason, at run-time, begin and end

Table 9 Transition System for ρ -spi

Notation: We omit the symmetric rule of PAR.

REPLICATION $\langle s, !P \rangle \rightarrow \langle s, P \mid !P \rangle$	NEW NAME AND KEY $\frac{v \in \{n, k_{IJ}\} \quad v \notin \text{bn}(s) \cup \text{fn}(s)}{\langle s, \text{new}(v).P \rangle \rightarrow \langle s :: \text{new}(v), P \rangle}$
INPUT $\frac{s \vdash G \quad \sigma = \text{bind}(R, G) \neq \uparrow}{\langle s, \text{in}(R).P \rangle \rightarrow \langle s :: \text{in}(R\sigma), P\sigma \rangle}$	OUTPUT $\langle s, \text{out}(R).P \rangle \rightarrow \langle s :: \text{out}(R), P \rangle$
BEGIN $\langle s, \text{begin}_G(A, I, G').P \rangle \rightarrow \langle s :: \text{begin}_G(A, I, G'), P \rangle$	
END $\langle s, \text{end}_G(A, I, G').P \rangle \rightarrow \langle s :: \text{end}_G(A, I, G'), P \rangle$	PRINCIPAL $\frac{\langle s, P \rangle \rightarrow \langle s :: \alpha, P' \rangle}{\langle s, A \triangleright P \rangle \rightarrow \langle s :: A \triangleright \alpha, A \triangleright P' \rangle}$
PAR $\frac{\langle s, P \rangle \rightarrow \langle s', P' \rangle}{\langle s, P \mid Q \rangle \rightarrow \langle s', P' \mid Q \rangle}$	
$\text{bind}(a, a) = []$	
$\text{bind}(k, k) = []$	
$\text{bind}(?x, a) = [G/x]$ G is neither a tagged term nor a pair	
$\text{bind}(\text{Tag}(R), \text{Tag}(G)) = \text{bind}(R, G)$	
$\text{bind}((R, R'), (G, G')) = \sigma \uplus \text{bind}(R'\sigma, G')$ where $\sigma = \text{bind}(R, G)$	
$\text{bind}(\{\! R \!\!\}_{\bar{k}}, \{G\}_k) = \text{bind}(R, G)$	
$\text{bind}(\text{MAC}_k(R), \text{MAC}_k(G)) = []$ $\text{bind}(R, G) = [] \wedge R$ ground	
$\text{bind}(h(R), h(G)) = []$ $\text{bind}(R, G) = [] \wedge R$ ground	
$\text{bind}(R, G) = \uparrow$ otherwise	
$\sigma_1 \uplus \sigma_2 = \sigma_1 \cup \sigma_2$ if $\sigma_1, \sigma_2 \neq \uparrow \wedge x \in \text{dom}(\sigma_1) \cap \text{dom}(\sigma_2) \Rightarrow \sigma_1(x) = \sigma_2(x)$	
$\sigma_1 \uplus \sigma_2 = \uparrow$ otherwise	

will only contain run-time messages G . The set of all possible actions, noted Act , includes the action $\text{new}(n)$ generated by restriction, $\text{in}(R)$ by input, $\text{out}(R)$ by output, $\text{begin}_G(A, I, G')$ and $\text{end}_G(A, I, G')$ by ‘begin’ and ‘end’. Actions are prefixed by $A \triangleright$ when executed by a principal A .

Some transitions apply substitutions to processes: a substitution $\sigma : x \mapsto G$ is a function from variables to run-time messages. Often substitutions are written explicitly by $[G_1/x_1, \dots, G_n/x_n]$. The application of the substitution σ to the process P is denoted by $P\sigma$ and applies only to free occurrences of the variables in P . We use a number of notation conventions. The restriction operators $\text{new}(n).P$ and $\text{new}(k_{IJ}).P$ are binders for name n and session key k_{IJ} , respectively; the input primitive is a binder for the input variables that occur in term T . In all cases the scope of the binders is the continuation process. Moreover, term T of input primitive is processed from left to right and the scope of the binding of input variables is the remaining part of T . For example, $\text{in}(?x, x)$ binds the second occurrence of x to the first (input) one. Thus, such an input succeeds only if the first and second components are the same. Similarly, $\text{new}(n)$ is a binder for names and its scope is the continuation trace. The notions of free/bound names and

Table 10 Message Manipulation Rules

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

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

variables, both for processes and traces, arise as expected.

Table 9 collects transition rules. REPLICATION arbitrarily replicates a principal by introducing an unbounded number of copies thereof in the system configuration. RES generates a new name n by checking that it differs from all the names already used in the trace s . It is possible to force this condition by applying α -conversion to n , i.e., by substituting n and all of its free occurrences in P with a different name. Indeed, as in companion transition systems, see, e.g. [14], we implicitly identify processes up to renaming of bound variables and names, i.e., up to α -equivalence. We also assume that the sets of free and bound names are disjoint and that key α -conversion preserves identity labels, i.e., k_{AB} can be converted to k'_{AB} or k'_{BA} . INPUT requires message G , read from the network, to be computable by the environment: the environment knowledge is defined by the message manipulation rules reported in Table 10 and discussed below. The run-time message G is read only if it can be pattern-matched with the input term T via the function $bind$, which is discussed below. In such a case, all the variables in T are bound to the respective submessages of G producing the run-time term $T\sigma$. OUTPUT, BEGIN and END are self-explanatory. Finally, PRINCIPAL adds the principal name to the performed action, and PAR interleaves two different protocol executions.

Pattern-matching is formalized through the function $bind : R \mapsto \sigma$. We write \bar{k} to denote the decryption key corresponding to k . For symmetric cryptography we have $\overline{k_{AB}} = k_{AB}$, for asymmetric it holds $\overline{k_A^+} = k_A^-$ and $\overline{k_A^-} = k_A^+$. Function $bind$ takes as input a static term T and a run-time message G and, in case it exists, yields the substitution σ which makes T equal to G , up to the different notation for encryption. If pattern-matching fails, $bind$ returns \uparrow . It is defined by cases on the structure of term T : names and keys are tested by equality and, if the matching succeeds, $bind$ returns the empty substitution. Similarly, since hashes and MACs are not invertible, no variable can be instantiated. A variable can be bound to atomic names, keys, ciphertexts, hashes and MACs; tagged terms require the same tag, and pairs are pattern-matched from left to right, yielding a substitution which is the union \uplus of the ones achieved for the subterms; the union $\sigma_1 \uplus \sigma_2$ succeeds only when variables substituted by both σ_1 and σ_2 are bound to the same run-time message; finally, decryptions must be performed with the correct decryption key. In the other cases, $bind$ returns \uparrow .

The message manipulation rules, in Table 10, formalize the environment actions. Rule OUT says that every message sent on the network is known to the environment.

