Causality-based Abstraction of Multiplicity in Security Protocols

Michael Backes[1], Matteo Maffei[1], and Agostino Cortesi[2]

[1] Saarland University, Saarbrücken, Germany, {backes,maffei}@cs.uni-sb.de
[2] Ca' Foscari University, Venice, Italy, cortesi@dsi.unive.it

Abstract. This paper presents a novel technique for analyzing security protocols based on an abstraction of the program semantics. This technique is based on a novel structure called causal graph which captures the causality among program events within a finite graph. A core property of causal graphs is that they abstract away from the multiplicity of protocol sessions, hence constituting a concise tool for reasoning about an even infinite number of concurrent protocol sessions; deciding security only requires a traversal of the causal graph, thus yielding a decidable, and typically very efficient, approach for security protocol analysis. Additionally, causal graphs allow for dealing with different security properties such as secrecy and authenticity in a uniform manner. Both the construction of the causal graph from a given protocol description and the analysis have been fully automated and tested on several example protocols from the literature.

1 Introduction

Proofs of security protocols are known to be error-prone and, owing to the distributed-system aspects of multiple interleaved protocol runs, awkward to make for humans. In fact, vulnerabilities have accompanied the design of such protocols ever since early authentication protocols like Needham-Schroeder [15, 30], over carefully designed de-facto standards like SSL and PKCS [31, 7], up to current widely deployed products like Microsoft Passport [20]. Formal methods, and in particular language-based techniques, have proved to constitute important tools for dealing with such flaws, by helping both to securely design and to analyze security protocols. A central intricacy that these approaches have to tackle is how to concisely treat the potentially unbounded number of concurrent protocol sessions. Techniques based on state-space exploration rely on the inspection of (abstractions of) all the unbounded sequences of messages exchanged by honest principals, and although such approaches often allow for a careful detection and reproduction of successful attacks, they are inherently constrained by only taking a finite number of sessions into account, or by restricting the analysis to certain classes of protocols, or by giving up guaranteed termination. In contrast to these approaches, static analysis techniques based on type systems rely on the identification of some syntactic patterns that suffice for guaranteeing the property of interest. Furthermore, type systems enjoy guaranteed termination since they work on the syntax of the protocol. The drawback of these techniques is that for analyzing different security properties one has to identify multiple different patterns and, consequently, to develop different type systems for these pattern. Furthermore, the aforementioned patterns are sometimes restrictive in

that, e.g., ensuring freshness of authentication requests presupposes that every authentication session is based on a different nonce (an unguessable random string) that is used only in a single protocol session and then discarded [22, 11]. This rules out some interesting protocols relying on composing multiple authentication sessions based on the same nonce, e.g., see [25].

This paper presents a novel technique for analyzing security protocols based on abstract interpretation of program semantics [13]. In particular, this technique abstracts away from the multiplicity of protocol sessions while still preserving the causality among program events. More precisely, the unbounded number of execution traces of a protocol generated by running an unbounded number of concurrent protocol sessions is concisely abstracted into a novel structure that we call a *causal graph*, i.e., a finite graph in which nodes represent process events and edges express the causality among events. Interestingly, causal graphs allow for soundly characterizing which terms abstract messages generated in the same protocol session and which ones may instead abstract messages generated in different protocol sessions; this information turns out to be crucial in order to determine the safety of protocol specifications. In contrast to state-space exploration techniques, our analysis enjoys guaranteed termination. Furthermore, deciding security requires only a traversal of the causal graph, and causal graphs turn out to be of decent size for commonly analyzed security protocols. Our work improves on existing type-based approaches in that we do not restrict the class of analyzable protocols to the ones adhering to some specific syntactic patterns; this is possible since our analysis is based on an abstraction of the program semantics. Finally, our approach allows for a uniform treatment of different security properties such as secrecy and two different variants of authenticity, namely agreement and injective agreement [26].

We can summarize our technique as follows. We first specify the protocol in a dialect of the spi-calculus [3]. We then construct the corresponding causal graph, which is proven to be unique for every protocol. The causal graph then allows for analyzing the intended security properties. Since causal graphs are of finite size, the analysis is assured to terminate. Finally, the safety of the causal graph implies the safety of the protocol and, more precisely, the safety of the possibly unbounded number of protocol execution traces. As usual for static analysis techniques and due to the undecidability of the original problem, failures in the verification may be caused by either a flaw in the protocol or a non-sufficient precision of the analysis ruling out safe protocols.

Finally, the analysis is amenable to full automation. We have implemented a tool for automating the analysis, and we have applied the tool to some common protocols in the literature [18] (among others, the Needham-Schroeder-Lowe public-key protocol, its fixed version proposed by Lowe, the BAN modified version of CCITT X.509(3) and SPLICE/AS). The analyses terminated within a few seconds and provided safety proofs for the correct versions of the protocols while failing to validate the flawed versions. Remarkably, attacks are often easily derivable by an inspection of the path sets. The only human effort required is to capture the protocol in the dialect of the spi-calculus which is often straightforwardly derivable from the protocol description.

Outline of the paper. Section 2 introduces a dialect of the spi-calculus used for modeling security protocols. Section 3 introduces causal graphs. Section 4 defines the abstract

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

Notation: u ranges over names and variables.

Name			**Process**		
$a ::=$	I, J, A, B, E	(Id)	$P, Q ::=$	$\mathsf{new}(n).P$	(Msg Res)
	n, m, k_{IJ}	(Msg)		$\mathsf{new}^{\mp}(k_I).P$	(Key Res)
	k_I^+	(Pub)		$\mathsf{in}(M).P$	(In)
	k_I^-	(Priv)		$\mathsf{out}(M).P$	(Out)
Term				$\mathsf{begin}_N^i(A, I, M).P$	(Begin)
$M, N ::=$	a	(Name)		$\mathsf{end}_N^i(A, I, M).P$	(End)
	x, y, z	(Var)		$A \triangleright P$	(Principal)
	(M, N)	(Pair)		$P \mid Q$	(Par)
	$\{M\}_u$	(Enc)		$!P$	(Repl)
				$\mathbf{0}$	(Stop)

interpretation framework and states the soundness results. Section 5 presents the safety results. Section 6 discusses further related work and Section 7 concludes.

2 ρ-spi Calculus

The ρ-spi calculus [11, 12] is derived from the spi calculus [3] and inherits many of the features of *Lysa* [8], 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 associates principal identities to processes; it syntactically binds keys to their owners; and it provides new authentication-specific constructs. 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.

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 of *names* partitioned into the set of *messages* $\mathcal{M}$, the set of identities $\mathcal{ID}$, the set of shared keys $\mathcal{K}$ and the set of public and private keys $\mathcal{K}^{\mp}$. The set $\mathcal{ID}$, ranged over by I and J, is further partitioned into the two sets of *trusted principals* $\mathcal{ID}_{\mathcal{P}}$, ranged over by A and B, and *enemies* $\mathcal{ID}_{\mathcal{E}}$, ranged over by E. The set $\mathcal{K}$ is composed of keys k_{IJ} shared between I and J. The set $\mathcal{K}^{\mp}$ is partitioned into public and private keys, noted $\mathcal{K}^{+}$ and $\mathcal{K}^{-}$, respectively. I's key-pair is composed of a public key k_I^+ and a private key k_I^-, related symbolically by k_I. The set $\mathcal{K}_{\mathcal{E}}$ contains the keys of malicious parties, i.e., the ones where some E occurs as subscript. For convenience, we syntactically bind keys to their owners, thus assuming that keys are already distributed among protocol participants. Notice that this simplification is not restrictive since, if

needed, processes can exchange keys thus modelling session-key distribution. Finally, terms can be paired or encrypted with other terms. [3]

Processes (or *protocols*), ranged over by P and Q, behave as follows: $\mathsf{new}(n).P$ generates a fresh name n local to P while $\mathsf{new}^{\mp}(k_I)$ generates a fresh key-pair for I composed of k_I^+ and k_I^-. We presuppose a unique unnamed public channel, the network, from/to which all principals, including intruders, read and send messages. Similarly to *Lysa*, our input primitive may atomically test part of the read message, by employing pattern-matching. If the input term matches the input pattern, then the variables occurring in the pattern are bound to the remaining sub-part of the term; otherwise the term is not read at all. 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 term encrypted with asymmetric cryptography, the correct decryption key has to be specified in the pattern. For giving the intuition of the semantics, which is formally defined in Section 2.2, the process '$\mathsf{in}(n).P$' tries to read a specific name n from the network and, if such a name can be read, no binding occurs and P is executed. This is useful, e.g., to check protocols where nonce n is sent encrypted as challenge and received back in clear as response. As another example, consider '$\mathsf{in}(\{x\}_{k_A^-}).P$'. This process reads any ciphertext of the form $\{a\}_{k_A^+}$, decrypts it on the fly, and binds all the free occurrences of x to a in process P. We remark that the key specified in the input pattern is the *decryption* key since for binding x to a the process has to perform a decryption and thus to know the correct decryption key. For easing the presentation, we only consider the instantiation of variables with names: as in [14, 24], we assume that messages are typed so as to distinguish names from the other terms like pairs and ciphertexts. In the long version of this paper, we extend our approach to additionally allow variables to be instantiated with ciphertexts.[4] The interesting complication arising with such an extension is the capability of the environment to forge arbitrarily nested ciphertexts, thus potentially causing an infinite number of branches in the input. We tackle this problem in the abstract model by guaranteeing that the number of ciphertexts generated by trusted principals is finite and by abstracting away from the ones generated by the environment.

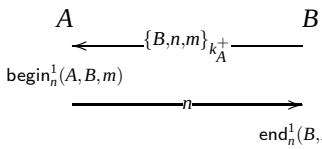
The $\mathsf{begin}_N^i(A,B,M)$ and $\mathsf{end}_N^i(B,A,M)$ primitives express the *correspondence assertions* [33] in a nonce handshake between A and B. The index i allows for specifying which begin assertion should match a specific end assertion: we assume that the same index is used for at most one begin assertion and one end assertion. The former primitive declares that A is starting a protocol session with B, while the latter declares that B is ending a protocol session in which he believes to have correctly authenticated A: N is the nonce used in the protocol session and M is the authenticated message. Finally, $A \triangleright P$ represents principal A executing process P; $P|Q$ is the parallel composition of P and Q; $!P$ indicates an arbitrary number of parallel instances of P, and $\mathbf{0}$ is the null process that does nothing. In the rest of the paper, we will often omit $\mathbf{0}$ from protocol specifications.

Example 1. To illustrate, let us consider the following protocol, in which B encrypts the message m, the nonce n and his own identifier with A's public-key and A acknowledges

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

[4] For convenience, this is also shown in Appendix D.

the reception of the first message by sending back the nonce in clear on the network.



$$\begin{aligned} \mathit{Init} &\triangleq \mathsf{new}(m).\mathsf{new}(n).\mathsf{out}(\{B,n,m\}_{k_A^+}).\mathsf{in}(n).\mathsf{end}_n^1(B,A,m) \\ \mathit{Resp} &\triangleq \mathsf{in}(\{B,x,z\}_{k_A^-}).\mathsf{begin}_x^1(A,B,z).\mathsf{out}(x) \\ \mathit{Prot} &\triangleq \mathsf{new}^{\mp}(k_A).(B \triangleright !\mathit{Init} \mid A \triangleright !\mathit{Resp}) \end{aligned}$$

The begin assertion says that A confirms the reception from B of the message m in a protocol session based on nonce n and, similarly, the end assertion says that B authenticates A receiving m in a protocol session based on n. The goal of this protocol is to guarantee the secrecy of m and strong authenticity between A and B.

