A Calculus of Challenges and Responses

Michael Backes Matteo Maffei
Saarland University
Computer Science Department
{backes,maffei}@cs.uni-sb.de

Agostino Cortesi Riccardo Focardi
Ca' Foscari University
Computer Science Department
{cortesi,focardi}@dsi.unive.it

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 process 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 additional protocol classes enjoy a soundness theorem provided that these extensions satisfy certain explicit, easily checkable conditions.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Verification*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages —*Program Analysis*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Security, Verification

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, 23, 19, 12, 21, 10, 14, 6]). 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 [21, 10, 14] 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 [10] and the ρ-spi calculus [12]. 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 [21] 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 [14] relies on some dynamic information attached to ciphertexts, in the form of tags, which uniquely determine the role of messages in the authentication task. An advantage is that tag and type definitions are automatically inferred [26]. 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 focusing on the challenge-response components that are inherent in the protocols. The resulting

FMSE'07, November 2, 2007, Fairfax, Virginia, USA.

abstractions are formalized in a new process calculus, the CR *calculus*, which is tailored to reasoning about challenge-response mechanisms within authentication protocols.

The abstraction enables more abstract and general proofs, and enjoys some properties constituting a significant advantage over existing type-based approaches. A soundness theorem states that abstractions preserve authentication properties, i.e., proving authentication for a CR protocol abstraction 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 entails that refining the abstraction does not affect the soundness of the analysis and viceversa. The abstraction scheme is extensible in that extensions to additional protocol classes immediately enjoy a soundness theorem, given that these extensions satisfy certain explicit, easily checkable conditions. This is a key difference with respect to other type-based approaches [2, 21, 14], where an extension of the analysis requires the soundness and the safety to be proved from scratch.

Further related work. CPSA (Cryptographic Protocol Shape Analyzer) [17], a tool based on strand spaces [23], and ProVerif [7, 3, 8] 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 [27], 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 [22]. 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 validated protocols is still safe, possibly relying on message tags. As opposed to our approach, both CPSA and ProVerif do not enjoy guaranteed termination, although the analysis terminates for a large class of protocols [9].

The control-flow analysis for message authenticity proposed by Bodei *et al.* in [10] and recently extended in [20] 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 relies on an abstraction of syntactic cryptographic patterns into challenge-response components that suffices to guarantee the safety of the protocol and, notably, to ensure the compositionality of the analysis. The control flow analysis of [10, 20] 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.* [19, 16, 15] 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 [22], 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 CR protocol narrations. Section 5 proposes an effect system for checking the safety of CR protocols. Section 6 concludes and outlines future work.

2. A DIALECT OF ρ-SPI CALCULUS

The ρ-spi calculus [12] derives from the spi calculus [4] and inherits many of the features of *Lysa* [10], a dialect of the spi calculus specifically tailored to the analysis of authentication protocols. The ρ-spi calculus differs from both calculi in several respects: it incorporates the notion of tagged message exchange, 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 deal with hash functions, message authentication codes (MACs) and session keys.

2.1 Syntax

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 $\mathcal{ID}$, ranged over by I and J, is further partitioned into *trusted principals* $\mathcal{ID}_\mathcal{P}$, ranged over by A and B, and *enemies* $\mathcal{ID}_\mathcal{E}$, ranged over by E. *Keys* are partitioned into symmetric long-term keys k_{IJ}, shared between I and J, 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 '$\mathsf{new}(k_{IJ})$'; session keys, instead, are always freshly generated through the command '$\mathsf{new}(k_{IJ})$'. 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,

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

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

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

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 $\mathsf{Tag} \in \mathcal{T}$, thus determining their role in the authentication task. Moreover, terms contain pairs [1], encryptions, hashes and MACs. The special name _ denotes the empty message and it will be omitted when occurring at the end of a tuple, e.g., $\mathsf{begin}_N(A, B, M_1, M_2, \_)$ will be written as $\mathsf{begin}_N(A, B, M_1, M_2)$. Such a name allows us to model protocols that, for instance, do not authenticate either messages or session keys but just mutually authenticate the agents on some nonces. In the rest of the paper, we write $nm(T)$, $var(T)$, and $term(T)$ to denote the set of names, variables and subterms of T, respectively.

Processes (or *protocols*), ranged over by P and Q, behave as follows: $\mathsf{new}(n).P$ generates a fresh name n and $\mathsf{new}(k_{IJ}).P$ generates a fresh session key k_{IJ}, which is intended to be shared between I and J. In both cases, the restriction is a binder whose scope is the continuation process P. We presuppose a unique anonymous public channel, the network, from/to which all principals, including intruders, read and send messages. Similarly to *Lysa* and other pattern-matching process calculi [29, 24], 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 $\mathsf{in}(?x, ?y).P$ 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 $\mathsf{in}(?x, x).P$ 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 $\mathsf{in}(\{\!\!\{ ?x \}\!\!\}_{k^-_A}).P$ 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 introduced in Section 2.2. Note that we distinguish between the *static* cryptographic terms T of the calculus, and the actual sent and received *run-time* terms G. Encryption, hashing and MAC generation for terms is denoted $\{\!\!\{ T \}\!\!\}_k, h(\!(T)\!)$ and $\mathrm{MAC}_k(\!(T)\!)$, respectively, while encrypted messages, hashes and MACs are denoted $\{G\}_k, h(G)$ and $\mathrm{MAC}_k(G)$, 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, $\mathsf{in}(?x).P$ and $\mathsf{in}(\{\!\!\{ ?y \}\!\!\}_{k_{AB}}).Q$ can both read message $\{G\}_{k_{AB}}$ but the first process just reads it, denoted $in(\{G\}_{k_{AB}})$, while the second one reads and decrypts it, denoted $in(\{\!\!\{ G \}\!\!\}_{k_{AB}})$. In particular, x is bound to $\{G\}_{k_{AB}}$ while y is bound to G. This makes it possible to express protocols where part of a message is unknown to the recipient, as also done, e.g., in symbolic model checking and constraint solving [5, 11, 28]. Finally, we remark that during protocol execution output terms may contain both encryption patterns and run-time terms, resulting from variable instantiation. We let R range over such terms and write $[\![R]\!]$ to denote the run-time term obtained from R by replacing all the occurrences of $\{\!\!\{ \ldots \}\!\!\}_k$, $\mathrm{MAC}_k(\!(\ldots)\!)$, and $h(\!(\ldots)\!)$ into $\{\ldots\}_k$, $\mathrm{MAC}_k(\ldots)$, and $h(\ldots)$, respectively.