Table 11 CR calculus: Binding abstract terms and run-time messages

$$\begin{aligned} \text{bind}(a, a) &= [] \\ \text{bind}(?x, a) &= [a/x] \\ \text{bind}((T_1, T_2), (G_1, G_2)) &= \sigma \uplus \text{bind}(T_2 \sigma, G_2) && \text{where } \sigma = \text{bind}(T_1, G_1) \\ \text{bind}(C_N^{(\ell, \ell')}(A, B, M, K), C_{G_1}^{(\ell, \ell')}(A, B, G_2, G_3)) &= \text{bind}(N, G_1) \uplus \text{bind}(M, G_2) \uplus \text{bind}(K, G_3) \\ \text{bind}(R_N^{(\ell, \ell')}(A, B, M, K), R_{G_1}^{(\ell, \ell')}(A, B, G_2, G_3)) &= \text{bind}(N, G_1) \uplus \text{bind}(M, G_2) \uplus \text{bind}(K, G_3) \\ \text{bind}(T, G) &= \uparrow && \text{otherwise} \end{aligned}$$

Table 12 CR calculus: Abstract Message Manipulation Rules

$\frac{\text{OUT} \quad \text{out}(G) \in s}{s \vdash G}$	$\frac{\text{ENV} \quad a \notin \text{bn}(s)}{s \vdash a}$	$\frac{\text{PAIR} \quad s \vdash G_1 \quad s \vdash G_2}{s \vdash (G_1, G_2)}$	$\frac{\text{PAIR DES} \quad s \vdash (G_1, G_2) \quad i \in [1, 2]}{s \vdash G_i}$	$\text{TOP} \quad s \vdash \top$
$\frac{\text{WRITABLE} \quad s \vdash G_1, G_2, G_3 \quad \ell \leq \text{Tnt} \vee I = E \vee J = E}{s \vdash C_{G_1}^{\ell, \ell'}(I, J, G_2, G_3) \quad s \vdash R_{G_1}^{\ell, \ell'}(I, J, G_2, G_3)}$				
$\frac{\text{READABLE} \quad s \vdash C_{G_1}^{\ell, \ell'}(I, J, G_2, G_3) \vee s \vdash R_{G_1}^{\ell, \ell'}(I, J, G_2, G_3) \quad \ell \leq \text{Int} \vee I = E \vee J = E}{s \vdash G_1, G_2, G_3}$				
$\frac{\text{CORRUPTED WRITE} \quad s \vdash G_1, G_2, G_3 \quad s \vdash k_{IJ} \vee s \vdash k_{JI}}{s \vdash C_{G_1}^{\ell, \ell'}(I, J, G_2, G_3) \quad s \vdash R_{G_1}^{\ell, \ell'}(I, J, G_2, G_3)}$				
$\frac{\text{CORRUPTED READ} \quad s \vdash C_{G_1}^{\ell, \ell'}(I, J, G_2, G_3) \quad s \vdash R_{G_1}^{\ell, \ell'}(I, J, G_2, G_3) \quad s \vdash k_{IJ} \vee s \vdash k_{JI}}{s \vdash G_1, G_2, G_3}$				

ENV allows the environment to know any name which is not bound (i.e., restricted) in the trace. By PAIR and PAIR DES, the environment may construct and destruct pairs. TAG and TAG DES allow the environment to tag and untag messages. By ENCRYPTION, and DECRYPTION the environment can encrypt and decrypt messages only knowing the required keys. HASH and MAC allow the environment to generate hashes and MACs: notice that rule MAC requires the knowledge of the symmetric key used by the MAC algorithm. By PUBLIC KEYS, all the public keys are known to the environment. Finally, by ENEMY KEYS, the environment may be provided with its own private keys and with long-term keys shared with honest participants. This gives the possibility to the enemy to start authentication sessions and to interact with other participants by pretending to be trusted.

B CR calculus semantics

Table 13 Abstraction for the environment

Run Time Messages	
$\alpha_e(a)$	$= a$
$\alpha_e(G_1, G_2)$	$= (\alpha_e(G_1), \alpha_e(G_2))$
$\alpha_e(\text{Tag}(G))$	$= \alpha_e(G)$
$\alpha_e(h(G))$	$= \alpha_e(G)$
$\alpha_e(\{G\}_k)$	$= \begin{cases} \underline{f}_e(\{G\}_k) & \text{if } \{G\}_k \in \text{dom}(\underline{f}_e) \\ \alpha_e(G) & \text{otherwise} \end{cases}$
$\alpha_e(\text{MAC}_k(G))$	$= \begin{cases} \underline{f}_e(\text{MAC}_k(G)) & \text{if } \text{MAC}_k(G) \in \text{dom}(\underline{f}_e) \\ \alpha_e(G) & \text{otherwise} \end{cases}$
where $\underline{f}_e([R]) \stackrel{\text{def}}{=} \underline{f}(R)$	

The semantics of CR calculus is based on the same transition rules of ρ -spi calculus presented in previous section (Table 9). The only difference is the way terms are bound and deducted by the intruder. Table 11 defines the function `bind` used in the input primitive of CR calculus. Table 12 describes message manipulation rules for abstract messages. Notice that security labels are not violated by the intruder, similarly to what is done with cryptography in the ρ -spi calculus. The only exception concerns session-key corruption: if the intruder gets the knowledge of session key k_{IJ} , then it can create and read the content of challenges and responses from I to J and vice-versa, regardless of their security levels. These rules abstract encryptions and decryptions performed by the attacker through the corrupted session-key k_{IJ} .

C Proofs of Soundness and Safety

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

C.1 Soundness of the Abstraction

We use a special abstraction $\alpha_e(G)$ of run-time messages G , defined in table 13. As proved by the following lemma, this abstraction is more precise than the usual α , as we need to consider all the potential challenges and responses, independently from the fact that processes will input them.

Lemma 2 (Environment Abstraction) *If $\alpha(R) \neq \perp$ and $s \vdash \alpha_e([R])$, then $s \vdash \alpha(R)$.*

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

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

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

□

The next lemma states that, for every abstract trace s , abstract names occurring as nonces and messages in an abstraction of a ciphertext (resp. MAC) are derivable from s if and only if the abstraction of the ciphertext (resp. MAC) content is also derivable from s .

Lemma 3 (Cryptographic Content) *Let $\{G\}_k \in \text{dom}(\underline{f}_e)$ (resp. $\text{MAC}_k(G) \in \text{dom}(\underline{f}_e)$) and $\alpha_e(\{G\}_k) = C/R_{G_1}^{\ell, \ell'}(1, J, G_2)$ (resp. $\alpha_e(\text{MAC}_k(G)) = C/R_{G_1}^{\ell, \ell'}(1, J, G_2)$). Then, for every abstract trace s , $s \vdash \alpha(G)$ if and only if $s \vdash G_1, G_2$*

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

The following definition introduces a notion of well-formedness for ρ -spi traces, requiring that any ciphertext and MAC, which is sent as output and is not generated therein, has been received before. It is easy to see that every trace generated by a ρ -spi process is well-formed. For this reason, in the following we shall only consider well-formed traces.