2.2 Operational semantics of our dialect of the ρ-spi calculus

Following [9], the ρ-spi calculus comes with a trace-based semantics. 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. In the following, ε denotes the empty trace. Each transition $\langle s,P\rangle \rightarrow \langle s :: t,P'\rangle$ simulates one computation step in P and records the corresponding action t in the trace. We denote by $\rightarrow^*$ a finite sequence of computation steps. In the following, G ranges over ground terms, namely terms containing no variable. 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 [17]: the knowledge of the environment associated with a ρ-spi trace, characterized by the judgement $s \vdash G$, is formalized by a set of deduction rules stating that the environment knows all the messages sent on the network, every message which is not restricted in the trace, all public keys, and enemies' keys. The environment can also construct/destruct pairs and encrypt/decrypt ciphertexts if the appropriate key is known. Formal definitions are postponed to Appendix A.

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

Secrecy and authenticity are defined in terms of ρ-spi traces and processes.

Definition 2 (Secrecy). *A trace s guarantees secrecy of G if and only if* $s \nvdash G$. *A process P guarantees secrecy of G if and only if* $s \nvdash G$ *for all* $s \in T(P)$.

The weak authenticity property refines the standard *agreement* property of [26, 33] by making explicit the nonce used in the handshake and the index binding the begin assertion with the corresponding end one.

Definition 3 (Weak Authenticity). *A trace s guarantees weak authenticity if and only if* $s = s_1 :: \mathit{end}^i_{G_1}(B,A,G_2) :: s_2$ *implies* $\mathit{begin}^i_{G_1}(A,B,G_2) \in s_1$. *A process P guarantees weak authenticity if and only if s guarantees weak authenticity for all* $s \in T(P)$.

Intuitively, this guarantees that whenever B authenticates A and the message G_2 in an handshake with the nonce G_1, then A engaged in a protocol session with nonce G_1 for authenticating G_2 with B. Similarly to the *injective agreement* property of [26, 33], the notion of strong authenticity requires the freshness of authentication requests.

Definition 4 (Strong Authenticity). *A trace s guarantees strong authenticity if and only if whenever* $s = s_1 :: end^i_{G_1}(B,A,G_2) :: s_2$*, we have that* $s_1 = s'_1 :: begin^i_{G_1}(A,B,G_2) :: s''_1$ *and* $s'_1 :: s''_1 :: s_2$ *guarantees strong authenticity. A process P guarantees strong authenticity if and only s guarantees strong authenticity for all* $s \in T(P)$.

2.3 Notational conventions

For easing the presentation of the static analysis technique, we use a number of notational conventions. The message restriction $\mathsf{new}(n).P$ is a binder for the message n, the key-pair restriction $\mathsf{new}^{\mp}(k_I).P$ is a binder for k_I^+ and k_I^-, namely the key-pair based on k_I, and the input primitive is a binder for the variables occurring in the input term with the exception of the decryption keys. In all cases the scope of the binders is the continuation process. Similarly, $new(n)$ is a binder for the message n and $new^{\mp}(k_I)$ is a binder for the key-pair based on k_I and their scope is the continuation trace. The names and variables occurring free and bound in processes and traces are defined as usual. As in companion transition systems, e.g. [10], we implicitly identify processes up to renaming of bound variables and names, i.e., up to α-equivalence. To simplify the definition of the static analysis and following [8], we discipline the α-renaming of bound names. We stipulate that for each message n there is a *canonical representative*, and we demand that two messages are α-convertible only when they have the same canonical representative. We write $\lfloor\!\lfloor n \rfloor\!\rfloor$ to denote the set of messages whose canonical representative is n. A similar assumption applies to key-pairs and variables. We also require that keys, both symmetric and asymmetric ones, having the same canonical representative depend on the same identifiers. We assume that the set $\mathcal{M}_{\mathcal{E}}$ of names generated by the environment (cf. rule ENV in Table 6) has a canonical representative which is different from the ones of the names bound in the process. Finally, we assume that the bound names of a process are renamed apart and that they do not clash with the free names; much in the same way variables are assumed to be all distinct. For convenience and without loss of generality, we shall reason on protocol specifications in which every bound name has a different canonical representative.

3 Causal Graphs

A *causal graph* $\mathscr{C} = (\mathcal{N}, \mathcal{E})$ is a finite directed graph with nodes $\mathcal{N}$ and edges $\mathcal{E}$. In the following, we will often write $nodes(\mathscr{C})$ to denote the set of nodes in $\mathscr{C}$ and $edges(\mathscr{C})$ to denote the set of edges in $\mathscr{C}$. Nodes are divided into *process nodes*, representing abstraction of ρ-spi processes, and *communication nodes*, representing the synchronization among a process performing an input and the processes outputting the terms used by the environment to produce the input term. Edges track the causality among nodes. In the following, we discuss each of these components.

3.1 Nodes and edges

Process nodes, also referred to as abstract processes and ranged over by P and Q, are ρ-spi processes built upon ρ-spi names and the following new name categories:

– *the special name* $\mathscr{E}$: it arises in the terms forged by the environment and abstracts the names generated by the environment, identities, public-keys and attackers' keys;

– *labelled names* $\mathsf{a}_{(\mathsf{x})}$: they are semantically equivalent to names but they allow for tracking variable instantiation. As an example, the instantiation in P of the variable x by the name n yields $\mathsf{P}[\mathsf{n}_{(\mathsf{x})}/\mathsf{x}]$. In fact, labelling is local to sequential processes and does not propagate through concurrent threads.
– *indexed names* $\mathsf{a}^{(\mathsf{out}(\mathsf{G}).\mathsf{P},i)}$, where $\mathsf{out}(\mathsf{G}).\mathsf{P}$ is the node outputting a and i the positional index of a in G: they only occur in the environment's knowledge and give a precise characterization of which names have been used by the environment to construct a certain term and, notably, the place (i.e., node and position) in which they are sent on the network.

For distinguishing ρ-spi terms from the ones used in causal graphs, we write the latter by sanserif fonts. In the following, we refer to the set of possibly labelled and indexed names as abstract names. Furthermore, we let G range over ground terms, M,N over terms possibly containing variables, v over abstract names, u over abstract names and variables and $\mathcal{G}$ over ground term sets. We will write $\lfloor \mathsf{M} \rfloor$ to denote the term obtained from M by label erasure and $\lceil \mathsf{M} \rceil$ to denote the term obtained from M by index erasure.

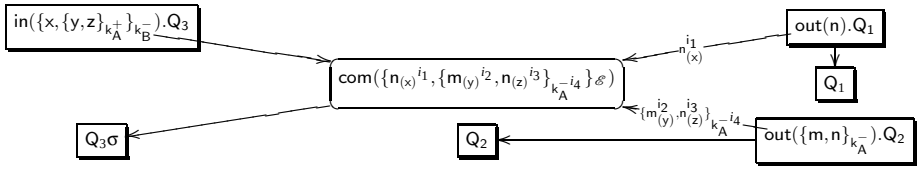
Communication nodes represent the synchronization among a process performing an input with the processes outputting the terms used by the environment to construct the input term. Such nodes have the form $\mathsf{com}(\mathsf{G})$, where G is the input term labelled so as to track the variable assignment induced by pattern-matching (cf. Example 2).

Edges connect two nodes, representing the causality among such nodes. We discern two kinds of causality, namely *intra-thread* and *inter-thread causality*.

Intra-thread causality relates nodes within the same thread and it is represented by edges of the form $\mathsf{P} \rightarrow \mathsf{Q}$, directly linking P to the process Q obtained by reducing P.

Inter-thread causality is due to the synchronization among a process P inputting G and reducing to Q and the nodes $\mathsf{out}(\mathsf{G}_i).\mathsf{P}_i$ outputting the terms used by the environment to construct the input term G. We call such terms integer components since they are not forged by the environment but simply forwarded. They are crucial for the soundness of our abstraction, which identifies unbounded protocol sessions into a finite model: the soundness of the security properties proved on the top of this abstraction requires to determine when different protocol sessions may be interleaved. This may happen when the environment exploits terms output in different protocol sessions to construct a term that is sent to a principal engaging in another protocol session: in fact, messages within the same integer component prove to abstract messages belonging to the same protocol session, while messages in different integer components may abstract messages belonging to different protocol sessions. This kind of causality is represented by (i) edges $\mathsf{in}(\mathsf{M}).\mathsf{P} \rightarrow \mathsf{com}(\mathsf{G})$ and $\mathsf{com}(\mathsf{G}) \rightarrow \mathsf{Q}$, connecting the input process with the communication node and this node with the process obtained by reducing the input; and (ii) edges of the form $\mathsf{out}(\mathsf{G}_i) \stackrel{\mathsf{G}'_i}{\rightarrow} \mathsf{com}(\mathsf{G})$, one for each integer component G'_i of the input term. These edges are labelled by the integer component and connect the process outputting such a component to the communication node.

Example 2. Let $\mathscr{C}$ be a causal graph containing the nodes $\mathsf{out}(\mathsf{n}).\mathsf{Q}_1$, $\mathsf{out}(\{\mathsf{m},\mathsf{n}\}_{\mathsf{k}_\mathsf{A}^-}).\mathsf{Q}_2$ and $\mathsf{in}(\{\mathsf{x},\{\mathsf{y},\mathsf{z}\}_{\mathsf{k}_\mathsf{A}^+}\}_{\mathsf{k}_\mathsf{B}^-}).\mathsf{Q}_3$. Let us define the following indexes: $i_1 = (\mathsf{out}(\mathsf{n}).\mathsf{Q}_1, 1)$, $i_2 = \mathsf{out}(\{\mathsf{m},\mathsf{n}\}_{\mathsf{k}_\mathsf{A}^-}).\mathsf{Q}_2, 1)$, $i_3 = \mathsf{out}(\{\mathsf{m},\mathsf{n}\}_{\mathsf{k}_\mathsf{A}^-}).\mathsf{Q}_2, 2)$, and $i_4 = \mathsf{out}(\{\mathsf{m},\mathsf{n}\}_{\mathsf{k}_\mathsf{A}^-}).\mathsf{Q}_2, 3)$. The environment may combine the two output terms into $\{\mathsf{n}^{i_1}, \{\mathsf{m}^{i_2}, \mathsf{n}^{i_3}\}_{\mathsf{k}_\mathsf{A}^- i_4}\}_{\mathscr{E}}$, which matches the input pattern $\{\mathsf{x},\{\mathsf{y},\mathsf{z}\}_{\mathsf{k}_\mathsf{A}^+}\}_{\mathsf{k}_\mathsf{B}^-}$ with substitution $\sigma = [\mathsf{n}_{(\mathsf{x})}/\mathsf{x}, \mathsf{m}_{(\mathsf{y})}/\mathsf{y}, \mathsf{n}_{(\mathsf{z})}/\mathsf{z}]$: the substitution σ is used to instantiate the free variables in the process Q_3 following the input pattern. Thus $\mathscr{C}$ contains the following nodes and edges:



Notice that the terms in the communication node track both the positional indexes of the names occurring therein and the variable assignment induced by the input pattern. Furthermore, processes obtained by output reduction are causally preceded by the output process and not by communication nodes in that ρ-spi calculus is asynchronous: for instance, Q_1 is causally preceded by $\mathsf{out}(\mathsf{n}).\mathsf{Q}_1$ and not by the communication node.