We use the two primitives $\mathsf{begin}_N(A, B, M_1, M_2, K)$ and $\mathsf{end}_N(B, A, M_1, M_2, K)$ to check the *correspondence assertions* [30] in a nonce handshake between A and B based on nonce N. The former declares that A is willing to authenticate with B, while the latter declares that B is authenticating A. The terms M_1 and M_2, when specified, are messages sent by B to A in the challenge and by A to B in the response, respectively; K is a session key sent by A to B. Process $A \triangleright P$ 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

[1] For the sake of readability, in the rest of the paper we often omit brackets: for instance, the nested pair $(a, (b, k))$ is simplified in a, b, k.

Table 2 Example protocol in ρ-spi

$\mathit{Resp} \triangleq$		$\mathit{Init} \triangleq$
		$\mathsf{new}(m).\mathsf{new}(n_B).$
$\mathsf{in}(?x, \{\!\|\, B, ?y \,\|\!\}_{k_A^-}).$	$\xleftarrow{\quad m,\{B,n_B\}_{k_A^+} \quad}$	$\mathsf{out}(m, \{\!\|\, B, n_B \,\|\!\}_{k_A^+}).$
$\mathsf{new}(n_A).\mathsf{new}(k_{AB}).$		
$\mathsf{begin}_y(A, B, \_, \_, k_{AB}).$		
$\mathsf{out}(\{\!\|\, y, k_{AB}, A \,\|\!\}_{k_B^+}, n_A).$	$\xrightarrow{\quad \{n_B,k_{AB},A\}_{k_B^+},n_A \quad}$	$\mathsf{in}(\{\!\|\, n_B, ?x_{AB}, A \,\|\!\}_{k_B^-}, ?y).$
		$\mathsf{end}_{n_B}(B, A, \_, \_, x_{AB}).$
		$\mathsf{begin}_y(B, A, \_, m).$
$\mathsf{in}(\{\!\|\, h(\!\|\, x \,\|\!), n_A, A \,\|\!\}_{k_{AB}}).$	$\xleftarrow{\quad \{h(m),n_A,A\}_{k_{AB}} \quad}$	$\mathsf{out}(\{\!\|\, h(\!\|\, m \,\|\!), y, A \,\|\!\}_{x_{AB}}).$
$\mathsf{end}_{n_A}(A, B, \_, x).$		

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

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 $\mathsf{end}_n(J, I, M_1, M_2, x_{IJ})$. 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. Consider the mutual authentication protocol reported in Table 2 along with the ρ-spi specification. The goal of the protocol is to allow B 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 by replying with a ciphertext encrypted by B's public key and containing the nonce n_B, a fresh session-key k_{AB} and A's identifier. Additionally, A sends B a fresh nonce n_A. 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 and checking the freshness of n_A, A authenticates B and message m. In the ρ-spi protocol specification, notice that B authenticates the session key received from A in the second message ($\mathsf{end}_{n_B}(B, A, \_, \_, x)$) and, before sending the third message, he declares his intention to authenticate m with A ($\mathsf{begin}_y(B, A, \_, m)$). Dually, A starts the protocol to exchange an authenticated session key k_{AB} with B ($\mathsf{begin}_y(A, B, \_, \_, k_{AB})$) and then authenticates the message received from B ($\mathsf{end}_{n_A}(A, B, \_, x)$). □

2.2 Operational Semantics

We define the operational semantics of our dialect of ρ-spi in terms of *traces*, following [11]. A trace is a possible sequence of *actions* performed by a process. Each process primitive has an associated action and we denote with *Act* the set of all possible actions. The dynamics of the calculus is formalized by means of a transition relation between *configurations*, i.e., pairs $\langle s, P \rangle$, where $s \in \mathit{Act}^+$ is a trace and P is a closed process, namely a process without free variables. In the following, we let ϵ denote the empty trace. Each transition $\langle s, P \rangle \rightarrow \langle s :: \tau, P' \rangle$ simulates one computation step in P and records the corresponding action τ 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 [18]: 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 asymmetric public keys, and every symmetric and asymmetric private key with E in the subscript, e.g., k_{EI} and k_E^-. Moreover, the environment can tag and untag messages, construct and destruct pairs, apply hash functions, and, if the appropriate key is known, encrypt/decrypt messages and build MACs.

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

The notion of safety extends the *agreement* property of [30, 25] by pointing out the nonce used in the handshake and the role of authenticated messages[2], as discussed in Section 2.1.

Definition 2 (Trace Safety). *A trace s is safe iff whenever $s = s_1 :: \mathit{end}_n(A, B, G) :: s_2$, there exist s_1', s_1'' such that $s_1 = s_1' :: \mathit{begin}_n(B, A, G) :: s_1''$ and $s_1' :: s_1'' :: s_2$ is safe. P is safe iff s is safe for all $s \in \mathit{Tr}(P)$.*

Intuitively, this definition guarantees that, whenever B authenticates A, A has been willing to authenticate with B and the two principals agree on the messages G exchanged in the handshake.