Definition 4 (Trace Well-Formedness) *A trace s is well-formed if and only if the following condition holds:*

- if $s = s' :: \text{out}(R)$ and $\{G\}_k \in \text{terms}(R)$ (resp. $\text{MAC}_k(G) \in \text{terms}(R)$), then $\exists G'$ s.t. $\text{in}(G') \in s'$ and $\{G\}_k \in \text{terms}(G')$ ($\text{MAC}_k(G) \in \text{terms}(G')$)

The next lemma states that the environment knows any ciphertext and MAC having a challenge-response interpretation and circulating on the network, even if nested.

Lemma 4 (Nesting and Environment's Knowledge) *Let s be a trace such that $\perp \notin \text{terms}(\alpha(s))$ and $s \vdash G$ implies $\alpha(s) \vdash \alpha_e(G)$. Then $\forall \sigma, G, G'$ such that*

- $s \vdash G$ and
- $G' \in \text{terms}(G)$ and
- $G' = [T\sigma]$, for some $T \in \text{dom}(f)$,

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

Proof. For the sake of readability, we shall reason on ciphertexts as the proof for MACs is the same. By induction on the length of s and on the derivation length of $s \vdash G$.

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

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

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

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

Base Cases The only interesting case is Out:

Out The judgement $s \vdash [R]$ derives by $out(R) \in s$. Let us suppose that $G' \in terms([R])$ and $G' = [T\sigma]$, for some $T \in dom(f)$. If G' is not generated by an encryption in R , then, since the trace is well-formed (cf. Definition 4), it turns out that G' has been previously received as input, possibly nested: since we assume that \perp does not occur in $\alpha(s)$, the thesis derives directly from the induction hypothesis. Let us suppose that G' is generated by an encryption in R . Recall that $G' = [T\sigma]$. Since $\alpha(R) \neq \perp$ and the trace is well-formed, $\forall \{G''\}_k \in range(\sigma)$, $\{G''\}_k$ has been previously received as input, possibly nested. If it occurred within an encryption representing either a challenge or a response, then the thesis follows from induction hypothesis. Otherwise, the thesis follows from definition of α_e and a sequence of applications of the CR manipulation rule PAIR DES.

Inductive Cases The proof derives straightforwardly from the induction hypothesis.

□

Finally, we discuss the proof of the main proposition.

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

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

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

Base Cases

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

Encryption In the following, we discuss public-key encryption: the proof for encryptions via enemy-keys is similar. From $\varepsilon \vdash G$ we may derive $\varepsilon \vdash \{G\}_{k_1^+}$. By induction, $\varepsilon \vdash \alpha_e(G)$. We now have two cases: if $\{G\}_{k_1^+} \notin dom(\underline{f}_e)$ we have that $\alpha_e(\{G\}_{k_1^+}) = \alpha_e(G)$ thus trivially obtaining the thesis. Otherwise, $\alpha_e(\{G\}_{k_1^+}) = \underline{f}_e(\{G\}_{k_1^+})$. By condition *Encryption* of Table 6, we know that $\underline{f}_e(\{G\}_{k_1^+})$ can be either $C_{G_1}^{\ell, \ell'}(J, I, G_2)$ or $R_{G_1}^{\ell, \ell'}(J, I, G_2)$, with $\ell \leq Tnt$. Moreover, by Lemma 2, we have that $\varepsilon \vdash \alpha_e(G)$ implies $\varepsilon \vdash \alpha(G)$. Notice that $\alpha(G) \neq \perp$ as run-time messages are never abstracted into \perp . By Lemma 3 we obtain that $\varepsilon \vdash G_1, G_2$. By rule *Forgeable*, we obtain that $\varepsilon \vdash \underline{f}_e(\{G\}_{k_1^+})$, i.e., $\varepsilon \vdash \alpha_e(\{G\}_{k_1^+})$.

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

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

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

Out The judgement $s \vdash [R]$ derives by $out(R) \in s$. By trace abstraction we know that $out(\alpha(R)) \in \alpha(s)$ and, by **Out**, $\alpha(s) \vdash \alpha(R)$. Since we assume $\alpha(R) \neq \perp$, $\alpha(R)$ and $\alpha_e([R])$ may only differ for some $\{G\}_k \in terms(R)$ which is abstracted into \top in $\alpha(R)$. However, by Definition 4, a message $\{G\}_k$ can be present in R only if it has been previously read from the environment. Let s' be the prefix of s where such an input occurs. It is easy to see that $\alpha(s') \vdash \alpha_e(\{G\}_k)$. If $\{G\}_k$ was nested into a ciphertext with challenge-response interpretation, then the result follows from Lemma 4: notice that we can now apply the induction hypothesis so that the hypothesis of Lemma 4 holds. Otherwise, it just follows by definition of α_e , induction hypothesis and repeated application of the CR manipulation rule **PAIR DES**.

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

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

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

Decryption This case is new, since with empty trace we had no encryptions apart the ones generated by the environment itself. We discuss private key decryption. From $s \vdash \{G\}_{k_I^-}$ we may derive $s \vdash G$. Assume that $\{G\}_{k_I^-}$ has not been generated by the environment, otherwise the thesis is trivially achieved as for the case with empty trace above. By induction, $\alpha(s) \vdash \alpha_e(\{G\}_{k_I^-})$. We now have two cases: if $\{G\}_{k_I^-} \notin dom(\underline{f}_e)$ we have that $\alpha_e(\{G\}_{k_I^-}) = \alpha_e(G)$ thus trivially obtaining the thesis. Otherwise, $\alpha_e(\{G\}_{k_I^-}) = \underline{f}_e(\{G\}_{k_I^-})$. By condition *Encryption* of Table 6, we know that $\underline{f}_e(\{G\}_{k_I^-})$ can be either $C_{G_1}^{\ell, \ell'}(I, J, G_2)$ or $R_{G_1}^{\ell, \ell'}(I, J, G_2)$, with $\ell \leq \text{Int}$. By rule **Readable**, we obtain that $\alpha(s) \vdash G_1, G_2$. By Lemma 3 we obtain that $\alpha(s) \vdash \alpha(G)$. We need to show that this implies $\alpha(s) \vdash \alpha_e(G)$. Recall that we have assumed that $\{G\}_{k_I^-}$ has not been generated by the environment. Thus it must have been sent as output by a process, namely $out(R) \in s$ with either $\{G\}_{k_I^-} \in terms(R)$ or $\{G\}_{k_I^-} \in terms([R])$. In the former case, the result follows from the well-formedness of s and Lemma 4. The latter case is more interesting as it represents an encryption performed by a process. By condition *Format* of Table 6, all terms, apart from nested run-time ciphertexts, are preserved by the abstraction. Furthermore, by condition *Nesting* of Table 6, nested encryptions $\{\!\| R \!\|\}_k$ do not belong to the domain of \underline{f}_e and $\alpha_e(\{\!\| R \!\|\}_k)$ is thus by definition $\alpha_e(R)$. Run time ciphertexts are abstracted into \top : however, by induction hypothesis this have been previously received from the network and, by Lemma 4, their abstraction is already known to the environment. Hence, we obtain the thesis, i.e., $\alpha(s) \vdash \alpha_e(G)$.