3.2 Causal graphs as abstraction of ρ-spi processes

Before defining the causal graph associated with an abstract process, we introduce some useful functions. We write $\mathit{output}(\mathsf{out}(\mathsf{G}).\mathsf{P})$ to denote the term obtained from G by (i) erasing the labels in G and (ii) indexing each name a in G with the pair composed of process $\mathsf{out}(\mathsf{G}).\mathsf{P}$ and the positional index of a in G. This function is naturally extended to node sets, thus denoting the set of terms sent on the network: formally, $\mathit{output}(\mathscr{N}) = \{\mathit{output}(\mathsf{out}(\mathsf{G}).\mathsf{P}) \mid \mathsf{out}(\mathsf{G}).\mathsf{P} \in \mathscr{N}\}$. We write $\mathit{index}(\mathsf{G})$ to denote the set of processes occurring in the indexes of G. Notice that labels are local and do not propagate into the environment's knowledge. The knowledge of the abstract environment is formalized by the judgement $\mathcal{G} \vdash \mathsf{G}$ (cf. Table 7 of Appendix B), meaning that the environment can construct G given the knowledge of the terms in $\mathcal{G}$. The environment knows every term in $\mathcal{G}$ and the special name $\mathscr{E}$, it can construct and destruct pairs and encrypt and decrypt terms provided that it knows the encryption and decryption keys, respectively. For reducing the number of terms known to the environment, we prevent the environment from deriving identities, public keys and enemies' keys, which abstract the same ρ-spi terms as $\mathscr{E}$. The function $\mathit{bind}_{\mathsf{abs}} : (\mathsf{M},\mathsf{G}) \mapsto \mathsf{G}'$, reported in Table 7 of Appendix B, defines the pattern-matching among terms. This function takes as input a term M and a ground term G and, if the two terms match, yields the term G' obtained by labelling G according to the variables in M. The substitution σ expressing the variable instantiation induced by the pattern-matching of M with G is returned by the function $\mathit{subst}(\mathsf{G}')$, with $\mathsf{G}' = \mathit{bind}_{\mathsf{abs}}(\mathsf{M},\mathsf{G})$: the function $\mathit{subst} : \mathsf{G} \mapsto \sigma$ yields the least substitution σ such that for every $\mathsf{v}_{(\mathsf{x})} \in \mathit{names}(\mathsf{G})$, $\sigma(\mathsf{x}) = \lceil \mathsf{v} \rceil_{(\mathsf{x})}$. If pattern-matching fails, $\mathit{bind}_{\mathsf{abs}}$ and consequently subst return $\uparrow$. For example, $\mathit{bind}_{\mathsf{abs}}(\mathsf{x},\mathsf{n}^5) = \mathsf{n}^5_{(\mathsf{x})}$ and $\mathit{subst}(\mathsf{n}^5_{(\mathsf{x})}) = [\mathsf{n}_{(\mathsf{x})}/\mathsf{x}]$. Function *threads*, defined below, yields the set of threads in P, abstracting away from identifiers and replications.

$$\mathit{threads}(\mathsf{P}) \triangleq \begin{cases} \mathit{threads}(\mathsf{P}_1) \cup \mathit{threads}(\mathsf{P}_2) & \text{if } \mathsf{P} = \mathsf{P}_1 | \mathsf{P}_2 \\ \mathit{threads}(\mathsf{Q}) & \text{if } \mathsf{P} \in \{\mathsf{A} \triangleright \mathsf{Q}, !\mathsf{Q}\} \\ \{\mathsf{P}\} & \text{otherwise} \end{cases}$$

Finally, function $\mathit{int}(\mathcal{N}, \mathsf{G})$ defines the set of integer terms in G: these are defined as the largest subterms of G that have not been generated by the environment, i.e., either names or ciphertexts whose encryption key is unknown to the environment.

$$\mathit{int}(\mathcal{N}, \mathsf{G}) \triangleq \begin{cases} \{\mathsf{v}\} & \text{if } \mathsf{G} = \mathsf{v} \wedge \mathsf{v} \neq \mathscr{E} \\ \emptyset & \text{if } \mathsf{G} = \mathsf{v} \wedge \mathsf{v} = \mathscr{E} \\ \mathit{int}(\mathcal{N}, \mathsf{G}_1) \cup \mathit{int}(\mathcal{N}, \mathsf{G}_2) & \text{if } \mathsf{G} = (\mathsf{G}_1, \mathsf{G}_2) \\ \{\mathsf{G}\} & \text{if } \mathsf{G} = \{\mathsf{G}'\}_{\mathsf{v}'} \wedge \mathit{output}(\mathcal{N}) \nvdash \mathsf{v}' \\ \mathit{int}(\mathcal{N}, \mathsf{G}') \cup \mathit{int}(\mathcal{N}, \mathsf{v}') & \text{if } \mathsf{G} = \{\mathsf{G}'\}_{\mathsf{v}'} \wedge \mathit{output}(\mathcal{N}) \vdash \mathsf{v}' \end{cases}$$

The next definition characterizes the causal graph associated with an abstract process.

Definition 5 (Causal Graph). *The causal graph $(\mathcal{N}, \mathcal{E})$ associated with P, written as $(\mathcal{N}, \mathcal{E}) = \mathit{graph}(P)$, is given by the least $\mathcal{N}$, $\mathcal{E}$ satisfying the following conditions:*

$\mathit{threads}(P) \subseteq \mathcal{N}$ *(Initial)*

$\mathsf{p.P} \in \mathcal{N} \wedge \mathsf{Q} \in \mathit{threads}(\mathsf{P}) \wedge \mathsf{p} \neq \mathsf{in}(\cdot)$ *(Intra-thread)*
$\Rightarrow \mathsf{Q} \in \mathcal{N} \wedge \mathsf{p.P} \rightarrow \mathsf{Q} \in \mathcal{E}$

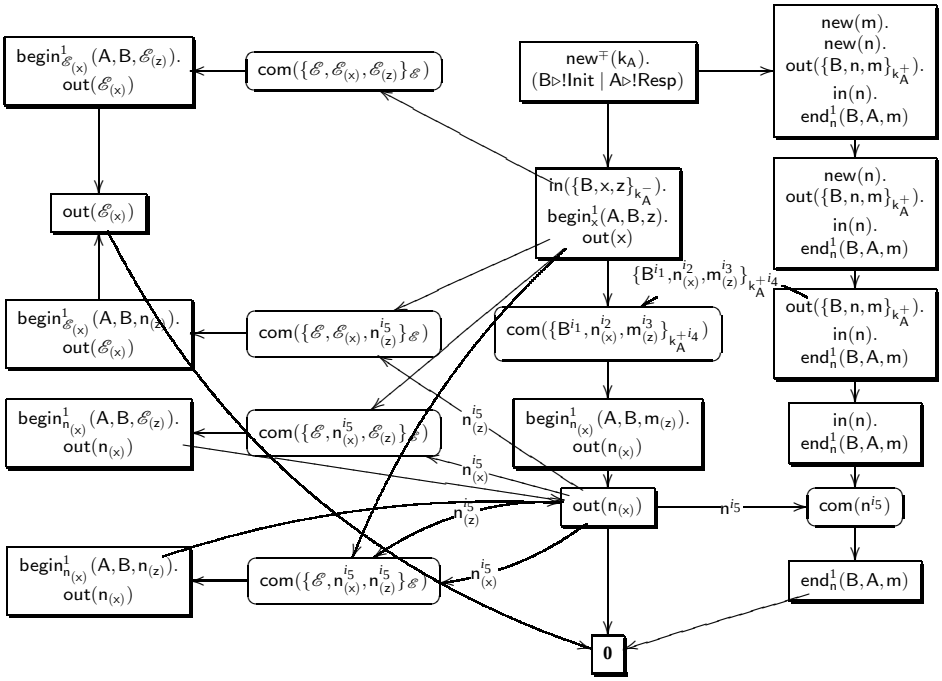
$\mathsf{in(M).P} \in \mathcal{N} \wedge \mathit{output}(\mathcal{N}) \vdash \mathsf{G} \wedge \mathit{bind}_{\mathsf{abs}}(\mathsf{M}, \mathsf{G}) = \mathsf{G}' \wedge \sigma = \mathit{subst}(\mathsf{G}') \wedge$ *(Inter-thread)*
$\quad \mathsf{Q} \in \mathit{threads}(\mathsf{P}\sigma) \wedge \mathsf{G}'_1 \in \mathit{int}(\mathcal{N}, \mathsf{G}') \wedge \{\mathsf{out}(\mathsf{G}_1).\mathsf{P}_1\} = \mathit{index}(\mathsf{G}'_1)$
$\Rightarrow \{\mathsf{Q}, \mathsf{com}(\mathsf{G}')\} \subseteq \mathcal{N} \wedge \{\mathsf{in(M).P} \rightarrow \mathsf{com}(\mathsf{G}'),\ \mathsf{out}(\mathsf{G}_1).\mathsf{P}_1 \xrightarrow{\mathsf{G}'_1} \mathsf{com}(\mathsf{G}'),\ \mathsf{com}(\mathsf{G}') \rightarrow \mathsf{Q}\} \subseteq \mathcal{E}$

Function $\mathit{graph} : P \mapsto (\mathcal{N}, \mathcal{E})$ is in fact a closure operator on causal graphs ordered by inclusion. In particular, the threads in P are part of the set $\mathcal{N}$ of nodes (*Initial*); if $\mathsf{p.P}$ is a node in the causal graph, with $\mathsf{p} \neq \mathsf{in}(\cdot)$, then the processes in $\mathit{threads}(\mathsf{P})$, which are connected to $\mathsf{p.P}$ by direct edges, belong to the set of nodes (*Intra-thread*); if $\mathsf{in(M).P}$ is a node and the environment knows a term G matching M with substitution σ, then the set of nodes contains also the processes in $\mathit{threads}(\mathsf{P}\sigma)$ and the communication node linking the output processes used by the environment to construct G to the processes in $\mathit{threads}(\mathsf{P}\sigma)$ (*Inter-thread*). Intuitively, the input of a term is preceded by the output of *all* messages required by the environment to construct such a term. This point will be clarified later on, when formalizing the causality relation expressed by causal graphs. As stated by the following proposition, every abstract process admits a unique causal graph. In the following, we write $\mathit{length}(P)$ to denote the number of primitives in P and $\mathit{length}(\mathcal{N})$ to denote the maximal length of the processes in $\mathcal{N}$, namely the number n such that $\mathit{length}(\mathsf{Q}) = n$, for some $\mathsf{Q} \in \mathcal{N}$, and, for every $\mathsf{Q}' \in \mathcal{N}$, $\mathit{length}(\mathsf{Q}') \leq n$.

Proposition 1 (Uniqueness). *For every process P, there exists a unique graph(P).*

Proof. Consider the function $\Phi_P(\mathcal{N}, \mathcal{E})$ yielding the least $\mathcal{N}', \mathcal{E}'$ satisfying the three conditions of Definition 5. For proving the thesis, we prove that, for every $\mathcal{N}$ and $\mathcal{E}$, there exists a unique least fixpoint of $\Phi_P(\mathcal{N}, \mathcal{E})$: this trivially implies the thesis. Notice that Φ_P is monotonous over node and edge sets ordered by inclusion.

Table 2 Causal graph *graph*(*Prot*)



Legend: for every $j \in [1,4]$, i_j= (out({B,n,m}$_{k_A^+}$).in(n).end^1_n(B,A,m),j); i_5=(out(n$_{(x)}$),1)

It is easy to see that if $\Phi_P(\mathcal{N},\mathcal{E}) = \mathcal{N}',\mathcal{E}'$, then $length(\mathcal{N}) = length(\mathcal{N}')$. The set of names ($\mathscr{E}$ excluded) and variables occurring in the processes of $\mathcal{N}'$ is contained into the one of names and variables occurring in the processes of $\mathcal{N}$. The set of processes of finite length and composed of a finite set of names and variables is finite, as well as the set of edges. By Knaster-Tarski theorem, Φ_P has a unique least fixpoint. □

The following corollary says that the size of the graph grows exponentially with the protocol *specification*, which is however fixed in advance, regardless of the number of considered sessions and the protocol run-time behaviour.