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. The idea is to abstract

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

Table 3 CR Calculus Names and Terms

$\mathsf{a} ::=$	n, m	(Msg)	$\mathsf{T} ::=$	a	(Name)
	k_{IJ}	(Session Key)		$\mathsf{x}, \mathsf{y}, \mathsf{z}$	(Var)
	$\mathsf{I}, \mathsf{J}, \mathsf{A}, \mathsf{B}, \mathsf{E}$	(Id)		$?\mathsf{x}, ?\mathsf{y}, ?\mathsf{z}$	(Input Var)
	$\top$	(Any)		$(\mathsf{T}_1, \mathsf{T}_2)$	(Pair)
	$\bot$	(Failure)		$\mathrm{C}_{\mathsf{N}}^{\ell,\ell'}(\mathsf{I}, \mathsf{J}, \mathsf{M}, \mathsf{K})$	(Chal)
				$\mathrm{R}_{\mathsf{N}}^{\ell',\ell}(\mathsf{I}, \mathsf{J}, \mathsf{M}, \mathsf{K})$	(Resp)

Notation: $\ell \in \{\mathsf{Pub}, \mathsf{Tnt}, \mathsf{Priv}, \mathsf{Int}\}$.
M, N denote terms with no challenges and responses. K is either a session key k_{IJ} or a key variable x_{IJ}.

cryptographic challenges or responses into two special messages, namely $\mathrm{C}_N^{\ell,\ell'}(B, A, M_1, K_1)$ and $\mathrm{R}_N^{\ell,\ell'}(A, B, M_2, K_2)$. The former may be read as "challenge sent by B to A to authenticate A reception of message M_1 and session-key K_1, using nonce N" and the latter may be read as "response sent by A to B to authenticate that M_2 and K_2 originate from A, using nonce N". The labels $\ell, \ell' \in \{\mathsf{Pub}, \mathsf{Tnt}, \mathsf{Int}, \mathsf{Priv}\}$ specify the security level of the challenge and the response, respectively, with the following meaning:

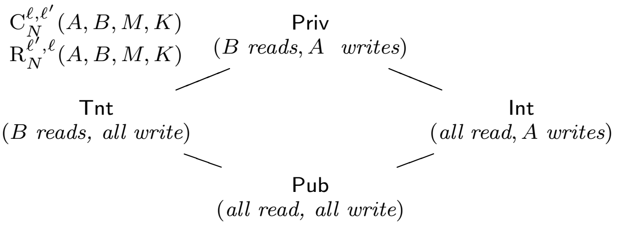
Pub **:** readable and writable by everyone; plaintexts fall in this class;

Tnt **:** writable by everyone but readable only by the intended receiver (e.g., public key cryptography);

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 (e.g., "authenticated symmetric encryption" as soon as the sender and the intended receiver are made explicit).

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

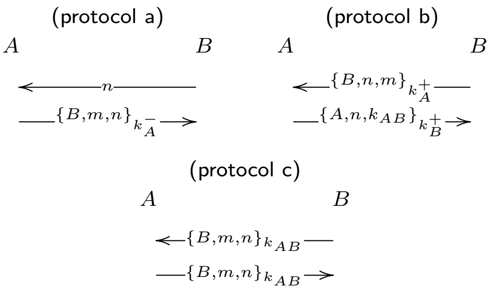


Messages with $\ell \leq \mathsf{Tnt}$ may be written by everyone, while messages with $\ell \leq \mathsf{Int}$ may be read by everyone. Moreover, only B can read messages with $\ell \geq \mathsf{Tnt}$ and only A can write messages with $\ell \geq \mathsf{Int}$. There are two ways in which a principal 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, the term $\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A}, \mathsf{B}, \mathsf{m})$ represents an integer response that may be preceded by the public challenge $\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B}, \mathsf{A})$, which might be concreted as in protocol a (cf. Table 4): A proves her identity by signing the nonce together with B's identifier and message m. As another example, $\mathrm{R}_{\mathsf{n}}^{\mathsf{Tnt},\mathsf{Tnt}}(\mathsf{A}, \mathsf{B}, \_, \mathsf{k}_{\mathsf{AB}})$ represents a tainted response that may be preceded by the tainted challenge $\mathrm{C}_{\mathsf{n}}^{\mathsf{Tnt},\mathsf{Tnt}}(\mathsf{B}, \mathsf{A}, \mathsf{m})$, which might be concreted as in protocol b: A proves her identity by decrypting the tainted challenge using her private key.

Attention should be payed when the security label of the challenge and the response is the same and symmetric key cryptography is used. For example, $\mathrm{R}_{\mathsf{n}}^{\mathsf{Priv},\mathsf{Priv}}(\mathsf{A}, \mathsf{B}, \mathsf{m}_2)$ with challenge $\mathrm{C}_{\mathsf{n}}^{\mathsf{Priv},\mathsf{Priv}}(\mathsf{B}, \mathsf{A}, \mathsf{m}_1)$ should never be concreted using messages that may 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 distinguishes between challenges and responses and the abstraction from ρ-spi to the CR calculus guarantees that they remain distinguishable even when concreted through symbolic cryptography (see Section 4).

Table 4 Protocol concretions



3.1 Syntax and semantics

The calculus of Challenges and Responses, also called CR calculus, has the same syntax as ρ-spi calculus apart from names and terms, reported in Table 3. CR names, noted a, correspond to the names in ρ-spi plus session keys, the symbol $\top$, used for abstracting arbitrary ρ-spi terms, and the symbol $\bot$, denoting failure.

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. As the ρ-spi calculus, the CR calculus comes with a notion of process well-formedness ruling out undesired process behaviors. In particular, (*i*) identity declarations do not nest; (*ii*) the first identity in begin and end assertions refers to the principal running the process; (*iii*) the first (resp. second) identity in output (resp. input) challenges and responses refers to the principal running the process; (*iv*) the failure symbol $\bot$ does not occur in the process; (*v*) 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 (*vi*) 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 po-