□

The next corollary is useful for the final theorem:

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

Proof. By the previous proposition we know that $s \vdash [T]\sigma$ implies $\alpha(s) \vdash \alpha_e([T]\sigma)$. The rest of the proof is easy since (i) α_e and α may only differ in ciphertexts that are mapped to some CR term by the former and to \top by the latter, and (ii) message \top is derivable from any trace. □

The following lemma says that the abstraction of terms, traces and processes is closed under any valid substitution, i.e., any substitution of (i) variables with names, ciphertexts, hashes and MACs and (ii) key variables of the form x_{IJ} with session keys of the form k_{IJ} (cf. Table 5). Here and throughout this paper, we write \mathcal{P} to denote an arbitrary pattern, namely either a term or a trace or a process.

Lemma 5 (Substitution) For every valid σ and \mathcal{P} , $\alpha(\mathcal{P}\sigma) = \alpha(\mathcal{P})\sigma^\sharp$.

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

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

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

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

$\mathcal{P} = (R_1, R_2)$ The result is obtained directly by the induction hypothesis. Similar reasoning applies to tagged terms and hash generation.

$\mathcal{P} = \{\!| R \!|\}_k$ If $\{\!| R \!|\}_k \in \text{dom}(f)$, then $\{\!| R \!|\}_k\sigma \in \text{dom}(f)$ as f is the closure of f under variable substitution. Thus $\alpha(\{\!| R \!|\}_k\sigma) = f(\{\!| R \!|\}_k\sigma) \triangleq f(\{\!| R \!|\}_k)\sigma^\sharp = \alpha(\{\!| R \!|\}_k)\sigma^\sharp$, as desired. If $\{\!| R \!|\}_k \notin \text{dom}(f)$, then the result is obtained by the induction hypothesis. The reasoning for mac generation is similar.

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

Traces and Processes By trivial induction on the length of the trace and the process, respectively, and by the previous item. □

The next lemma states that the abstraction of processes guaranteeing authentication of session keys (cf. Definition 3) is preserved under reduction.

Theorem 1 (Soundness) If $\alpha(P)$ is well-formed and guarantees authentication of key-variables, then $\langle s, P \rangle \rightarrow \langle s', P' \rangle$ implies $\langle \alpha(s), \alpha(P) \rangle \rightarrow \langle \alpha(s'), \alpha(P') \rangle$

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

Base By cases, according to the semantic rule applied. We only discuss the most interesting case, namely INPUT, as the other ones follow straightforwardly from an inspection of the concrete reduction rules and the corresponding abstract ones. We thus reason on reductions of the form $\langle s, \text{in}(T).P \rangle \rightarrow \langle s :: \text{in}(t)\sigma, P\sigma \rangle$, for some σ . By the ρ -spi rule INPUT, $s \vdash \overline{[T]}\sigma$. By Corollary 1, $\alpha(s) \vdash \alpha(\overline{[T]}\sigma)$. By Lemma 5, $\alpha(\overline{[T]}\sigma) = \alpha(\overline{[T]}\sigma)^\sharp$. Since $\alpha(P)$ guarantees authentication of key variables, it is easy to see that σ is valid. By the CR semantic rule INPUT, $\langle \alpha(s), \alpha(P) \rangle \rightarrow \langle \alpha(s) :: \text{in}(\alpha(\overline{[T]}\sigma)^\sharp), \alpha(P)\sigma^\sharp \rangle$. By Lemma 5, $\alpha(P\sigma) = \alpha(P)\sigma^\sharp$, as desired.

Induction The proof straightforwardly derives from the induction hypothesis. □

C.2 Safety of the Analysis

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

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

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

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

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

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

Definition 5 (Balanced Configuration) *Let P be a process and e an effect such that $\vdash P : e$. We say that a trace s , P and e are balanced iff the following conditions hold:*

1. $\text{names}(e) \subseteq \text{fn}(s) \cup \text{bn}(s)$

2. if $n \in bn(s)$ and $n \in fn(P)$, then $n \notin bn(P)$
3. if $s \vdash C/R_n^{\ell, \ell'}(A, B, M, K)$ and $s \not\vdash n$, then $n \notin msgs(P) \cup plaintexts(P)$
4. $|fresh(n)|_e + |end_n(\cdot)|_s \leq |new(n)|_s \leq 1$
5. if $P = P_1 | P_2$ then $\exists e_1, e_2$ s.t. $e_1 + e_2 = e$ and s, P_i, e_i are balanced, $\forall i \in [1, 2]$
6. if $out(\dots, C_n^{\ell, \ell'}(A, B, M), \dots) \in P$
then (i) $s \vdash N \Rightarrow n \notin names(N)$ and (ii) $n \notin plaintexts(P) \cup msgs(P)$
7. if $[!|?]C_n^{\ell, \ell'}(A, B, M) \in e$ (resp. $?R_n^{\ell, \ell'}(A, B, M, K) \in e$),
then $s \vdash C_n^{\ell, \ell'}(A, B, M) \in e$ (resp. $s \vdash R_n^{\ell, \ell'}(A, B, M, K)$ and $K = k_{AB}$, for some k_{AB})
8. if $!R_n^{\ell_C, \ell_R}(A, B, M_2, K) \in e$,
then $begin_n(A, B, M_1, M_2, K) \in s$, $s \vdash C_n^{\ell'_C, \ell'_R}(B, A, M_1)$, with $\ell'_C, \ell'_R \diamond \ell_C, \ell_R$, and $K = k_{AB}$, for some k_{AB} .
9. if $new(n) \in s$ and $s \vdash C/R_n^{\ell_C, \ell_R}(A, B, \dots)$, with $\ell_C, \ell_R \geq Tnt$,
then $s \not\vdash n$
10. if $out(\dots, R_n^{\ell_C, \ell_R}(B, A, M_2, K), \dots) \in s$,
then $begin_n(B, A, M_1, M_2, K) \in s$, and $s \vdash C_n^{\ell'_C, \ell'_R}(A, B, M_1)$, with $\ell'_C, \ell'_R \diamond \ell_C, \ell_R$
11. if $!C_n^{\ell, Pub}(B, A, M) \in e$ and $s \vdash n$, with $\ell \geq Tnt$,
then $begin_n(A, B, M) \in s$
12. if $fresh(n) \in e$ and $out(\dots, C_n^{\ell, \ell'}(A, B, M), \dots) \in s$,
then $!C_n^{\ell, \ell'}(A, B, M) \in e$

The intuitive reading is as follows:

1. The names in the effect occur in the trace.
2. Processes are α -converted so as to avoid clashes in bound names
3. The nonces, which circulate on the network within challenge-response messages and are not known to the enemy, do not occur within plaintexts or authenticated messages.