Corollary 1 (Size of causal graphs). *Let $\mathscr{C}$ be a causal net and P a process such that $\mathscr{C} = graph(P)$. Let N and X be the number of restrictions and variables in P, respectively. Then $|nodes(\mathscr{C})| \leq length(P) * N^X$.*

Example 3. To illustrate, the causal graph associated with protocol *Prot* of Example 1 is depicted in Table 2. The rounded boxes represent communication nodes while the other ones denote process nodes. The analysis of the causal graph gives us some interesting information about the run-time behavior of protocol participants. Even if the protocol is simple, it turns out to be interesting since the attackers know both the public-key used for encrypting the first message and, after the second message, the

nonce used in the protocol session. This increases the number of actions at their disposal and, consequently, the number of nodes and edges in the causal graph. In general, fixed the number of message exchanges, the causal graph for protocols preserving the secrecy of messages and relying on digital signature or symmetric-key cryptography is typically simpler than the one of protocols based on public-key cryptography and possibly exposing some messages to the attackers. Let $\mathcal{N}$ be the set of nodes in the causal graph: we can see that $\mathit{output}(\mathcal{N}) \not\vdash \mathsf{m}^i$ for any index i and this intuitively means that the authenticated message is kept secret, as expected. Furthermore, the only process asserting an end event is $\mathsf{end}^1_{\mathsf{n}}(\mathsf{B},\mathsf{A},\mathsf{m})$. This is preceded by the output of $\mathsf{n}_{(\mathsf{x})}$ (inter-thread causality), which is in turn preceded (intra-thread causality) by node $\mathsf{begin}^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{m}_{(\mathsf{z})}).\mathsf{out}(\mathsf{n}_{(\mathsf{x})})$. Note that the output of $\mathsf{n}_{(\mathsf{x})}$ may be also preceded (intra-thread causality) by $\mathsf{begin}^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathscr{E}_{(\mathsf{z})}).\mathsf{out}(\mathsf{n}_{(\mathsf{x})})$ and $\mathsf{begin}^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{n}_{(\mathsf{z})}).\ \mathsf{out}(\mathsf{n}_{(\mathsf{x})})$. These events are enabled by the environment forging the messages $\{\mathscr{E},\mathsf{n}^{i_5},\mathscr{E}\}_{\mathscr{E}}$ and $\{\mathscr{E},\mathsf{n}^{i_5},\mathsf{n}^{i_5}\}_{\mathscr{E}}$, respectively, and sending them to the responder. This may happen only if the environment knows n and these processes are thus preceded (inter-thread causality) by the output of $\mathsf{n}_{(\mathsf{x})}$. This cycle in the graph tells us that the environment can actively interact with the responder only after the responder has received the message generated by the initiator and "asserted" $\mathsf{begin}^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{m}_{(\mathsf{z})})$, according to the intended protocol run. Intuitively, this means that the causal graph guarantees authenticity. Finally, the portion of the graph in the upper-left corner shows that the environment can forge messages in which the message $\mathscr{E}$ replaces the nonce, but the resulting processes do not increase the environment's knowledge as they eventually output $\mathscr{E}$, which is always known to the environment.

3.3 Paths

Edges describe the causality among process events and each edge may be naturally associated with an action: for instance, if a causal graph contains the edge $\mathsf{out}(\mathsf{n}).\mathsf{P} \rightarrow \mathsf{P}$, then the process P is causally preceded by the reduction of the process $\mathsf{out}(\mathsf{n}).\mathsf{P}$ with action $\mathit{out}(\mathsf{n})$. In general, we express the causality dependency among process events through *paths*, which are sequences of *actions* expressing process events:

$$\mathsf{t} ::= \mathit{new}(\mathsf{n}) \mid \mathit{new}^{\mp}(\mathsf{k}_{\mathsf{I}}) \mid \mathit{in}(\mathsf{G}) \mid \mathit{out}(\mathsf{G}) \mid \mathit{out}_{\mathit{com}}(\mathsf{G}_1,\mathsf{G}_2) \mid \mathit{begin}^i_{\mathsf{G}}(\mathsf{A},\mathsf{I},\mathsf{G}) \mid \mathit{end}^i_{\mathsf{G}}(\mathsf{A},\mathsf{I},\mathsf{G})$$
$$\mathsf{s} ::= \varepsilon \mid \mathsf{s} :: \mathsf{t}$$

Abstract actions extend ρ-spi ones with the new action $\mathit{out}_{\mathit{com}}(\mathsf{G}_1,\mathsf{G}_2)$, which is used for tracking outputs giving rise to inter-thread causality: G_1 is the output term and G_2 is an integer component. We now argue on the number of paths associated with a node. We have mentioned that the input of a term is causally preceded by the outputs of *all* the integer components needed by the environment to construct such a term. Thus an input is preceded by a set of paths. However, the same process may be generated by the input of different terms, say G_1 and G_2. Then such a process is preceded by either all the outputs needed by the environment to construct G_1 or all the outputs needed to construct G_2. This is the reason why the paths associated with a node are actually a set of path sets, meaning that the node is causally preceded by *all* the paths in *one* of its path sets. Notice that, as result of our abstraction which essentially collapses an

unbounded number of instances of each principal into a strand of nodes related by intra-thread causality, causal graphs may contain cycles. For this reason, when evaluating the paths preceding a node, we need to avoid loops: this is achieved by traversing edges connecting an input node with a communication one only once, thus abstracting away from cycles in the causal graph which, in fact, do not alter the causality among nodes.

We write $\{\mathsf{s}_1,\ldots,\mathsf{s}_n\} :: \tau$ to denote $\{\mathsf{s}_1 :: \tau,\ldots,\mathsf{s}_n :: \tau\}$ and $\mathit{labedge}(\mathsf{com}(\mathsf{G}),\mathscr{C})$ to denote the set of labelled edges in $\mathscr{C}$ incoming in $\mathsf{com}(\mathsf{G})$. The function $\mathit{paths} : \mathscr{C},\mathsf{P},\mathcal{E} \mapsto \mathscr{S}$, defined below, yields the set $\mathscr{S}$ of paths sets that are associated with the node P and do not traverse the edges in $\mathcal{E}$. We often write $\mathit{paths}(\mathscr{C},\mathsf{P})$ to denote $\mathit{paths}(\mathscr{C},\mathsf{P},\emptyset)$.

Definition 6 (Paths). *Let $\mathscr{C}$ be a causal graph and $\mathsf{Q} \in \mathit{nodes}(\mathscr{C})$. The paths preceding Q in $\mathscr{C}$ and not traversing the edges in $\mathcal{E}$, written $\mathit{paths}(\mathscr{C},\mathsf{Q},\mathcal{E})$, are given by the least $\mathscr{S}$ such that*

$$\mathsf{Q}\ \textit{has no incoming edge} \ \Rightarrow \mathscr{S} = \{\emptyset\} \qquad \textit{(Initial)}$$

$$\mathsf{p.P} \to \mathsf{Q} \in \mathit{edges}(\mathscr{C}) \ \wedge\ \mathcal{S} \in \mathit{paths}(\mathscr{C},\mathsf{p.P},\mathcal{E}) \ \Rightarrow\ \mathcal{S} :: \mathsf{p} \in \mathscr{S} \qquad \textit{(Intra)}$$

$$\begin{array}{l}\mathsf{com}(\mathsf{G}) \to \mathsf{Q} \in \mathit{edges}(\mathscr{C}) \ \wedge\ \mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G}) \in \mathit{edges}(\mathscr{C}) \setminus \mathcal{E} \ \wedge\ \sigma = \mathit{subst}(\mathsf{G}) \ \wedge \\ \quad \{\mathsf{out}(\mathsf{G}_1).\mathsf{P}_1 \xrightarrow{\mathsf{G}'_1} \mathsf{com}(\mathsf{G}),\ldots,\mathsf{out}(\mathsf{G}_n).\mathsf{P}_n \xrightarrow{\mathsf{G}'_n} \mathsf{com}(\mathsf{G})\} = \mathit{labedge}(\mathsf{com}(\mathsf{G}),\mathscr{C}) \ \wedge \\ \quad \mathcal{S} \in \mathit{paths}(\mathscr{C},\mathsf{in}(\mathsf{M}).\mathsf{P},\mathcal{E} \cup \{\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})\}) \ \wedge \\ \quad \mathcal{S}_i \in \mathit{paths}(\mathscr{C},\mathsf{out}(\mathsf{G}_i).\mathsf{P}_i,\mathcal{E} \cup \{\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})\}) \\ \Rightarrow\ \mathcal{S} :: \mathit{in}(\mathsf{M}\sigma) \cup \bigcup\limits_{i\in[1,n]} \mathcal{S}_i :: \mathit{out}_{com}(\mathsf{G}_i,\mathsf{G}'_i) :: \mathit{in}(\mathsf{M}\sigma) \in \mathscr{S}\end{array} \qquad \textit{(Inter)}$$

If the node Q is preceded by intra-thread causality by $\mathsf{p.P}$, then the path sets associated with Q are the ones associated with $\mathsf{p.P}$, each of them extended with the action p. If Q is obtained by the reduction of a process receiving G ($\mathsf{com}(\mathsf{G}) \to \mathsf{Q} \in \mathit{edges}(\mathscr{C})$, $\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G}) \in \mathit{edges}(\mathscr{C}) \setminus \mathcal{E}$, $\sigma = \mathit{subst}(\mathsf{G})$), where the edge $\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})$ is not in $\mathcal{E}$, then the paths associated with Q are the ones associated to $\mathsf{in}(\mathsf{M}).\mathsf{P}$ ($\mathcal{S} \in \mathit{paths}(\mathscr{C},\mathsf{in}(\mathsf{M}).\mathsf{P},\mathcal{E} \cup \{\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})\})$), each of them extended with the action $\mathit{in}(\mathsf{M}\sigma)$, plus the ones associated with the processes outputting the terms used to construct the input term G ($\{\mathsf{out}(\mathsf{G}_1).\mathsf{P}_1 \xrightarrow{\mathsf{G}'_1} \mathsf{com}(\mathsf{G}),\ldots,\mathsf{out}(\mathsf{G}_n).\mathsf{P}_n \xrightarrow{\mathsf{G}'_n} \mathsf{com}(\mathsf{G})\} = \mathit{labedge}(\mathsf{com}(\mathsf{G}),\mathscr{C})$ and $\mathcal{S}_i \in \mathit{paths}(\mathscr{C},\mathsf{out}(\mathsf{G}_i).\mathsf{P}_i,\mathcal{E} \cup \{\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})\})$), extended with $\mathit{out}_{com}(\mathsf{G}_i,\mathsf{G}'_i) :: \mathit{in}(\mathsf{M}\sigma)$. Notice that the edge $\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})$ is inspected only once, thus avoiding loops due to cycles in the graph.

Example 4. For instance, $\mathit{paths}(\mathit{graph}(\mathit{Prot}),\mathsf{end}^1_{\mathsf{n}}(\mathsf{B},\mathsf{A},\mathsf{m}),\emptyset)$, where $\mathit{graph}(\mathit{Prot})$ is the causal graph in Table 2, yields the path sets reported in Table 8 of Appendix B. The number of possible paths depends on the environment's capability of forging challenges and thus interacting with the responder. Notice that every path set contains a path whose prefix is as follows:

$$\begin{array}{rl} c & \triangleq \mathit{new}^{\mp}(\mathsf{k_A}) :: \mathit{new}(\mathsf{n}) :: \mathit{new}(\mathsf{m}) :: \mathit{out}_{com}(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+},\{\mathsf{B}^{i_1},\mathsf{n}_{(\mathsf{x})}{}^{i_2},\mathsf{m}_{(\mathsf{z})}{}^{i_3}\}_{\mathsf{k}_\mathsf{A}^{+i_4}}) :: \\ & \quad \mathit{in}(\{\mathsf{B},\mathsf{n}_{(\mathsf{x})},\mathsf{m}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: \mathit{begin}^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{m}_{(\mathsf{z})}) \end{array}$$

Notice also that this path represents the intended protocol behavior. Intuitively, this means that the environment can play a number of actions but these necessarily follow

the intended protocol run and thus they do not affect the safety of the protocol as far as authenticity is concerned.