Table 5 Example Protocol in CR calculus

$Resp \triangleq$		$Init \triangleq$
		$\mathsf{new(m).new(n_B)}.$
$\mathsf{in}(?\mathsf{x}, \mathrm{C}^{\mathsf{Tnt,Tnt}}_{?\mathsf{y}}(\mathsf{B}, \mathsf{A}))$	$\longleftarrow$	$\mathsf{out}(\mathsf{m}, \mathrm{C}^{\mathsf{Tnt,Tnt}}_{\mathsf{n_B}}(\mathsf{B}, \mathsf{A})).$
$\mathsf{new(n_A).new(k_{AB})}.$		
$\mathsf{begin_y}(\mathsf{A}, \mathsf{B}, \_, \_, \mathsf{k_{AB}}).$		
$\mathsf{out}(\mathrm{R}^{\mathsf{Tnt,Tnt}}_{\mathsf{y}}(\mathsf{A}, \mathsf{B}, \_, \mathsf{k_{AB}}),$		
$\mathrm{C}^{\mathsf{Pub,Priv}}_{\mathsf{n_A}}(\mathsf{A}, \mathsf{B})).$	$\longrightarrow$	$\mathsf{in}(\mathrm{R}^{\mathsf{Tnt,Tnt}}_{\mathsf{n_B}}(\mathsf{A}, \mathsf{B}, \_, ?\mathsf{x_{AB}}),$
		$\mathrm{C}^{\mathsf{Pub,Priv}}_{?\mathsf{y}}(\mathsf{A}, \mathsf{B})).$
		$\mathsf{end_{n_B}}(\mathsf{B}, \mathsf{A}, \_, \_, \mathsf{x_{AB}}).$
		$\mathsf{begin_y}(\mathsf{B}, \mathsf{A}, \_, \mathsf{m}).$
$\mathsf{in}(\mathrm{R}^{\mathsf{Pub,Priv}}_{\mathsf{n_A}}(\mathsf{B}, \mathsf{A}, \mathsf{x})).$	$\longleftarrow$	$\mathsf{out}(\mathrm{R}^{\mathsf{Pub,Priv}}_{\mathsf{y}}(\mathsf{B}, \mathsf{A}, \mathsf{m}))$
$\mathsf{end_{n_A}}(\mathsf{A}, \mathsf{B}, \_, \mathsf{x}).$		
	$System \triangleq !\mathsf{B} \triangleright Init \mid !\mathsf{A} \triangleright Resp$	

sition 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. To illustrate, we report in Table 5 the CR calculus specification corresponding to the protocol of Example 1. As before, we use arrows to point out synchronizing inputs and outputs.

We define the operational semantics of CR calculus in terms of *traces*, similarly to the $\rho-$spi calculus. Here, instead of symbolic cryptographic terms, the intruder can manipulate challenges and responses, provided that security labels are respected: the intruder can forge messages with labels less than or equal to Tnt and read messages with labels less than or equal to Int. For instance, if the intruder knows n and m, then it can generate terms of the form $\mathrm{C}^{\mathsf{Tnt,Tnt}}_{\mathsf{n}}(\mathsf{A}, \mathsf{B}, \mathsf{m})$. As another example, if the intruder knows $\mathrm{C}^{\mathsf{Int,Int}}_{\mathsf{n}}(\mathsf{A}, \mathsf{B}, \mathsf{m})$, then it also knows n and m. Finally, if the intruder gets the knowledge of a session-key $\mathsf{k_{IJ}}$, 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 $\rightarrow_{\mathsf{a}}$ and $\rightarrow^{+}_{\mathsf{a}}$ to denote one-step and multi-step process reduction, respectively.

4. ABSTRACTION

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, yields a CR protocol that is proved to be a faithful abstraction of the former. The abstraction of syntactic terms is reported in Table 6: the most interesting aspect is the abstraction of 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 other 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 that are abstracted into $\top$. Non-atomic terms are terms different from 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 or response, 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 2. If $f(\{\!\!|\, A, x, z \,|\!\!\}_{k_{AB}}) = \mathrm{R}^{\mathsf{Pub,Priv}}_{\mathsf{x}}(\mathsf{A}, \mathsf{B}, \mathsf{z})$ then $\underline{f}(\{\!\!|\, A, n_A, \{n\}_{k_{BC}} \,|\!\!\}_{k_{AB}}) = \mathrm{R}^{\mathsf{Pub,Priv}}_{\mathsf{n_A}}(\mathsf{A}, \mathsf{B}, \top)$, since $\{n\}_{k_{BC}}$ is non-atomic, while $\underline{f}(\{\!\!|\, A, n_A, m \,|\!\!\}_{k_{AB}}) = \mathrm{R}^{\mathsf{Pub,Priv}}_{\mathsf{n_A}}(\mathsf{A}, \mathsf{B}, \mathsf{m})$, given that m is, instead, atomic.

Although we can define a unique function f covering several interesting protocol classes, as discussed in the conclusions, our framework allows the programmer to extend such a function when needed. The abstraction is sound as far as f is an *encryption abstraction.*

DEFINITION 3 (ENCRYPTION ABSTRACTION). *A partial function* $f : \{\!\!|\, T \,|\!\!\}_K \mid \mathrm{MAC}_K(\!|\, T \,|\!) \mapsto \mathrm{C/R}^{\ell,\ell'}_{\mathsf{N}}(\mathsf{I}, \mathsf{J}, \mathsf{M}, \mathsf{K})$ *is an encryption abstraction iff the following conditions hold:*

Format $nm(T) \subseteq \mathcal{ID} \wedge nm(\mathsf{N}) = nm(\mathsf{M}) = nm(\mathsf{K}) = \emptyset;$
$\wedge\ x \in var(T)$ *iff* $\mathsf{x} \in var(\mathsf{N}) \cup var(\mathsf{M}) \cup var(\mathsf{K});$
$\wedge$ *for every* $\mathsf{x}, \mathsf{y} \in var(\mathsf{M})$, *$x$ precedes y in T iff* x *precedes* y *in* M.
$\wedge$ *T does not contain input variables*

Unique Abstraction for every $T, T' \in dom(f)$ and valid σ, σ' such that $\llbracket T \rrbracket \sigma = \llbracket T \rrbracket ' \sigma'$, $T = T'$;

Nesting for every $T' \in dom(f)$, $\{\!\!|\, T \,|\!\!\}_K \in term(T')$ (resp. $\mathrm{MAC}_K(\!|\, T \,|\!) \in$ *$term(T')$), $\alpha(\{\!\!|\, T \,|\!\!\}_K) = \alpha(T)$ (resp.* $\alpha(\mathrm{MAC}_K(\!|\, T \,|\!)) = \alpha(T)$*);*