4. If a bound name n is a nonce, then either n is fresh or one $end_n(\cdot)$ has been asserted.
5. Balancing is propagated through parallel composition.
6. If a nonce is sent as challenge, then it is unknown to the enemy and it does not occur as either plaintext or authenticated message in any following output.
7. The presence of effects *tracking* the reception-generation of challenge-response messages implies the presence of such messages in the knowledge of the environment.
8. The presence of effects *justifying* the generation of a response implies that a suitable *begin* assertion has been asserted and the corresponding challenge is known to the environment.
9. The nonces sent in a Priv or Tnt challenge (resp. response) requiring a Priv or Tnt response (resp. challenge) are not known to the environment.
10. If B sends a response with nonce n and message M_2 ($out(\dots, R_n^{\ell_C, \ell_R}(B, A, M_2), \dots) \in s$), then B has asserted $begin_n(B, A, M_1, M_2)$ and the environment knows the corresponding challenge ($s \vdash C_n^{\ell_C, \ell_R}(A, B, M_1)$).
11. If the nonce is sent in a challenge requiring a Pub response ($!C_n^{\ell, \text{Pub}}(B, A, M) \in e$) and it is known to the environment ($s \vdash n$), then A has asserted $begin_n(A, B, M)$.
12. This condition says that if the nonce n is fresh ($fresh(n) \in e$) and a challenge with nonce n and message M is sent on the network ($out(\dots, C_n^{\ell, \ell'}(A, B, M), \dots) \in s$), then A has not completed the handshake yet ($!C_n^{\ell, \ell'}(A, B, M) \in e$).

The next theorem states that successfully validated processes preserve under reduction the invariants of Definition 5.

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

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

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

- $\vdash P : e'$
- s', P, e' are balanced

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

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

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

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

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

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

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

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

The proof for the restriction of session keys is similar.

BEGIN $\boxed{\text{begin}_N(A, I, M_1, M_2, k_{A1}).P : e + [?C_N^{\ell_C, \ell'_R}(I, A, M_1)] \rightarrow P : e + [!R_N^{\ell_C, \ell'_R}(A, I, M_2, k_{A1})]}$

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

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

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

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

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

Condition 6 Let us suppose by contradiction that $\text{out}(\dots, C_n^{\ell_C, \ell'_R}(A, B, M), \dots) \in P$, for some ℓ_C, ℓ'_R , and the output of R reveals a term containing n , namely $s :: \text{out}(R) \vdash M$ with $n \in \text{names}(M)$. Since the source configuration is balanced, by condition 6 $n \notin \text{msgs}(P) \cup \text{plaintexts}(P)$. Let us suppose that

$C_n^{\ell_C, \ell_R}(\dots) \in \text{terms}(\mathbb{R})$, for some ℓ_C, ℓ_R . Since $C_n^{\ell_C, \ell_R}(\dots) \in e_C$, and the process is successfully validated, $\text{fresh}(n) \in e$. By condition 4, $\text{new}(n) \in s$. By condition 2, $n \notin \text{bn}(P)$. The contradiction arises by Lemma 6. The reasoning is similar for $R_n^{\ell_C, \ell_R}(\dots) \in \text{terms}(\mathbb{R})$ and relies on the balancing conditions 8 and 12.

Condition 9 Let us suppose by contradiction that $\text{new}(n) \in s$, $s :: \text{out}(\mathbb{R}) \vdash n$ and $s :: \text{out}(\mathbb{R}) \vdash C/R_n^{\ell, \ell'}(A, B, M, K)$, with $\ell, \ell' \geq \text{Tnt}$. Since the source configuration is balanced, by condition 9 at least one between n and $C/R_n^{\ell, \ell'}(A, B, M)$ is unknown to the environment. We proceed by cases according to what is revealed by the output of \mathbb{R} :

$s \vdash C/R_n^{\ell, \ell'}(A, B, M, K)$ and $s \not\vdash n$ We proceed by cases on the form of \mathbb{R} :

$\mathbb{R} = \dots, n, \dots$

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

$\mathbb{R} = \dots, C_N^{\ell_C, \ell_R}(I, J, M'), \dots$, with $n \in \text{names}(N, M')$ and $\ell_C \leq \text{Int}$ Since the source configuration is balanced and $s \vdash C/R_n^{\ell, \ell'}(A, B, M, K)$, by condition 3, $n \notin \text{names}(M')$. Thus we get $n \in \text{names}(N)$ and $C_N^{\ell_C, \ell_R}(I, J, M') \in e_R$. Since the process is successfully validated, it is easy to see that $\text{fresh}(n) \in e$ and $n = N$. Let us suppose that $s \vdash C_n^{\ell, \ell'}(A, B, M)$: since $s \not\vdash n$, we get $\text{out}(\dots, C_n^{\ell, \ell'}(A, B, M), \dots) \in s$. By condition 12, $!C_n^{\ell, \ell'}(A, B, M) \in e$. The contradiction arises by Lemma 6. If $s \vdash R_n^{\ell, \ell'}(A, B, M, K)$, then the reasoning is similar and relies on the balancing conditions 8 and 12.

$\mathbb{R} = \dots, R_N^{\ell_C, \ell_R}(I, J, M', K'), \dots$, with $n \in \text{names}(N, M', K')$ and $\ell_C \leq \text{Int}$

By OUTPUT, $R_N^{\ell_C, \ell_R}(I, J, M', K') \in e_R$ and, by condition 8, we have that $s \vdash C_N^{\ell_C, \ell_R}(J, I, M'')$, with $\ell'_C, \ell'_R \diamond \ell_C, \ell_R$. By the message manipulation rule READABLE, $s \vdash N, M'$ and, consequently, $s \vdash n$. This contradicts the hypothesis $s \not\vdash n$.

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

$s \not\vdash n$ and $s \not\vdash C/R_n^{\ell, \ell'}(A, B, M, K)$ The reasoning is similar to the one in the previous items.

Condition 10 The proof follows directly from condition 8.

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

END $\text{end}_n(A, I, M_1, M_2, K).P : e + [\text{fresh}(n), !C_n^{\ell_C, \ell_R}(A, I, M_1), ?R_n^{\ell_C, \ell_R}(I, A, M_2, K)] \rightarrow P : e$

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

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

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

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

$$\text{IDENTITY} \quad \boxed{A \triangleright P : e \quad \rightarrow \quad A \triangleright P' : e'}$$

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

$$\text{PAR} \quad \boxed{P_1 | P_2 : e_1 + e_2 \quad \rightarrow \quad P'_1 | P_2 : e'_1 + e_2}$$

By PAR, $\vdash P_i : e_i$, with $i \in [1, 2]$. Let us suppose that $\langle s, P_1 \rangle \rightarrow \langle s', P'_1 \rangle$: the symmetric case is analogous. By induction hypothesis, we can find an effect e'_1 s.t. $\vdash P'_1 : e'_1$. By PAR, $\vdash P'_1 | P_2 : e'_1 + e_2$.