3.4 Session messages

Since a causal graph abstracts an unbounded number of protocol sessions, an interesting issue related to our abstraction is that different occurrences in a path of the same name n might actually abstract different ρ-spi names, one for each protocol session. The problem is that the environment may exploit messages generated in different protocol sessions so as to forge a message which is then used to interact with another session: in fact, this kind of interleaving may break the protocol security goals. According to the definition of causal graphs, the labels of the edges incoming in communication nodes express the integer components in the input term. This information can be used for soundly characterizing the set of *session messages* in a path, namely those messages abstracting ρ-spi messages generated in the same protocol session. Indeed, it is sufficient to check for every pair of output and input actions related by inter-thread causality which messages belong to the integer component and which do not: the former abstract over the same protocol session while the latter may have been generated in different protocol sessions. The function $sm : \mathsf{s} \mapsto \mathcal{G}$, defined below, yields the set $\mathcal{G}$ of session messages in the path s. Here and throughout this paper, we write $\mathsf{msgs}(\mathsf{M})$ to denote the set of messages in M, including those labelled. We also write $\pi^i(\mathsf{G})$ to denote the name occurring in the i-th position of G.

Definition 7 (Session Messages). *We say that $\mathcal{G}$ is the set of session messages of a path s if and only if $sm(\mathsf{s}) = \mathcal{G}$, where function $sm : \mathsf{s} \mapsto \mathcal{G}$ is defined as follows:*

$$sm(\mathsf{s}) \triangleq \begin{cases} \emptyset & \text{if } \mathsf{s} = \varepsilon \\ sm(\mathsf{s}') \cup \{\mathsf{n}\} & \text{if } \mathsf{s} = \mathsf{s}'::new(\mathsf{n}) \\ bound(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2) \cup bymatch(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2) & \text{if } \mathsf{s} = \mathsf{s}'::out_{com}(\mathsf{G}_1, \mathsf{G}_2) :: in(\mathsf{G}) \\ sm(\mathsf{s}') & \text{otherwise } (\mathsf{s} = \mathsf{s}'::\mathsf{t}) \end{cases}$$

$$\text{with } bound(\mathsf{s}, \mathsf{G}_1, \mathsf{G}_2) \triangleq \{\mathsf{v} | \exists j \text{ s.t. } \mathsf{v}^{(\mathsf{P},j)} \in \mathsf{msgs}(\mathsf{G}_2) \wedge \pi^j(\mathsf{G}_1) = \mathsf{v}' \wedge \mathsf{v}' \in sm(\mathsf{s})\}$$

$$bymatch(\mathsf{s}, \mathsf{G}_1, \mathsf{G}_2) \triangleq \{\mathsf{v} | \exists \mathsf{n} \in bound(\mathsf{s}, \mathsf{G}_1, \mathsf{G}_2) \text{ s.t. } \mathsf{s} = \mathsf{s}_1 :: new(\mathsf{n}) :: \mathsf{s}_2 \wedge \mathsf{v} \in sm(\mathsf{s}_1)\}$$

The definition is given by induction on the length of the path. The empty path contains no session message. The restriction inserts the new message in the session messages of the path, while the other actions generated by intra-thread causality do not affect the session messages of the path. In the case of inter-thread causality, the set of session messages is given by the session messages of the path prefix occurring in the integer component ($bound(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2)$) and the messages occurring as session messages in the path at the time of the restriction of some other session message matched in the input pattern ($bymatch(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2)$). The idea is that, by the pattern-matching of a fresh name, one may recover a protocol session described in a previous part of the path.

Example 5. Let us consider the path $d \triangleq c :: out_{com}(\mathsf{n}_{(\mathsf{x})}, \mathsf{n}^{i_5}) :: in(\mathsf{n}) :: end^i_{\mathsf{n}}(\mathsf{B}, \mathsf{A}, \mathsf{m})$, where c is the path of Example 4. The messages occurring in the begin and end assertions are session messages. In particular, $\mathsf{n} \in sm(d)$ since $\mathsf{n} \in bound(c, \mathsf{n}_{(\mathsf{x})}, \mathsf{n}^{i_5})$.

Furthermore, $\mathsf{m} \in sm(d)$ since $\mathsf{m} \in bymatch(c, \mathsf{n}_{(\mathsf{x})}, \mathsf{n}^{i_5})$: the pattern-matching of n, along with the fact that m occurs as session message at the time of n's restriction, guarantees that m and n abstract messages generated in the same protocol session. Similarly, $\mathsf{n}_{(\mathsf{x})} \in sm(c)$ and $\mathsf{m}_{(\mathsf{z})} \in sm(c)$.

In Section 3.3, we have defined the paths associated with a node in a causal graph and, for avoiding loops due to cycles in the graph, we have requested that edges connecting an input node with a communication one are inspected only once. This approximation does not affect the causality among nodes but it might affect the session messages: in particular, a cycle might restrict the set of session messages in a path and thus the approximation might not be sound. For this reason, we only consider a class of causal graphs, called *cycle-invariant* graphs, for which cycles do not restrict the set of session messages in any path. In the following, we write $\mathsf{s} \sim \mathsf{s}'$ to say that s and s' are equal up to some integer components of output actions.

Definition 8 (Cycle-invariance). *We say that $\mathscr{C}$ is cycle-invariant if and only if whenever $\{\mathsf{in}(\mathsf{M}).\mathsf{P} \rightarrow \mathsf{com}(\mathsf{G}), \mathsf{out}(\mathsf{G}_1).\mathsf{P}' \xrightarrow{\mathsf{G}_2} \mathsf{com}(\mathsf{G}), \mathsf{com}(\mathsf{G}) \rightarrow \mathsf{Q}\} \subseteq edges(\mathscr{C})$ and $\mathcal{S} :: in(\mathsf{G}') \in paths(\mathscr{C}, \mathsf{Q}, \emptyset)$, the following conditions hold:*

- *for every $\mathcal{S} \in paths(\mathscr{C}, \mathsf{in}(\mathsf{M}).\mathsf{P}, \emptyset)$ and $\mathsf{s}_1 :: in(\mathsf{G}') :: \mathsf{s}_2 \in \mathcal{S}$ such that $\mathsf{s}_1 \neq \mathsf{s}'_1 :: out_{com}(\mathsf{G}'_1, \mathsf{G}'_2)$, there exists $\mathsf{s} = \mathsf{s}_1 :: in(\mathsf{G}') :: \mathsf{s}'_2 \in \mathcal{S}$ such that $\mathsf{s}_2 \sim \mathsf{s}'_2$ and $sm(\mathsf{s}_1) \subseteq sm(\mathsf{s})$.*
- *for every $\mathcal{S}' \in paths(\mathscr{C}, \mathsf{out}(\mathsf{G}_1).\mathsf{P}', \emptyset)$ and $\mathsf{s}_1 :: out_{com}(\mathsf{G}_1, \mathsf{G}_2) :: in(\mathsf{G}') :: \mathsf{s}_2 \in \mathcal{S}'$, there exists $\mathsf{s} = \mathsf{s}_1 :: out_{com}(\mathsf{G}_1, \mathsf{G}'_2) :: in(\mathsf{G}') :: \mathsf{s}'_2 \in \mathcal{S}'$ such that $\mathsf{s}_2 \sim \mathsf{s}'_2$ and $sm(\mathsf{s}_1) \cap \mathsf{msgs}(\mathsf{G}_1) \subseteq sm(\mathsf{s})$.*

For example, $graph(Prot)$ is cycle-invariant in that input nodes do not belong to cycles, the only output node within cycles is $\mathsf{out}(\mathsf{n}_{(\mathsf{x})})$, and cycles in the causal graph preserve $\mathsf{n}_{(\mathsf{x})}$ as session message. In our experiments, we did not find any protocol whose specification is not cycle-invariant.

4 Abstract Interpretation

In this section we illustrate the relation between causal graphs and ρ-spi semantics. This is formalized by a concretization function, defined on abstract terms, paths, path sets and causal graphs.

4.1 Concretization of causal graphs

Terms The relation between ρ-spi terms and abstract terms is formally defined in Table 3 by the concretization function $\gamma_{\mathsf{trm}} : \mathsf{M} \mapsto \{M_1, \ldots, M_n\}$. Abstract identities are instantiated by the corresponding ρ-spi identity. The concretization of an abstract message yields the set of messages having the same canonical representative. Similar reasoning applies to variables, public and private keys. The special name $\mathscr{E}$ abstracts over identities, public keys, messages possibly generated by the environment and attackers' keys. Finally, the concretization of the remaining terms is given by instantiating the names and the variables occurring therein.

Table 3 Concretization of paths, processes and causal graphs

$$\gamma_{\mathsf{trm}}(\mathsf{M}) \triangleq \begin{cases} \{I\} & \text{if } \mathsf{M} = \mathsf{I} \\ \{n \mid n \in [\![m]\!] \} & \text{if } \mathsf{M} = \mathsf{m} \\ \{y \mid y \in [\![x]\!] \} & \text{if } \mathsf{M} = \mathsf{x} \\ \{k_I'^{+} \mid k_I' \in [\![k_I]\!] \} & \text{if } \mathsf{M} = \mathsf{k}_\mathsf{I}^{+} \\ \{k_I'^{-} \mid k_I' \in [\![k_I]\!] \} & \text{if } \mathsf{M} = \mathsf{k}_\mathsf{I}^{-} \\ \mathcal{ID} \cup \mathcal{K}^{+} \cup \mathcal{M}_{\mathcal{I}} \cup \mathcal{K}_{\mathcal{I}} & \text{if } \mathsf{M} = \mathscr{E} \\ \gamma_{\mathsf{trm}}(\mathsf{a}) & \text{if } \mathsf{M} \in \{\mathsf{a}_{(\mathsf{x})}, \mathsf{a}^{\mathsf{i}}, \mathsf{a}^{\mathsf{i}}_{(\mathsf{x})}\} \\ \{\mathsf{M}\sigma \mid \forall \mathsf{u} \in \mathit{dom}(\sigma), \sigma(\mathsf{u}) \in \gamma_{\mathsf{trm}}(\mathsf{u})\} & \text{otherwise} \end{cases}$$

$$\gamma_{\mathsf{path}}(\mathsf{s}) \triangleq \begin{cases} \{\varepsilon\} & \text{if } \mathsf{s} = \varepsilon \\ \{s' :: \mathit{new}(m) \mid s' \in \gamma_{\mathsf{path}}(\mathsf{s}') \wedge \\ \quad m \in \gamma_{\mathsf{trm}}(\mathsf{n}) \wedge m \notin \mathit{names}(s')\} & \text{if } \mathsf{s} = \mathsf{s}'::\mathit{new}(\mathsf{n}) \\ \{s' :: \mathit{out}(\mathsf{G}_1\sigma') :: \mathit{in}(\mathsf{G}\sigma) \mid s' \in \gamma_{\mathsf{path}}(\mathsf{s}') \wedge \\ \quad \sigma \in \mathsf{ext}(\mathsf{bsm}(s,\mathsf{s})) \wedge \sigma' \in \mathsf{ext}(\mathsf{bsm}(s',\mathsf{s}'))\} & \text{if } \mathsf{s} = \mathsf{s}' :: \mathit{out}_{\mathit{com}}(\mathsf{G}_1,\mathsf{G}_2) :: \mathit{in}(\mathsf{G}) \\ \{s' :: \mathsf{t}\sigma \mid s' \in \gamma_{\mathsf{path}}(\mathsf{s}') \wedge \sigma \in \mathsf{ext}(\mathsf{bsm}(s',\mathsf{s}'))\} & \text{otherwise } (\mathsf{s} = \mathsf{s}'::\mathsf{t}) \end{cases}$$