Enc-MAC for every $\{\!\!|\, T \,|\!\!\}_K \in dom(f)$ (resp. $\mathrm{MAC}_K(\!|\, T \,|\!)$ *$\in dom(f)$)*

if $K = k^{+}_{I}$, then $f(\{\!\!|\, T \,|\!\!\}_K)$ (resp. $f(\mathrm{MAC}_K(\!|\, T \,|\!))$*)* $\in \{\mathrm{C}^{\ell,\ell'}_{\mathsf{N}}(\mathsf{J}, \mathsf{I}, \mathsf{M}, \mathsf{K}), \mathrm{R}^{\ell',\ell}_{\mathsf{N}}(\mathsf{J}, \mathsf{I}, \mathsf{M}, \mathsf{K})\}$, *with* $l \leq \mathsf{Tnt}$*;*
if $K = k^{-}_{I}$, then $f(\{\!\!|\, T \,|\!\!\}_K)$ (resp. $f(\mathrm{MAC}_K(\!|\, T \,|\!))$*)* $\in \{\mathrm{C}^{\ell,\ell'}_{\mathsf{N}}(\mathsf{I}, \mathsf{J}, \mathsf{M}, \mathsf{K}), \mathrm{R}^{\ell',\ell}_{\mathsf{N}}(\mathsf{I}, \mathsf{J}, \mathsf{M}, \mathsf{K})\}$, *with* $l \leq \mathsf{Int}$*;*
if $K \in \{k_{IJ}, x_{IJ}\}$, then $f(\{\!\!|\, T \,|\!\!\}_K)$ (resp. $f(\mathrm{MAC}_K(\!|\, T \,|\!))$*)* $\in \{\mathrm{C}^{\ell,\ell'}_{\mathsf{N}}(\mathsf{I}, \mathsf{J}, \mathsf{M}, \mathsf{K}), \mathrm{R}^{\ell,\ell'}_{\mathsf{N}}(\mathsf{I}, \mathsf{J}, \mathsf{M}, \mathsf{K})$ $\mathrm{C}^{\ell',\ell}_{\mathsf{N}}(\mathsf{J}, \mathsf{I}, \mathsf{M}, \mathsf{K}), \mathrm{R}^{\ell',\ell}_{\mathsf{N}}(\mathsf{J}, \mathsf{I}, \mathsf{M}, \mathsf{K})\}$.

Table 6 Abstraction of syntactic terms

$$\begin{aligned} \alpha(a) &= \mathsf{a} \\ \alpha(x) &= \mathsf{x} \\ \alpha(?x) &= ?\mathsf{x} \end{aligned} \qquad \begin{aligned} \alpha((T_1, T_2)) &= (\alpha(T_1), \alpha(T_2)) \\ \alpha(\mathsf{Tag}(T)) &= \alpha(T) \\ \alpha(h(\!|\, T \,|\!)) &= \alpha(T) \end{aligned}$$

$$\alpha(\{\!|\, T \,|\!\}_K) = \begin{cases} \underline{f}(\{\!|\, T \,|\!\}_K) & \text{if } \{\!|\, T \,|\!\}_K \in \mathit{dom}(\underline{f}) \\ \underline{\alpha}(T) & \text{if } \{\!|\, T \,|\!\}_K \notin \mathit{dom}(\underline{f}) \wedge \\ & \nexists \text{ valid } \sigma \text{ and } R \in \mathit{dom}(\underline{f}) \\ & \text{s.t. } [\![\, \{\!|\, T \,|\!\}_K \,]\!] \sigma = [\![\, R \,]\!] \\ \perp & \text{otherwise} \end{cases} \qquad \alpha(\mathrm{MAC}_K(\!|\, T \,|\!)) = \begin{cases} \underline{f}(\mathrm{MAC}_K(\!|\, T \,|\!)) & \text{if } \mathrm{MAC}_K(\!|\, T \,|\!) \in \mathit{dom}(\underline{f}) \\ \underline{\alpha}(T) & \text{if } \mathrm{MAC}_K(\!|\, T \,|\!) \notin \mathit{dom}(\underline{f}) \wedge \\ & \nexists \text{ valid } \sigma \text{ and } R \in \mathit{dom}(\underline{f}) \\ & \text{s.t. } [\![\, \mathrm{MAC}_K(\!|\, T \,|\!) \,]\!] \sigma = [\![\, R \,]\!] \\ \perp & \text{otherwise} \end{cases}$$

Condition *Format* requires that the only names occurring in T are identities and that 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 CR processes nonces and session-keys occur only once in the same abstract term.

Condition *Unique Abstraction* requires that the encryption (and MAC) patterns in $\underline{f}$ 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 [14]. We verify this condition by applying a standard most general unifier.

Condition *Nesting* requires that nested encryptions and MACs represent neither challenges nor responses. This prevents the uncontrolled leakage of challenges and responses. As future work, we plan to design a more precise definition of encryption abstraction so as to identify the security level and role of nested terms and to abstract them accordingly.

Condition *Enc-MAC* requires that 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 determining 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, $\mathrm{C}_{\mathsf{v}}^{\mathsf{Pub},\ell}(\mathsf{I},\mathsf{J})$ or $\mathrm{R}_{\mathsf{v}}^{\ell,\mathsf{Pub}}(\mathsf{I},\mathsf{J})$, 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 3. Let us consider the protocol of Example 1 and formulate the following abstraction function:

$$\begin{aligned} f:\quad & \{\!|\, B, x \,|\!\}_{k_A^+} && \mapsto && \mathrm{C}_{\mathsf{x}}^{\mathsf{Tnt},\mathsf{Tnt}}(\mathsf{B},\mathsf{A}), \\ & \{\!|\, x, y_{AB}, A \,|\!\}_{k_B^+} && \mapsto && \mathrm{R}_{\mathsf{x}}^{\mathsf{Tnt},\mathsf{Tnt}}(\mathsf{A},\mathsf{B},\_,\mathsf{y}_{\mathsf{AB}}), \\ & \{\!|\, h(\!|\, x \,|\!), y, A \,|\!\}_{x_{AB}} && \mapsto && \mathrm{R}_{\mathsf{y}}^{\mathsf{Pub},\mathsf{Priv}}(\mathsf{B},\mathsf{A},\mathsf{x}) \end{aligned}$$

By applying the abstraction on terms defined in Table 6, and in particular the closure of the encryption abstraction defined above, we can abstract the protocol of Table 2 into the protocol of Table 5. It is easy to see that function f satisfies the conditions of Definition 3 and is thus an encryption abstraction. □

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, successfully validated abstract processes always guarantee this property.