By hypothesis the source configuration is balanced and, by condition 5, s, P_i, e_i are balanced, $\forall i \in [1, 2]$. By induction hypothesis, even s', P'_1, e'_1 are balanced. It remains to prove that $s', P'_1 | P_2, e'_1 + e_2$ are balanced. We only discuss the interesting cases.

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

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

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

Condition 12 Let us suppose by contradiction that condition 12 does not hold.

This means that $\text{fresh}(n) \in e'_1 + e_2$, $\text{out}(\dots, C_n^{\ell, \ell'}(A, B, M_1), \dots) \in s :: \alpha$ and $!C_n^{\ell, \ell'}(A, B, M_1) \notin e'_1 + e_2$. The only interesting case is when $\alpha = \text{end}_n(A, B, M_1, M_2, K)$ with $\text{fresh}(n) \in e_2$ as $!C_n^{\ell, \ell'}(A, B, M_1)$ is removed by END, i.e., $e_1 = e'_1 + [\text{fresh}(n), !C_n^{\ell, \ell'}(A, B, M_1), ?R_n^{\ell_C, \ell_R}(B, A, M_2, K)]$. Since the source configuration is balanced and $\text{fresh}(n) \in e_1$, by condition 4, $\text{fresh}(n) \notin e'_1 + e_2$, giving a contradiction.

□

The Safety Theorem exploits the preservation of the balancing conditions under process reduction for proving the safety of processes with empty effect. Notice that we only consider prevalidated processes, where $nonces(P) \cap (msgs(P) \cup plaintexts(P)) = \emptyset$ so that condition 6 holds.

THEOREM 2 (SAFETY) *If $\vdash P : []$, 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 $\langle \varepsilon, P \rangle \rightarrow^* \langle s, A \triangleright end_n(A, B, M_1, M_2, K).R \mid Q \rangle \rightarrow \langle s :: A \triangleright end_n(A, B, M_1, M_2, K), A \triangleright R \mid Q \rangle$ be a semantic derivation leading to an *end* assertion.

The proof is divided in two steps. First, we prove that $begin_n(B, A, M_1, M_2, K) \in s$, thus showing that every $end_n(A, B, M_1, M_2, K)$ in s is preceded by a corresponding begin (Non-Injective Agreement). Next, we also prove that s does not contain actions having the form $end_n(\cdot)$, thus proving that every $end_n(A, B, M_1, M_2, K)$ in s is preceded by a *distinct* $begin_n(B, A, M_1, M_2, K)$ (Agreement).

Non-Injective Agreement By Theorem 5, $\exists e$ s.t. $\vdash A \triangleright end_n(A, B, M_1, M_2, K).R \mid Q : e$.

By an inspection of the validation rules, that judgement is proved by PAR and END. Hence, $fresh(n) \in e$, $!C_n^{\ell_C, \ell_R}(A, B, M_1) \in e$ and $?R_n^{\ell_C, \ell_R}(B, A, M_2, K) \in e$. We proceed by cases according to ℓ'_R .

$\ell'_R = \text{Priv}$ By the balancing condition 7, $s \vdash R_n^{\ell'_C, \ell'_R}(B, A, M_2, K)$. Since such a ciphertext cannot be forged by the environment, we obtain that the action $out(\dots, R_n^{\ell'_C, \ell'_R}(B, A, M_2, K), \dots) \in s$ and, by the balancing condition 10 we get $begin_n(B, A, M'_1, M_2, K) \in s$, $K = k_{BA}$, and $s \vdash C_n^{\ell, \ell'}(A, B, M'_1)$, with $\ell, \ell' \diamond \ell'_C, \ell'_R$. We have to show that $M_1 = M'_1$. If $\ell'_C = \text{Pub}$, then $M_1 = M'_1 = \varepsilon$, as desired. If $\ell'_C \neq \text{Pub}$ then the challenge is not forgeable by the environment: indeed, if $\ell'_C = \text{Tnt}$, then, by the balancing condition 9, $s \not\vdash n$. By the balancing condition 12, $!C_n^{\ell, \ell'}(A, B, M'_1) \in e$. By Lemma 6, $M_1 = M'_1$, as desired.

$\ell'_R = \text{Int}$ By the balancing condition 7, $s \vdash R_n^{\ell'_C, \ell'_R}(B, A, M_2, K)$. Since the ciphertext is not forgeable by the environment, $out(\dots, R_n^{\ell'_C, \ell'_R}(B, A, M_2, K), \dots) \in s$ and, by the balancing condition 10, we get $begin_n(B, A, M'_1, M_2, K) \in s$, $K = k_{BA}$, and $s \vdash C_n^{\ell, \ell'}(A, B, M'_1)$, with $\ell, \ell' \diamond \ell'_C, \ell'_R$. We have to show that $M_1 = M'_1$. If $\ell'_C = \text{Pub}$, then $M_1 = M'_1 = \varepsilon$, as desired. Let us suppose that $\ell'_C \neq \text{Pub}$. By the constraints on security levels imposed by END, $\ell'_C \in \{\text{Priv}, \text{Int}\}$. Thus the ciphertext is not forgeable by the environment and, by the balancing condition 12, $!C_n^{\ell, \ell'}(A, B, M'_1) \in e$. By Lemma 6, $M_1 = M'_1$, as desired.

$\ell'_R = \text{Tnt}$ By the constraints on security levels imposed by END, $\ell'_C \in \{\text{Priv}, \text{Tnt}\}$. By the balancing condition 9, $s \not\vdash n$. By the balancing condition 7, $s \vdash R_n^{\ell'_C, \ell'_R}(B, A, M_2, K)$. Since the ciphertext is not forgeable by the environment, $out(\dots, R_n^{\ell'_C, \ell'_R}(B, A, M_2, K), \dots) \in s$ and, by the balancing condition 10, we get $begin_n(B, A, M'_1, M_2, K) \in s$, $K = k_{BA}$, and $s \vdash C_n^{\ell, \ell'}(A, B, M'_1)$,

Table 14 Encryption Abstraction

A partial function $f : \{\!| T \!\!|_K \mid \text{MAC}_K(T)\} \mapsto T_1, \dots, T_n$, with $T_i = C/R_{N_i}^{\ell, \ell'}(I, J, M_i, K_i)$ is an *encryption abstraction* iff

- Format*
- $\text{names}(T) \subseteq I\mathcal{D}$ and $\text{names}(N_i) = \text{names}(M_i) = \text{names}(K_i) = \emptyset$;
 - $x \in \text{vars}(T)$ iff $x \in \text{vars}(N_i) \cup \text{vars}(M_i) \cup \text{vars}(K_i)$, for some i ;
 - $\forall x, y \in \bigcup_{i \in [1, n]} \text{vars}(M_i)$, x precedes y in T iff x precedes y in the tuple M_1, \dots, M_n .
 - T does not contain input variables

Unique Abstraction As in Table 6.