$$\gamma_{\mathsf{pset}}(\mathcal{S}) \triangleq \{ \{s_1,\ldots,s_n\} \mid \mathcal{S} = \{\mathsf{s}_1,\ldots,\mathsf{s}_n\} \wedge \forall i \in [1,n], s_i \in \gamma_{\mathsf{path}}(\mathsf{s}_i) \wedge \uplus_{i\in[1,n]} \mathsf{bsm}(s_i,\mathsf{s}_i) \neq \uparrow \}$$

$$\begin{aligned} \gamma_{\mathsf{net}}(\mathscr{C}) \triangleq \{ & \langle t_1 :: \ldots :: t_n, P \rangle \mid \\ & (i)\ \forall i \in [1,n], \exists \mathsf{Q}_i \in \mathit{nodes}(\mathscr{C}), \mathcal{S}_i \in \mathit{paths}(\mathscr{C}, \mathsf{Q}_i, \emptyset), S_i \text{ s.t. } S_i \in \gamma_{\mathsf{pset}}(\mathcal{S}_i) \wedge \\ & \quad ((\mathcal{S}_i = \mathcal{S}_j :: \mathsf{t}_i \wedge S_i = S_j :: t_i, \text{ with } j < i) \vee \\ & \quad (\mathcal{S}_i = \bigcup_{j=1,k} \mathcal{S}_{i'_j} :: \mathit{out}_{\mathit{com}}(\mathsf{G}_{i_j}, \mathsf{G}'_{i_j}) :: \mathit{in}(\mathsf{G}) \cup \mathcal{S}_r :: \mathit{in}(\mathsf{G}) \wedge \mathcal{S}_{i_j} = \mathcal{S}_{i'_j} :: \mathit{out}(\mathsf{G}_{i_j}) \wedge \\ & \quad S_i = \bigcup_{j=1,k} S_{i_j} :: t_i \cup S_r :: t_i \text{ with } i'_j < i_j \text{ and } i_j, r < i)) \\ & (ii)\ \forall Q \in \mathit{threads}(P),\ \exists i \text{ s.t. } Q = \mathsf{Q}_i\sigma, \text{ with } \sigma \in \mathsf{ext}(\mathsf{bsm}(S_i,\mathcal{S}_i)) \end{aligned}$$

Notation:

$\pi^i(s)$ yields the i-th action in s

$\mathsf{bsm}(s,\mathsf{s}) \triangleq \{\sigma \mid \mathit{dom}(\sigma) = \mathit{sm}(\mathsf{s}) \wedge \sigma(\mathsf{v}) = n \text{ iff } \exists i \text{ s.t. } \pi^i(\mathsf{s}) = \mathit{new}([\mathsf{v}]) \wedge \pi^i(s) = \mathit{new}(n)\}$

$\mathsf{bsm}(\{s_1,\ldots,s_n\},\{\mathsf{s}_1,\ldots,\mathsf{s}_n\}) = \uplus_{i\in[1,n]} \mathsf{bsm}(s_i,\mathsf{s}_i)$

$\mathsf{ext}(\sigma) \triangleq \{\sigma' \mid \sigma'(\mathsf{u}) = \begin{cases} \sigma(\mathsf{u}) & \text{if } \mathsf{u} \in \mathit{dom}(\sigma) \\ u \in \gamma_{\mathsf{trm}}(\mathsf{u}) & \text{otherwise} \end{cases} \}$

$\sigma_1 \uplus \sigma_2 = \sigma_1 \cup \sigma_2 \qquad$ if $\sigma_1 \neq \uparrow \wedge \sigma_2 \neq \uparrow \wedge \mathsf{u} \in \mathit{dom}(\sigma_1) \cap \mathit{dom}(\sigma_2) \Rightarrow \sigma_1(\mathsf{u}) = \sigma_2(\mathsf{u})$

$\sigma_1 \uplus \sigma_2 = \uparrow \qquad$ otherwise

Paths The concretization function $\gamma_{\mathsf{path}} : \mathsf{s} \mapsto \{s_1, \ldots, s_n\}$ takes as input a path and yields a set of ρ-spi traces. More precisely, the concretization of the empty path is the set composed of the empty trace. The concretization of $\mathsf{s} = \mathsf{s}'::new(\mathsf{n})$, yields the set of traces of the form $s' :: new(n)$, where n is a message fresh in the trace and belonging to $\gamma_{\mathsf{trm}}(\mathsf{n})$ and $s' \in \gamma_{\mathsf{path}}(\mathsf{s}')$.

The concretization of $\mathsf{s}' :: out_{com}(\mathsf{G}_1, \mathsf{G}_2) :: in(\mathsf{G})$ yields the set of traces of the form $s' :: out(\mathsf{G}_1\sigma') :: in(\mathsf{G}\sigma)$, with $s' \in \gamma_{\mathsf{path}}(\mathsf{s}')$, $\sigma' \in \mathsf{ext}(\mathsf{bsm}(s', \mathsf{s}'))$ and $\sigma \in \mathsf{ext}(\mathsf{bsm}(s, \mathsf{s}))$. The function $\mathsf{bsm}(s', \mathsf{s}')$ maps each session message m in s' to the corresponding ρ-spi message in s', which is determined according to the restriction operator $\mathsf{new}(\mathsf{m})$ in the path and to its representative $\mathsf{new}(m)$ in the ρ-spi trace. The set $\mathsf{ext}(\sigma)$ contains the substitutions extending σ by mapping abstract names to one of their possible ρ-spi concretizations. This way the occurrences, possibly labelled, of a session message are instantiated by the same ρ-spi name. Finally, the concretization of any other path $\mathsf{s} = \mathsf{s}'::\mathsf{t}$ is given by the set of traces of the form $s' :: \mathsf{t}\sigma$, with $s' \in \gamma_{\mathsf{path}}(\mathsf{s}')$ and $\sigma \in \mathsf{ext}(\mathsf{bsm}(s', \mathsf{s}'))$.

Example 6 (Path Concretization). To illustrate, a concretization of the path reported in Example 5 is as follows:

$$new^{\mp}(k_A) :: new(m) :: new(n) :: out(\{B, n, m\}_{k_A^+}) ::$$
$$in(\{B, n, m\}_{k_A^-}) :: begin_n^1(A, B, m) :: out(n) ::$$
$$in(n) :: end_n^1(B, A, m)$$

Notice that both n and $\mathsf{n}_{(\mathsf{x})}$ occur as session messages in the path and are consequently instantiated by the same name n. Similarly, m and $\mathsf{m}_{(\mathsf{z})}$ occur as session messages and are instantiated by the same name m. Notice also that the path guarantees authenticity and the trace guarantees strong authenticity.

Path sets The concretization of a path set is given by the concretization of the paths occurring therein. It is worth to mention that the instantiation of the session messages in such paths has to be consistent, i.e., if different paths share the same session message then the concretization of such a message has to be the same ($\biguplus_{i \in [1,n]} \mathsf{bsm}(s_i, \mathsf{s}_i) \neq \uparrow$). The union $\sigma_1 \uplus \sigma_2$ succeeds only when the abstract names substituted by both σ_1 and σ_2 are bound to the same ρ-spi name.

Configurations The concretization function $\gamma_{\mathsf{net}} : \mathscr{C} \mapsto \{\langle s_1, P_1\rangle, \ldots, \langle s_n, P_n\rangle\}$ takes as input a causal graph and yields a set of ρ-spi configurations satisfying two constraints, the former concerning traces and the latter concerning processes. In the following, we let S range over ρ-spi trace sets.

i for every action τ_i in the trace there exists a node Q_i in the causal graph, a path set $\mathcal{S}_i$ associated with Q_i and a trace set S_i such that $S_i \in \gamma_{\mathsf{pset}}(\mathcal{S}_i)$, where S_i is the trace set associated with a preceding action in the trace extended with τ_i.

ii for every thread Q of the process in the ρ-spi configuration, there exists a node Q_i in the causal graph, a path set $\mathcal{S}_i$ and a trace set S_i such that $Q = \mathsf{Q}_i\sigma$, where σ extends $\mathsf{bsm}(S_i, \mathcal{S}_i)$.

Intuitively, condition (i) requires that the actions in the trace respect the causality paths induced by the causal graph and condition (ii) requires that every process has an abstraction in the causal graph and the trace respects the causality paths associated with such a node.

Finally, we state the soundness results of the abstract interpretation. The following theorem says that the causal graph generated from a process is an abstraction of the configuration composed of such a process and the empty trace.

Theorem 1 (Soundness). *If* $graph(P) = \mathscr{C}$, *then* $\langle \varepsilon, P \rangle \in \gamma_{net}(\mathscr{C})$.

Due to space constraints, we postpone the proofs to Appendix C. The following theorem says that the set of configurations abstracted by a causal graph is closed under process reduction, i.e., causal graphs are a sound abstraction of the ρ-spi semantics.

Theorem 2 (Preservation). *If* $\langle s, P \rangle \rightarrow \langle s', P' \rangle$ *and* $\langle s, P \rangle \in \gamma_{net}(\mathscr{C})$, *then* $\langle s', P' \rangle \in \gamma_{net}(\mathscr{C})$.

5 Safety Results

In this section we state the safety results of our static analysis technique. As stated by the following definition, a causal graph guarantees the secrecy of a name v, if the abstract environment cannot deduce any term which is equal to v up to index erasure.

Definition 9 (Abstract secrecy). *A causal graph* $\mathscr{C}$ *guarantees the secrecy of* v *if and only if* $output(nodes(\mathscr{C})) \vdash \mathsf{v}'$ *implies* $\lceil \mathsf{v} \rceil \neq \lceil \mathsf{v}' \rceil$.

As stated below, a path guarantees authenticity if (i) every end assertion is preceded by a corresponding begin assertion; and (ii) begin and end assertions contain only session messages. This guarantees that authenticity carries over the action sequences abstracted by the path. A causal graph guarantees authenticity if for every $\mathsf{end}^i_{\mathsf{G}_1}(\mathsf{A},\mathsf{B},\mathsf{G}_2).\mathsf{P}$ and $\mathcal{S} \in paths(\mathscr{C}, \mathsf{end}^i_{\mathsf{G}_1}(\mathsf{A},\mathsf{B},\mathsf{G}_2).\mathsf{P})$, there exists a path in $\mathcal{S}$ containing a suitable begin assertion.