Definition 4 (Authentication of key variables). *An abstract process* P *guarantees authentication of key variables iff in every reduction of* P *key variables of the form* x_{IJ} *are only replaced by session keys of the form* k_{IJ}. *A similar definition applies to ρ-spi processes.*

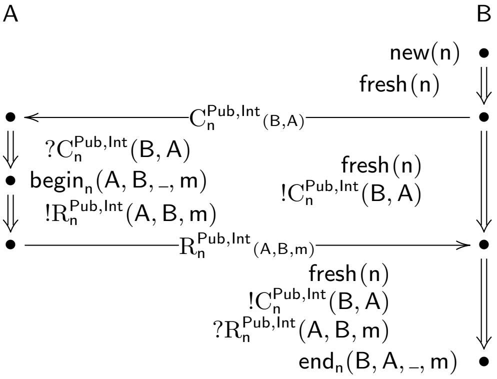
The following theorem states that every concrete computation has a direct counterpart in the abstract model, given that the abstract process guaranteed authentication of key variables.

Theorem 1 (Reduction). *If* $\alpha(P)$ *is well-formed and guarantees authentication of key variables, then* $\langle s, P \rangle \rightarrow^{+} \langle s', P' \rangle$ *implies* $\langle \alpha(s), \alpha(P) \rangle \rightarrow_{\mathsf{a}}^{+} \langle \alpha(s'), \alpha(P') \rangle$.

5. EFFECT SYSTEM

Our use of effects for locally checking the correct behavior of principals is inspired from [14]. Superseding [14] 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 handshake

Table 7 CR protocol with effects


A
B
$\mathsf{new}(\mathsf{n})$
$\mathsf{fresh}(\mathsf{n})$
$\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$
$?\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$
$\mathsf{begin}_{\mathsf{n}}(\mathsf{A},\mathsf{B},\_,\mathsf{m})$
$!\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A},\mathsf{B},\mathsf{m})$
$\mathsf{fresh}(\mathsf{n})$
$!\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$
$\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A},\mathsf{B},\mathsf{m})$
$\mathsf{fresh}(\mathsf{n})$
$!\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$
$?\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A},\mathsf{B},\mathsf{m})$
$\mathsf{end}_{\mathsf{n}}(\mathsf{B},\mathsf{A},\_,\mathsf{m})$

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 $\mathsf{end}_{\mathsf{n}}(\mathsf{A},\mathsf{I},\mathsf{M}_1,\mathsf{M}_2,\mathsf{K})$ only after the output of $\mathrm{C}_{\mathsf{n}}^{\ell,\ell'}(\mathsf{A},\mathsf{I},\mathsf{M}_1)$ (i.e., a challenge with nonce n to authenticate message M_1), the input of $\mathrm{R}_{\mathsf{n}}^{\ell,\ell'}(\mathsf{I},\mathsf{A},\mathsf{M}_2,\mathsf{K})$ (i.e., a response with the same nonce to authenticate message M_2 and session key K) and the check on the freshness of n. Dually, B should respond to challenge $\mathrm{C}_{\mathsf{n}}^{\ell,\ell'}(\mathsf{I},\mathsf{B},\mathsf{M}_1)$ with response $\mathrm{R}_{\mathsf{n}}^{\ell,\ell'}(\mathsf{B},\mathsf{I},\mathsf{M}_2,\mathsf{K})$, only after having asserted $\mathsf{begin}_{\mathsf{n}}(\mathsf{B},\mathsf{I},\mathsf{M}_1,\mathsf{M}_2,\mathsf{K})$. For the sake of simplicity, we assume that session keys circulate only within responses: this is a common way to distribute session keys in that sending them within challenges requires the receiver, and consequently the sender, to engage in a further nonce handshake for verifying the freshness. Although our framework can be extended to deal with such handshakes similarly to [13, 26], this is left as future work.

5.1 Effects and 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, an effect e is a multi-set of atomic effects $f_1,\ldots,f_n$, as formalized below:

$$\begin{array}{l} f ::= \mathsf{fresh}(\mathsf{n}) \mid [!|?]\mathrm{C}_{\mathsf{N}}^{\ell,\ell'}(\mathsf{I},\mathsf{J},\mathsf{M}) \mid [!|?]\mathrm{R}_{\mathsf{N}}^{\ell,\ell'}(\mathsf{I},\mathsf{J},\mathsf{M},\mathsf{K}) \\ e ::= [f_1,\ldots,f_n] \end{array}$$

As mentioned before, we assume that session keys only circulate within responses and, consequently, the last field of the challenge effect is empty. The atomic effect $\mathsf{fresh}(\mathsf{n})$ tracks the freshness of nonce n, allowing for a successive $\mathsf{end}_{\mathsf{n}}(\cdots)$ on the same nonce: a new name n is fresh until it is used in an $\mathsf{end}_{\mathsf{n}}(\cdots)$. The atomic effect $?\mathrm{C}_{\mathsf{N}}^{\ell,\ell'}(\mathsf{I},\mathsf{J},\mathsf{M})$ (resp. $!\mathrm{C}_{\mathsf{N}}^{\ell,\ell'}(\mathsf{I},\mathsf{J},\mathsf{M})$) tracks the reception (resp. generation) of a challenge with nonce N sent by I to J to authenticate message M, thus allowing the occurrence of a successive $\mathsf{begin}_{\mathsf{N}}(\mathsf{J},\mathsf{I},\mathsf{M},\cdots,\cdots)$ (resp. $\mathsf{end}_{\mathsf{N}}(\mathsf{I},\mathsf{J},\mathsf{M},\cdots,\cdots)$) assertion. The security levels ℓ and ℓ' have the same semantics as in Section 3. The atomic effect $?\mathrm{R}_{\mathsf{N}}^{\ell,\ell'}(\mathsf{I},\mathsf{J},\mathsf{M},\mathsf{K})$ tracks the reception of a response with nonce N sent by I to J to authenticate M and session key K, thus allowing a successive $\mathsf{end}_{\mathsf{N}}(\mathsf{J},\mathsf{I},\cdots,\mathsf{M},\mathsf{K})$. The atomic effect $!\mathrm{R}_{\mathsf{N}}^{\ell,\ell'}(\mathsf{I},\mathsf{J},\mathsf{M},\mathsf{K})$ has different semantics as it *enables* I to generate a response for J with nonce N to authenticate M and session key K. This atomic effect is justified by a $\mathsf{begin}_{\mathsf{N}}(\mathsf{I},\mathsf{J},\cdots,\mathsf{M},\mathsf{K})$ assertion. This asymmetry is discussed below. In the following, we illustrate by an example our use of effects in the static analysis of authentication protocols.