Nesting As in Table 6.

Encryption If $f(\{\!| T \!\!|_K) = T_1, \dots, T_n$ or $f(\text{MAC}_K(T)) = T_1, \dots, T_n$ then for every $i \in [1, n]$.

- $K = k_I^+$ implies $T_i \in \{C_N^{\ell, \ell'}(J, I, M_i, K_i), R_N^{\ell', \ell}(J, I, M_i, K_i)\}$, with $I \leq \text{Tnt}$;
 - $K = k_I^-$ implies $T_i \in \{C_N^{\ell, \ell'}(I, J, M_i, K_i), R_N^{\ell', \ell}(I, J, M_i, K_i)\}$, with $I \leq \text{Int}$;
 - $K \in \{k_{IJ}, x_{IJ}\}$ implies $T_i \in \{C/R_N^{\ell, \ell'}(I, J, M_i, K_i), C/R_N^{\ell', \ell}(J, I, M_i, K_i)\}$;
-

with $\ell, \ell' \diamond \ell'_C, \ell'_R$. We have to show that $M_1 = M'_1$. Since $s \neq n$, the challenge is not forgeable by the environment and, by the balancing condition 12, $!C_n^{\ell, \ell'}(A, B, M'_1) \in e$. By Lemma 6, $M_1 = M'_1$, as desired.

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

Agreement The balancing condition 4 proves that $\text{end}_n(\cdot) \notin s$, as desired.

□

Finally, notice that Lemma 1 is a trivial corollary of Theorem 5.

D Extension to Mutual Authentication

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

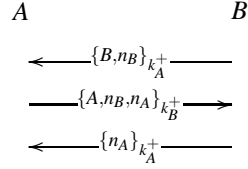
Table 14 extends the encryption abstraction by considering ciphertexts that are both challenges and responses. The modifications affect conditions *Format* and *Nesting* and simply amount to propagate the original condition to all the challenge-response messages in the abstraction of the ciphertext. The modification required in the proof are marginal and harmless. We leave them to the interested reader.

E Examples

In this section we present some examples to clarify the abstraction and the analysis.

E.1 Needham-Schroeder-Lowe Public-Key Authentication Protocol

The Needham-Schroeder-Lowe public-key authentication protocol is reported below:



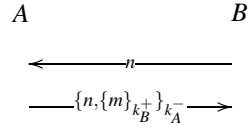
The encryption abstraction for the three ciphertexts may be defined as follows:

$$f(T) = \begin{cases} C_x^{\text{Tnt}, \text{Tnt}}(B, A) & \text{if } T = \{B, x\}_{k_A^+} \\ (R_x^{\text{Tnt}, \text{Tnt}}(A, B), C_y^{\text{Tnt}, \text{Pub}}(A, B)) & \text{if } T = \{A, x, y\}_{k_B^+} \\ R_x^{\text{Tnt}, \text{Pub}}(B, A) & \text{if } T = \{x\}_{k_A^+} \\ \uparrow & \text{otherwise} \end{cases}$$

Notice that the third ciphertext is abstracted into an untrusted response: this respects condition *Encryption* which requires $\ell \leq \text{Tnt}$. Indeed, from an authentication point of view, this ciphertext does not provide any guarantee and might in principle be sent in clear, thus simplifying the protocol. The reason why the nonce is encrypted is that the two nonces are intended to be used later on to generate a secret session-key shared between A and B and thus their privacy has to be guaranteed. We remark that the structure of the three ciphertexts is different and thus the use of tags is not required: in fact, f is an encryption abstraction. We leave to the interested reader the definition of the ρ -spi process narration and the validation of its abstraction.

E.2 A simple nested protocol

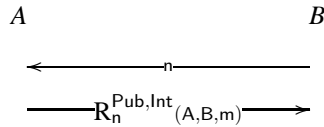
To illustrate the use of nested encryptions, let us consider the following protocol:



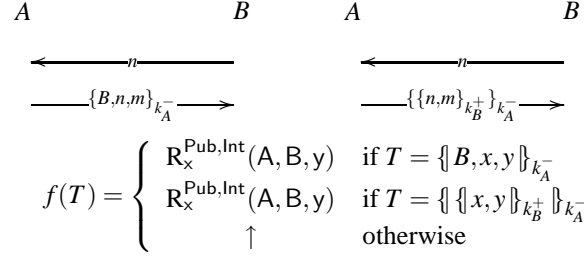
The nonce n is sent as plaintext in the challenge while the response is signed with A 's private key, thus being integer. The authenticated message m is encrypted with B 's public-key, thus simultaneously protecting the privacy of m and specifying the intended verifier. We can define the encryption abstraction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

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

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



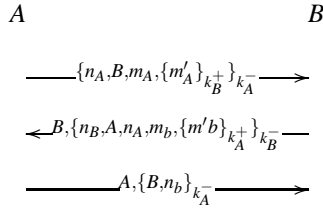
Notice that this CR protocol actually abstracts several ρ -spi protocols, besides the one previously reported. For instance, some other possible instances are depicted below along with the corresponding encryption abstraction:



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

E.3 Ban modified version of CCITT X.509

The BAN modified version of CCITT X.509 is a mutual authentication protocol based on public-key cryptography and digital signature. The protocol is depicted below:



In the first message exchange, A sends two messages to B: m_A is just signed and can be thus read by the attacker while m'_A is encrypted with B's public-key and is thus kept secret. Similar reasoning applies to messages m_B and m'_B in the second message exchange. The encryption abstraction for the above mentioned protocol is reported below:

$$f(T) = \begin{cases} \mathbf{C}_x^{\text{Int,Int}}(A, B, y, z) & \text{if } T = \{x, B, y, \{z\}_{k_B^+}\}_{k_A^-} \\ (\mathbf{C}_x^{\text{Int,Int}}(B, A, y, z), \mathbf{R}_w^{\text{Int,Int}}(B, A)) & \text{if } T = \{x, A, w, y, \{z\}_{k_A^+}\}_{k_B^-} \\ \mathbf{R}_x^{\text{Int,Int}}(A, B) & \text{if } T = \{B, x\}_{k_A^-} \\ \uparrow & \text{otherwise} \end{cases}$$

Notice that nested encryptions is abstracted away since it does not contribute to the authentication goal. Such encryptions do guarantee the secrecy of the authenticated messages occurring therein, but this property is not captured by our analysis since it focuses on the security level of challenges and responses and not on the one of their single components. As future work, we plan to investigate a refinement of the abstraction allowing us to precisely characterize the security level of single messages.