Definition 10 (Abstract Authenticity). *A path* s *guarantees authenticity if and only if whenever* $\mathsf{s} = \mathsf{s}_1 :: end^i_{\mathsf{G}_1}(\mathsf{A},\mathsf{B},\mathsf{G}_2) :: \mathsf{s}_2$, *the following conditions hold:*

i $\exists \mathsf{G}'_1, \mathsf{G}'_2$ *s.t.* $\mathsf{s}_1 = \mathsf{s}'_1 :: begin^i_{\mathsf{G}'_1}(\mathsf{B},\mathsf{A},\mathsf{G}'_2) :: \mathsf{s}'_2$, $\lfloor \mathsf{G}_1 \rfloor = \lfloor \mathsf{G}'_1 \rfloor$ *and* $\lfloor \mathsf{G}_2 \rfloor = \lfloor \mathsf{G}'_2 \rfloor$

ii $\mathsf{msgs}(\mathsf{G}'_1, \mathsf{G}'_2) \subseteq sm(\mathsf{s}_1{}') \wedge \mathsf{msgs}(\mathsf{G}_1, \mathsf{G}_2) \subseteq sm(\mathsf{s}_1)$

A causal graph $\mathscr{C}$ *guarantees authenticity iff for every* $\mathsf{end}^i_{\mathsf{G}_1}(\mathsf{A},\mathsf{B},\mathsf{G}_2).\mathsf{P} \in nodes(\mathscr{C})$ *and* $\mathcal{S} \in paths(\mathscr{C}, \mathsf{end}^i_{\mathsf{G}_1}(\mathsf{A},\mathsf{B},\mathsf{G}_2).\mathsf{P})$, *there exists* $\mathsf{s} \in \mathcal{S}$ *s.t.* $\mathsf{s} :: end^i_{\mathsf{G}_1}(\mathsf{A},\mathsf{B},\mathsf{G}_2)$ *guarantees authenticity.*

For example, the path *d* in Example 5 guarantees authenticity as the end assertion is preceded by a corresponding begin assertion and the messages occurring therein are session messages. By an inspection of Table 8, every $\mathcal{S} \in paths(graph(\mathsf{Prot}), \mathsf{end}^1_{\mathsf{n}}(\mathsf{B},\mathsf{A},\mathsf{m}))$ contains a path guaranteeing authenticity. Thus $graph(Prot)$ guarantees authenticity. The following theorems state that causal graphs constitute a sound model for the static verification of secrecy and authenticity. In particular, the next theorem says that if the causal graph associated with *P* guarantees the secrecy of v, then *P* guarantees the secrecy of any concretization of *v*.

Theorem 3 (Secrecy). *Let P be a process and $\mathscr{C} = graph(P)$ a cycle-invariant causal graph. If $\mathscr{C}$ guarantees the secrecy of v then P guarantees the secrecy of any $v \in \gamma_{trm}(\mathsf{v})$.*

For instance, the causal graph of Example 3 guarantees the secrecy of m and, consequently, the protocol of Example 1 guarantees the secrecy of the authenticated message in every protocol session. In the following we say that a process is *nonce linear* iff (i) every end assertion has the form $\mathsf{end}_n^i(I,J,M)$ and is preceded by $\mathsf{new}(n)$ and (ii) if $\mathsf{end}_n^i(I,J,M)$ occurs inside the scope of a replication then $\mathsf{new}(n)$ occurs inside the scope of the same replication. In fact, this syntactic condition suffices to prove that authenticity on causal graphs implies strong authenticity on the ρ-spi processes abstracted by such causal graphs.

Theorem 4 (Authenticity). *Let P be a process and $\mathscr{C} = graph(P)$ be cycle-invariant. If $\mathscr{C}$ guarantees authenticity, then P guarantees weak authenticity. If $\mathscr{C}$ guarantees authenticity and P is nonce linear, then P guarantees strong authenticity.*

This means that causal graphs, which express the causality among process events, are a sound (and decidable) model for proving weak authenticity while the freshness of authentication requests, namely the condition distinguishing weak from strong authenticity, may be directly verified on the syntax of ρ-spi processes. For example, since $graph(Prot)$ guarantees authenticity and *Prot* is nonce-linear, then *Prot* guarantees strong authenticity. Notice that *Prot* describes an unbounded number of instances of A acting as claimant in protocol sessions with B and an unbounded number of instances of B acting as verifier in protocol sessions with A. We can easily extend the protocol specification with the parallel composition of A and B running protocol sessions with an arbitrary malicious party E. Even in this scenario, the protocol turns out to be safe.

6 Related Work

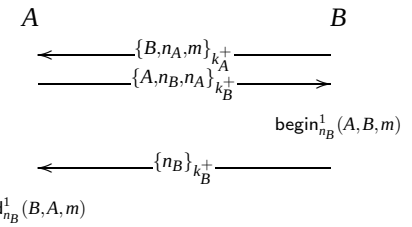
Causality-based modelling of concurrency is a widely studied research topic and several important results have been proposed. Event structures [32] are a general and expressive framework for modelling the causality among events in concurrent and distributed systems. This model captures the dependency among events and the interleaving of concurrent events by a partial order. A tricky problem when abstracting away from the multiplicity of protocol sessions is that an event may causally precede itself, thus loosing the antisymmetry property and, consequently, the partial order. This is easily seen by thinking of a process inputting a message and then sending out another message, which is used by the environment to construct a message sent to a replication of the former process and so on. In this scenario, since we abstract away from the multiplicity of protocol sessions, the input is causally preceded by itself. The safety of this kind of abstraction requires to determine which events abstract over the same protocol session and which ones may instead abstract over different protocol sessions. Thus a more specific structure was needed, containing some additional information about message integrity, and relying on paths representing computational flows instead of a partial order among events. Crazzolara and Winskel have applied Petri nets [29], a well-known causality-based model for distributed systems, to the analysis of cryptographic protocols [14] but

their work does not abstract away from the multiplicity of sessions. However, Petri nets might constitute a suitable alternative foundation for our work.

Type systems proved successful in analyzing different security properties of cryptographic protocols, e.g., [1, 2] for secrecy and [22, 11] for authenticity. As mentioned in the introduction, they exploit syntactic patterns of security protocols while providing guaranteed compositionality, but they constrain the class of analyzable protocols and are typically specific to individual security properties. On the one hand, the type system in [22] is very general and applies to several settings including authorization policies [21] and key-compromise [23], but type definitions quickly get cumbersome and no type-inference algorithm is currently available. The type system in [11] is compositional, modular, and allows for automatic type inference, but this generality is paid by restricting the form of protocols to some specific tagged patterns.

As a simple example, to the best of our knowledge, the protocol in Table 4 can be analyzed by neither the type system from [22] nor the one from [11]. This protocol is a variation of the Needham-Schroeder-Lowe public-key authentication protocol, in which B sends a message m to A in the first message exchange. From A's point of view, the authentication of this message is guaranteed by the handshake with nonce n_B (second and third message). The problem is that m occurs within neither the second nor the third message, thus resulting free in the corresponding type definitions and not enabling the end assertion based on nonce n_B. Another remarkable difference with respect to existing type-based approaches, which in case of failure do not give any information on possibly attacks, is that the analysis of computational flows does yield insights on possible attacks and, in fact, their derivation is often immediate.

Table 4 Variant of NSL



The work by Bodei *et al.* on control-flow analysis for message authentication in *Lysa* [8] is closely related to our approach. The motivations and goals are different though, since the analysis only applies to message authenticity, namely the origin of a message, without addressing freshness of authentication requests and the willingness of the claimant to authenticate with the verifier. Another interesting work is an abstract interpretation for mobile systems and, as an instance, for spi-calculus [19]. The analysis deals with the origin of messages but does not address freshness.

Strand spaces [24, 25] constitute an effective framework for studying secrecy and authenticity. While some effective patterns for deriving the proofs of safety are provided, these are in general written by hand. Interestingly, such proofs often rely on the causality among events and it would be interesting to explore how our analysis could be applied in such a framework to provide mechanical proofs of safety. A recent paper [16] investigates how to automatically detect those specific executions, among the infinitely many possible, that should be considered for analysis, by identifying causality between nodes in graphs and describing execution of protocol roles by different processes. Al-

though this method does not enjoy guaranteed termination, we plan to exploit the ideas underlying this result to further refine the expressivity of our analysis.

Proverif [4, 5] constitutes a powerful tool that takes as input spi-like protocol descriptions and, by Horn clause resolution, verifies a variety of different security properties such as secrecy, perfect secrecy and authenticity. The analysis is general and fully automated but guarantees termination only for protocols where every ciphertext is tagged differently [6].

7 Conclusion and Future Work

We have proposed a novel technique for analyzing security protocols based on abstract interpretation of the causality among process events. In this paper, we have specifically shown that secrecy and authenticity can be soundly characterized in terms of causality, but we remark that the analysis is not tailored to these security properties but may as well be applicable to verify properties formulated in terms of causality among the actions of execution traces; we are currently extending our technique to analyze more sophisticated security properties emerging in modern applications such as e-voting protocols. The analysis enjoys guaranteed termination since the size of causal graphs is finite, the generation of paths terminates since communication edges (i.e., edges connecting an input node with a communication one) are inspected only once, thus avoiding loops due to cycles in the graph, and the analysis is linear on the number of paths. We have implemented a tool for automating the analysis, and we have applied the tool to some common protocols in the literature [18] (among others, the Needham-Schroeder-Lowe public-key protocol, its fixed version proposed by Lowe, the BAN modified version of CCITT X.509(3) and SPLICE/AS). The analyses terminated within a few seconds and provided safety proofs for the correct versions of the protocols while failing to validate the flawed ones. Remarkably, attacks are often easily derivable by an inspection of the path sets. The only human effort required is to capture the protocol in the dialect of the spi-calculus which is often straightforwardly derivable from the protocol description.

As future work, we plan to investigate a more sophisticated abstraction allowing us to relax some of the constraints that are currently imposed by our analysis: for instance, our experiments show that some false positives occur when an authenticated term, e.g., a session key, has to be kept secret until authentication requests are accepted and is then leaked out. This may be exploited for modeling session key corruption. The reason for such false positives is that the check on the freshness of nonces is used so far as sufficient condition for proving that weak authenticity implies strong authenticity: more generally, nonce checks guarantee that different protocol sessions rely on different nonces and we believe that this information can be used for refining the analysis and, more precisely, for excluding those paths in causal graph where a check on the same nonce is performed more than once, thus ruling out this kind of false positives.

We remark that, for the sake of readability, we have not considered operators such as tags and hashes. Their insertion in our framework does not induce any complication but is left as future work. Finally, we plan to investigate compositionality properties on causal graphs and to apply the principles underlying our technique to the applied pi-calculus, whose flexible and general equational theory for terms will give rise to interesting issues concerning the convergence of the analysis.

References

1. M. Abadi. Secrecy by typing in security protocols. *JACM*, 46(5):749–786, 1999.
2. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. *Theoretical Computer Science*, 298(3):387–415, 2003.
3. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
4. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of 14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.
5. B. Blanchet. From Secrecy to Authenticity in Security Protocols. In Manuel Hermenegildo and Germán Puebla, editors, *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes on Computer Science*, pages 342–359, Madrid, Spain, September 2002. Springer Verlag.
6. B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proceedings of Foundations of Software Science and Computation Structures*, pages 136–152, 2003.
7. D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *LNCS*, pages 1–12. Springer-Verlag, 1998.
8. 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.
9. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of 28th International Colloquium on Automata, Languages and Programming (ICALP 01)*, volume 2076, pages 667–681. LNCS 2076, Springer Verlag, 2001.
10. M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proceedings of 14th Symposium on Logic in Computer Science (LICS '99)*, pages 157–166, 1999.
11. M. Bugliesi, R. Focardi, and M. Maffei. Dynamic types for authentication. To appear in Journal of Computer Security.
12. M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer-Verlag, 2004.
13. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, New York, NY, USA, 1977. ACM Press.
14. F. Crazzolara and G. Winskel. Events in security protocols. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 96–105, New York, NY, USA, 2001. ACM Press.
15. D. Denning and G. Sacco. Timestamps in key distributed protocols. *Communication of the ACM*, 24(8):533–535, 1981.
16. Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Searching for shapes in cryptographic protocols, 2007. To appear in Proceedings of Thirteenth International Conference on tools and algorithms for the construction and analysis of systems (TACAS 2007).
17. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
18. Laboratoire Spécification et Vérification. Security protocols open repository. http://www.lsv.ens-cachan.fr/spore/.
19. J. Feret. *Analysis of Mobile Systems by Abstract Interpretation*. PhD thesis, École Polytechnique, 2005.