Example 4. Let us consider protocol a of Table 4: the corresponding abstract protocol and the effects for the analysis are reported in Table 7. B generates a nonce n, whose freshness is tracked by the atomic effect $\mathsf{fresh}(\mathsf{n})$, and sends it in a challenge to A: $!\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$ tracks on the initiator's code the generation of the challenge. The reception of this message is tracked by $?\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$. The following $\mathsf{begin}_{\mathsf{n}}(\mathsf{A},\mathsf{B},\_,\mathsf{m})$ requires the reception of a challenge from B (effect $?\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$) and justifies the generation of a response to authenticate m (effect $!\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A},\mathsf{B},\mathsf{m})$). This is the reason why effects of the form $!\mathrm{R}_{\mathsf{N}}^{\ell,\ell'}(\ldots)$ are used for *enabling* rather than tracking the generation of responses, as mentioned above. Hence $!\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A},\mathsf{B},\mathsf{m})$ enables A to generate the corresponding response, which is eventually received by B as tracked by $?\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A},\mathsf{B},\mathsf{m})$. After completing the handshake (effects $!\mathrm{C}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{B},\mathsf{A})$ and $?\mathrm{R}_{\mathsf{n}}^{\mathsf{Pub},\mathsf{Int}}(\mathsf{A},\mathsf{B},\mathsf{m})$) and checking the freshness of n (effect $\mathsf{fresh}(\mathsf{n})$), B can assert $\mathsf{end}_{\mathsf{n}}(\mathsf{B},\mathsf{A},\_,\mathsf{m})$.

5.2 Typing Rules

In the following, $+$ and $-$ are the usual union and subtraction operators on multi-sets: $e_1 + e_2$ yields the effect composed of all the atomic effects in e_1 plus the ones in e_2, while $e_1 - e_2$ yields the effect obtained by removing, if present, one occurrence of each atomic effect in e_2 from e_1.

The binding between abstract messages and their effects is formalized in Table 8 by the judgement $\vdash \mathsf{M} : (e_C; e_R)$, 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 that Tnt or the response security level is greater than Int, as discussed in Section 3. The main judgement in our analysis is $\vdash \mathsf{P} : e$, read as "P has effect e", meaning that process P is safe under the conditions expressed by effect e. For instance, $\vdash \mathsf{P} : [\mathsf{fresh}(\mathsf{n})]$ means that P is safe if n is fresh. In the following, we give informal explanations of 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 **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 $[\mathsf{fresh}(\mathsf{n})]$ 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 $\mathsf{fresh}(\mathsf{n})$, at most one use of n as fresh nonce in the continuation process. Indeed, the deletion of one occurrence of $\mathsf{fresh}(\mathsf{n})$ in the thesis allows the continuation process to exploit the nonce for asserting one end assertion (cf. rule End). In justifies in the continuation process the reception

Table 8 Effect System (Terms and Processes)

$$\textsc{Chal}\quad \frac{\ell_C \geq \mathsf{Tnt} \vee \ell_R \geq \mathsf{Int}}{\vdash \mathrm{C}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{I},\mathsf{J},\mathsf{M}) : ([\mathrm{C}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{I},\mathsf{J},\mathsf{M})];[])}$$

$$\textsc{Resp}\quad \frac{\ell_C \geq \mathsf{Tnt} \vee \ell_R \geq \mathsf{Int} \qquad \mathsf{K} \neq \epsilon \Rightarrow \ell_R \geq \mathsf{Tnt}}{\vdash \mathrm{R}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{I},\mathsf{J},\mathsf{M},\mathsf{K}) : ([];[\mathrm{R}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{I},\mathsf{J},\mathsf{M},\mathsf{K})])}$$

$$\textsc{Atom}\quad \vdash \mathsf{a} : ([];[])$$

$$\textsc{Pair}\quad \frac{\vdash \mathsf{T} : (e_C; e_R) \qquad \vdash \mathsf{T}' : (e'_C; e'_R)}{\vdash (\mathsf{T},\mathsf{T}') : (e_C + e'_C; e_R + e'_R)}$$

$$\textsc{Nil}\quad \vdash \mathbf{0} : []$$

$$\textsc{Repl}\quad \frac{\vdash \mathsf{P} : []}{\vdash !\mathsf{P} : []}$$

$$\textsc{Par}\quad \frac{\vdash \mathsf{P} : e_P \qquad \vdash \mathsf{Q} : e_Q}{\vdash \mathsf{P}|\mathsf{Q} : e_P + e_Q}$$

$$\textsc{Id}\quad \frac{\vdash \mathsf{P} : e}{\vdash \mathsf{A} \triangleright \mathsf{P} : e}$$

$$\textsc{New}\quad \frac{\vdash \mathsf{P} : e}{\vdash \mathsf{new}(\mathsf{n}).\mathsf{P} : e - \mathsf{fresh}(\mathsf{n})}$$

$$\textsc{In}\quad \frac{\vdash \mathsf{P} : e + ?e_C + ?e_R \qquad \vdash \mathsf{T} : (e_C; e_R)}{\vdash \mathsf{in}(\mathsf{T}).\mathsf{P} : e}$$

$$\textsc{Out}\quad \frac{\vdash \mathsf{P} : e + !e_C \qquad \vdash \mathsf{T} : (e_C; e_R)}{\vdash \mathsf{out}(\mathsf{T}).\mathsf{P} : e + !e_R}$$

$$\textsc{Begin}\quad \frac{\vdash \mathsf{P} : e + [!\mathrm{R}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{A},\mathsf{I},\mathsf{M}_2,\mathsf{K})] \qquad \mathsf{M}_2 \ \textit{ground}}{\vdash \mathsf{begin}_{\mathsf{N}}(\mathsf{A},\mathsf{I},\mathsf{M}_1,\mathsf{M}_2,\mathsf{K}).\mathsf{P} : e + [?\mathrm{C}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{I},\mathsf{A},\mathsf{M}_1)]}$$

$$\textsc{End}\quad \frac{\vdash \mathsf{P} : e \qquad \mathsf{M}_1, \ \textit{ground}}{\vdash \mathsf{end}_{\mathsf{n}}(\mathsf{A},\mathsf{I},\mathsf{M}_1,\mathsf{M}_2,\mathsf{K}).\mathsf{P} : e + [!\mathrm{C}^{\ell_C,\ell_R}_{\mathsf{n}}(\mathsf{A},\mathsf{I},\mathsf{M}_1), ?\mathrm{R}^{\ell_C,\ell_R}_{\mathsf{n}}(\mathsf{I},\mathsf{A},\mathsf{M}_2,\mathsf{K}), \mathsf{fresh}(\mathsf{n})]}$$

An abstract term is ground if it contains neither variables nor $\top, \bot$ symbols.