E.4 A unique abstraction for tagged ciphertexts

As shown in [18], the compositionality of security protocols can be enforced by tagging message components. The idea is to rely on a set of tags that is expressive enough for characterizing the role of message components in the authentication task. For instance, the following untagged ciphertext

$$\{A, m, n\}_{k_{AB}}$$

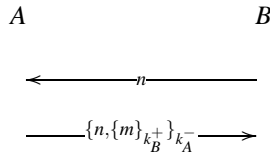
does not contain any information describing its role: it might be a challenge sent from A to B as well as a response sent from B to A . To clarify the role of such a ciphertext we can rely on tag Id for the identity label, Auth for the authenticated message and Verif_{PC} for the nonce. The latter tag says that (i) the identity label A is the verifier of the authentication session and (ii) the ciphertext is a response in a PC (Plain-Cipher) handshake. The type and effect system of [18] contains a typing rule of the following form:

$$\frac{\text{PC VERIF} \quad \Gamma \vdash M : \text{Un} \quad \Gamma \vdash N : \text{Un} \quad \Gamma \vdash A : \text{Un} \quad \Gamma \vdash _ : \text{Un}}{\Gamma \vdash \{\text{Id}(A), \text{Auth}(m), \text{Verif}_{\text{PC}}(n), _\}_{k_{AB}} : \text{Enc}(C_N^{\text{Priv}}(B, A, M))}$$

Here the order of terms within the ciphertext is immaterial. We can easily map such a typing rule into a specific encryption abstraction as follows:

$$f(T) = \begin{cases} C_x^{\text{Pub,Priv}}(B, A, y, \tilde{z}) & \text{if } T = \{\{\text{Id}(A), \text{Auth}(y), \text{Verif}_{\text{PC}}(x), \tilde{z}\}\}_{k_{AB}} \\ \uparrow & \text{otherwise} \end{cases}$$

Notice that the challenge in a PC handshake is of level Pub . In the rule above, \tilde{z} denotes an arbitrary tuple of variables and we implicitly assume that the order of message components is immaterial, as well as in the type and effect system. We can similarly translate the other typing rules for ciphertexts of [18], thus covering the same protocol class. We remark that the analysis here proposed is strictly more general than the type and effect system in [18]. For instance, the type and effect system does not cover session-key distribution and authentication protocols. Furthermore, nested encryptions are generated and analyzed independently as if they were not nested. This way we could not analyze protocols as the following one:



The nested ciphertext does not only protect message m but also denotes, via the encryption key, the intended receiver. This sophisticated reasoning couldn't be captured by our previous type and effect system. However, we can easily define an encryption abstraction expressing the role of the ciphertext as follows:

$$f(T) = \begin{cases} R_x^{\text{Pub,Int}}(A, B, y) & \text{if } T = \{x, \{y\}_{k_B^+}\}_{k_A^-} \\ \uparrow & \text{otherwise} \end{cases}$$

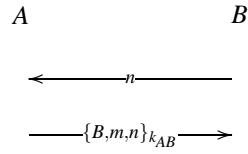
Notice that the public-key k_B^+ “binds” the occurrence of B in the abstract response term, as desired.

Table 15 Attack on the multi-protocol system and fix

Attack	Fixed protocols
$ \begin{array}{ccc} A & \xrightarrow{n} & B \\ \leftarrow -n- & & \\ \{B,m,n\}_{k_{AB}} & & \\ \leftarrow \{B,m,n\}_{k_{AB}} & & \end{array} $	$ \begin{array}{ccc} A & \xleftarrow{n} & B \\ \leftarrow \{\text{Verif}(B),m,n\}_{k_{AB}} & & \\ \\ A & \xrightarrow{n} & B \\ \leftarrow \{\text{Claim}(B),m,n\}_{k_{AB}} & & \end{array} $

E.5 Tagging solves ambiguities

Let us consider again a very simple PC protocol:



Let us define the encryption abstraction as $f = \{\{B, z, x\}_{k_{AB}}\} \mapsto \mathbb{R}_x^{\text{Pub,Priv}}(A, B, z)$. The abstract narration resulting from the protocol abstraction is reported below.

$$\begin{array}{l}
 !A \triangleright \text{in}(x).\text{new}(m).\text{begin}_x(A, B, _, m).\text{out}(\mathbb{R}_x^{\text{Pub,Priv}}(A, B, m)) \\
 | !B \triangleright \text{new}(n).\text{out}(n).\text{in}(\mathbb{R}_n^{\text{Pub,Priv}}(A, B, y)).\text{end}_n(B, A, _, y)
 \end{array}$$

Suppose now that we want to abstract a second protocol, similar to the previous one but with A instead of B as identity label inside the ciphertext. The protocol and the encryption abstraction for the ciphertext are reported below:

$$\begin{array}{ccc}
 \xleftarrow{n} & & \\
 \leftarrow \{B, m, n\}_{k_{AB}} & & g = \{\{B, z, x\}_{k_{AB}}\} \mapsto \mathbb{R}_x^{\text{Pub,Priv}}(B, A, z)
 \end{array}$$

Unfortunately, f and g cannot be combined together as they map the same encryption to two different abstract messages: $f(\{\{B, z, x\}_{k_{AB}}\}) = \mathbb{R}_x^{\text{Pub,Priv}}(A, B, z)$ and $g(\{\{B, z, x\}_{k_{AB}}\}) = \mathbb{R}_x^{\text{Pub,Priv}}(B, A, z)$, thus violating property *Unique Abstraction* in Table 6. As a matter of fact, the two protocols are safe as far as they are not put in parallel composition. Indeed, their concurrent execution gives rise to the standard *reflection* attack depicted on the left side of Table 15. A is running both the first protocol (\rightarrow) as responder and the second one (\leftarrow) as initiator: at the end of the session, she authenticates B even if B is not present at all in the authentication session. The problem is that A generates in the first protocol a ciphertext that the enemy may reply to A , by impersonating B running the second protocol. The problem can be fixed, and the attack prevented, by the use of tags telling the claimant from the verifier, as shown in Table 15. The use of tags makes the two ciphertexts syntactically different and, consequently, the union of the two “fixed” functions f_{fix} and g_{fix} (reported below) is still a function and, in fact, it is an encryption abstraction.

$$\begin{array}{l}
 f_{fix} = \{\text{Id}(B), \text{Auth}(z), \text{Verif}_{\text{PC}}(x)\}_{k_{AB}} \mapsto \mathbb{R}_x^{\text{Pub,Priv}}(A, B, z), \\
 g_{fix} = \{\text{Id}(B), \text{Auth}(z), \text{Claim}_{\text{PC}}(x)\}_{k_{AB}} \mapsto \mathbb{R}_x^{\text{Pub,Priv}}(B, A, z),
 \end{array}$$