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 5.1, the assertion $\mathsf{begin}_{\mathsf{N}}(\mathsf{A},\mathsf{I},\mathsf{M}_1,\mathsf{M}_2,\mathsf{K})$ (rule Begin) requires the reception of $\mathrm{C}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{I},\mathsf{A},\mathsf{M}_1)$ and justifies the generation of $\mathrm{R}^{\ell_C,\ell_R}_{\mathsf{N}}(\mathsf{A},\mathsf{I},\mathsf{M}_2,\mathsf{K})$. Similarly, the assertion $\mathsf{end}_{\mathsf{n}}(\mathsf{A},\mathsf{I},\mathsf{M}_1,\mathsf{M}_2,\mathsf{K})$ (rule End) requires the freshness of n, the generation of $\mathrm{C}^{\ell_C,\ell_R}_{\mathsf{n}}(\mathsf{A},\mathsf{I},\mathsf{M}_1)$ and the reception of $\mathrm{R}^{\ell_C,\ell_R}_{\mathsf{n}}(\mathsf{I},\mathsf{A},\mathsf{M}_2,\mathsf{K})$.

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 $\top$ and $\bot$ never occur within authenticated messages at run time. This is crucial for carrying safety properties from the abstract semantics to the concrete one, since authenticated messages might otherwise be instantiated differently in the matching begin and end assertions. Finally, notice that validation rules uniquely 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 a prevalidation step by labelling end assertions by the security level of challenges and responses based on the same nonce, thus making the analysis fully deterministic.

5.3 Soundness and Safety Results

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

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

The main theorem states that 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 (Soundness). *If* $\alpha(P)$ *is well-formed and* $\vdash \alpha(P) : []$, *then* P *is safe.*

This theorem can be easily proved by exploiting Theorem 1 and the following lemma.

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

For example, the protocol of Table 5 is validated 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* $!\mathsf{P}_1|\ldots|!\mathsf{P}_\mathsf{m}$. *Then* $\vdash \mathsf{P} : []$ *if and only if* $\vdash !\mathsf{P}_\mathsf{i} : [], \forall i \in [1,m]$.

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 safe (*modularity*), and (*ii*) a multiprotocol system composed of safe protocols is safe (*compositionality*). We remark that our result does not prevent the interleaving of protocol executions and, in particular, an execution trace is safe even if an end assertion and the matching begin assertion are generated by principals running different protocols. We do not regard this aspect as limiting or inconvenient since, besides of allowing for strong compositionality results, it is a direct consequence of our approach: in fact, the semantics of principal actions does not depend on the protocol as a whole, but it is instead determined by the semantics of the ciphertexts that principals generate and receive in each message exchange. The encryption abstraction, and in practice the use of tags, makes such semantics formal and explicit, independently of the rest of the protocol.

6. CONCLUSIONS

We have proposed a static analysis of authentication protocols based on an abstraction of cryptography: deriving authentication guarantees for protocol abstractions suffices to prove such guarantees for the actual protocols. The approach enjoys compositionality and guaranteed termination. Furthermore, abstraction refinements are sound as far as they satisfy the conditions on the encryption abstraction.

In [14], we propose a compositional type and effect system, which is general enough to analyze several existing protocols: as a matter of fact, we can formulate a *unique* encryption abstraction comprehending all the protocol classes considered in [14]. This is achieved by tagging messages within ciphertexts so as to make explicit their role in the authentication task (e.g., claimant identifier, nonce, authenticated message, And so on). The abstraction-based analysis here presented is thus strictly more general than our previous type and effect system. As future work, we plan to exploit recent results on linking symbolic cryptography with actual cryptographic algorithms to verify abstract authentication protocols in a way that ensures strong authentication guarantees even for concrete, cryptographic implementations.

7. 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] M. Abadi and A. 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. Vanackére. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2003.

[6] M. Backes, A. Cortesi, and M. Maffei. Causality-based abstraction of multiplicity in cryptographic protocols. In *Proc. 20th IEEE Symposium on Computer Security Foundations (CSF)*, pages 355–369. IEEE Computer Society Press, 2007.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] M. Bugliesi, R. Focardi, and M. Maffei. Dynamic types for authentication, 2007. To appear in Journal of Computer Security.

[15] A. Datta, A. Derek, J.C. Mitchell, and A. Roy. Protocol composition logic (pcl). *Electronic Notes on Theoretical Computer Science*, 172:311–358, 2007.

[16] 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.

[17] 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)*, Lecture Notes in Computer Science, pages 523–538. Springer-Verlag, 2007.

[18] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[19] 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.

[20] 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, 2007.

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

[22] 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.

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

[24] C. Haack and A. Jeffrey. Pattern-matching spi-calculus. *Information and Computation*, 204(8):1195–1263, 2006.

[25] G. Lowe. "A Hierarchy of Authentication Specification". In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE Computer Society Press, 1997.

[26] M. Maffei. *Dynamic Typing for Security Protocols.* PhD thesis, Università Ca' Foscari di Venezia, Dipartimento di Informatica, 2006.

[27] 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.

[28] 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, New York, NY, USA, 2001. ACM Press.

[29] C. R. Nielsen, F. Nielson, and H. R. Nielson. Cryptographic pattern matching. *Electronic Notes on Theoretical Computer Science*, 168:91–107, 2007.

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