20. D. Fisher. Millions of .Net Passport accounts put at risk. *eWeek*, May 2003. (Flaw detected by Muhammad Faisal Rauf Danka).
21. C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization policies. In *Proceedings of European Symposium on Programming (ESOP 2005)*, LNCS, pages 141–156. Springer-Verlag, 2005.
22. A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3/4):435–484, 2004.
23. A. D. Gordon and A. Jeffrey. Secrecy despite compromise: Types, cryptography, and the pi-calculus. In *Proceedings of CONCUR 2005 - Concurrency Theory, 16th International Conference*, volume 3653, pages 186–201. Springer-Verlag, 2005.
24. J. D. Guttman and F. J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.
25. J. D. Guttman, F.J. Thayer, J. A. Carlson, J. C. Herzog, J. D. Ramsdell, and B. T. Sniffen. Trust management in strand spaces: a rely-guarantee method. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag, 2004.
26. G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop (CSFW'97)*, pages 31–44. IEEE Computer Society Press, 1997.
27. R. M. Needham and M. D. Schroeder. Authentication revisited. *ACM SIGOPS Operating Systems Review*, 21(1):7–7, 1987.
28. B. Clifford Neuman and Theodore Ts'o. Kerberos : An authentication service for computer networks. *IEEE Communication*, 32(9):33–38, 1994.
29. J. L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.
30. M. D. Schroeder R. M. Needham. Using encryption for authentication in large networks of computers. *ACM Communication*, 21(12):993–999, 1978.
31. D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of 2nd USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.
32. G. Winskel. Event structures. In *Advances in Petri nets 1986, part II on Petri nets: applications and relationships to other models of concurrency*, pages 325–392, New York, NY, USA, 1987. Springer-Verlag New York, Inc.
33. T.Y.C. Woo and S.S. Lam. "A Semantic Model for Authentication Protocols". In *Proceedings of 1993 IEEE Symposium on Security and Privacy*, pages 178–194, 1993.

A Semantics of ρ-spi

The dynamics of ρ-spi is formalized by means of a transition relation, which is reported in Table 5, between *configurations*, i.e., pairs $\langle s, P\rangle$, where $s \in Act^*$ is a trace and P is a closed process. Each transition $\langle s, P\rangle \rightarrow \langle s :: t, P'\rangle$ simulates one computation step in P and records the corresponding action in the trace. The set of all possible actions, noted Act, includes the action $new(n)$ generated by name restriction and $new^{\mp}(k_I)$ by key-pair restriction, $in(G)$ generated by input, $out(G)$ by output, $begin^i_G(A, I, G_1; G_2)$ and $end^i_G(A, I, G_1; G_2)$ by 'begin' and 'end', respectively. Some transitions apply substitutions to processes: formally, a substitution $\sigma : x \mapsto G$ is a function from variables to run-time messages. Often substitutions are written explicitly by $[G_1/x_1, \ldots, G_n/x_n]$. 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. NAME 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

Table 5 Transition System for ρ-spi

Transition rules: We omit the symmetric rule of PAR.

NAME RES
$$\frac{n \notin \mathrm{bn}(s) \cup \mathrm{fn}(s)}{\langle s, \mathsf{new}(n).P\rangle \rightarrow \langle s :: new(n), P\rangle}$$

KEY-PAIR RES
$$\frac{k_I^-, k_I^+ \notin \mathrm{bn}(s) \cup \mathrm{fn}(s)}{\langle s, \mathsf{new}^{\mp}(k_I).P\rangle \rightarrow \langle s :: new^{\mp}(k_I), P\rangle}$$

INPUT
$$\frac{s \vdash G \qquad \sigma = bind(M,G) \neq \uparrow}{\langle s, \mathsf{in}(M).P\rangle \rightarrow \langle s :: in(M\sigma), P\sigma\rangle}$$

OUTPUT
$$\langle s, \mathsf{out}(G).P\rangle \rightarrow \langle s :: out(G), P\rangle$$

BEGIN
$$\langle s, \mathsf{begin}^i_G(A, I, G_1; G_2).P\rangle \rightarrow \langle s :: begin^i_G(A, I, G_1; G_2), P\rangle$$

END
$$\langle s, \mathsf{end}^i_G(A, I, G_1; G_2).P\rangle \rightarrow \langle s :: end^i_G(A, I, G_1; G_2), P\rangle$$

PRINCIPAL
$$\frac{\langle s, P\rangle \rightarrow \langle s :: t, P'\rangle}{\langle s, A \triangleright P\rangle \rightarrow \langle s :: t, A \triangleright P'\rangle}$$

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

REPLICATION
$$\langle s, !P\rangle \rightarrow \langle s, P \mid !P\rangle$$

occurrences in P with a different name having the same canonical representative. Similar reasoning applies to the restriction of key-pairs. 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 6 and discussed below. The run-time message G is read only if it can be pattern-matched with the input term M via the function *bind*, which is defined in Table 6 and discussed below. We write $\overline{a}$ to denote the decryption key corresponding to a. We have $\overline{n} = n$, $\overline{k_{IJ}} = k_{IJ}$, $\overline{I} = I$, $\overline{k_I^+} = k_I^-$ and $\overline{k_I^-} = k_I^+$. OUTPUT, BEGIN and END are self-explanatory. Finally, PRINCIPAL adds the principal name to the performed action, PAR interleaves two different protocol executions and REPLICATION arbitrarily replicates a principal.

Function *bind* takes as input a static term M and a run-time message G and, in case it exists, yields the substitution σ which makes M equal to G, up to the different notation for encryption. If pattern-matching fails, *bind* returns $\uparrow$. This function is defined by cases on the structure of term M: a name matches a name with empty substitution; a variable can be bound to either an atomic name a; pairs match pairs yielding a substitution which is the union $\uplus$ of the ones achieved for the subterms; finally, decryptions must be performed with the correct decryption key. In all the other cases, *bind* returns failure $\uparrow$.

The knowledge of the environment is formalized by the deductive system reported in Table 6. Rule OUT says that every message sent on the network is known by the environment. ENV allows the environment to know any name which is not bound (i.e., restricted) in the trace. By PAIR and PAIR DES, the environment may construct and

Table 6 Deductive system and binding

Message Manipulation Rules

$$\textsc{Out}\ \frac{out(G) \in s}{s \vdash G} \qquad \textsc{Env}\ \frac{a \notin \mathrm{bn}(s)}{s \vdash a} \qquad \textsc{Pair}\ \frac{s \vdash G_1 \quad s \vdash G_2}{s \vdash (G_1, G_2)} \qquad \textsc{Pair Des}\ \frac{s \vdash (G_1, G_2)}{s \vdash G_i,\ i \in [1,2]}$$

$$\textsc{Encryption}\ \frac{s \vdash G \quad s \vdash a}{s \vdash \{G\}_a} \qquad \textsc{Decryption}\ \frac{s \vdash \{G\}_a \quad s \vdash \overline{a}}{s \vdash G}$$

$$\textsc{Public Keys}\ s \vdash k_I^+ \qquad \textsc{Enemy Keys}\ s \vdash k_{EI} \quad s \vdash k_{IE} \quad s \vdash k_E^-$$

Binding

$$\begin{aligned}
&bind(a,a) = [] \\
&bind(x,a) = [a/x] \\
&bind((M,M'),(G,G')) = bind(M,G) \uplus bind(M',G') \\
&bind(\{M\}_{\overline{a}},\{G\}_a) = bind(M,G) \\
&bind(M,G) = \uparrow \qquad \text{otherwise}
\end{aligned}$$

destruct pairs. By ENCRYPTION, and DECRYPTION the environment can encrypt and decrypt messages only knowing the required keys. By PUBLIC KEYS, all the public keys are known by the environment. Finally, by ENEMY KEYS, the environment may be provided with its own private keys and with long-term keys shared with honest participants. This gives the possibility to the enemy to start authentication sessions and, generally speaking, to interact with the other participants by pretending to be a trusted principal.

A.1 Process well-formedness

We remark that the ρ-spi syntax is quite liberal, giving the possibility of writing nonsense processes like $A \triangleright B \triangleright P$, where an identity declaration nests within another one, or $A \triangleright \mathsf{begin}_n(B,I,M)$, where A asserts a begin-event in place of B. Furthermore, it makes sense to require that a shared key k_{IJ} should only be used by I and J and a private key k_I^- should only be used by I. Although this is not required by the analysis, the ρ-spi calculus comes with a notion of process well-formedness ruling out the above mentioned undesired process behaviors and enforcing the correct use of long-term keys. It is formalized through the standard notion of scope: we say that the *scope* of $A\triangleright$ in process $A \triangleright P$, is process P.

Definition 11 (Process Well-Formedness). *A process P is well-formed if the following conditions hold:*

1. *All the occurrences of $A\triangleright$ in P are not in the scope of some other $B\triangleright$;*
2. *Every $\mathsf{begin}^i_N(A,I,M)$ in P is in the scope of $A\triangleright$;*

3. *Every* $\mathsf{end}^i_N(B,J,M)$ *in P is in the scope of* $B\triangleright$*;*
4. *Every symmetric key* k_{IJ} *occurring in P is in the scope of either* $I\triangleright$ *or* $J\triangleright$*;*
5. *Every private key* k_I^- *occurring in P is in the scope of* $I\triangleright$*.*

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

B Semantics of Causal Graphs

The knowledge of the abstract environment is formalized by the judgement $\mathcal{G} \vdash \mathsf{G}$, meaning that the environment can construct G given the knowledge of the terms in $\mathcal{G}$: this judgement is defined by a deductive system similar to the one of ρ-spi calculus and reported in Table 7. The environment knows every term sent on the network (OUT) and the special name $\mathscr{E}$ (ENV MSG), it can construct and destruct pairs (PAIR and PAIR DES) and encrypt and decrypt terms only knowing the required keys (ENCRYPTION and DECRYPTION). For reducing the number of terms known to the environment and abstracting the same set of ρ-spi terms, we prevent the environment from deriving labelled version of identities, public keys, enemy's keys and $\mathscr{E}$, which abstract the same ρ-spi terms as $\mathscr{E}$.

Function $bind_{\mathsf{abs}}$, reported in Table 7, defines the pattern-matching among terms. This function takes as input a term M and a ground term G and, if the two terms match, yields the the term G' obtained by labelling G according to the variables in M. If pattern-matching fails, $bind_{\mathsf{abs}}$ returns $\uparrow$. Function $bind_{\mathsf{abs}}$ is defined by cases on the structure of term M: a name matches itself and, similarly, the name $\mathscr{E}$ matches identities, public keys and enemy's keys; a variable can only be bound to an atomic name v; pattern-matching of pairs and ciphertexts is defined component-wise: notice that decryptions must be performed by the correct decryption key. In all the other cases, $bind_{\mathsf{abs}}$ returns failure $\uparrow$. Function $\overline{\mathsf{v}}$ yields the decryption key matching the encryption key v: this function is smoothly extended to arbitrary abstract terms G by inverting the encryption keys in G.

C Proofs of Soundness and Safety

This section proves the main results of our static analysis technique. For proving soundness results we need to refine the concretization function for terms. In particular, the definition in Table 3 says that the occurrences of the same name in an abstract term are instantiated by the same ρ-spi name. This definition can be soundly used for the concretization of paths and causal graphs into traces and configurations, respectively, but it is too precise for the concretization of terms in the knowledge of the environment. For instance, if the abstract environment knows $(\mathsf{n}^j, \mathsf{n}^j)$, where j is an index, then this means that the abstract name n is leaked out. In the ρ-spi calculus, this means that the environment may combine different protocol sessions, thus knowing terms like (n_1, n_2), where $\{n_1, n_2\} \subseteq \gamma_{\mathsf{trm}}(\mathsf{n})$ and the same abstract name is concretized into two different ρ-spi names. For tackling this problem, we relax the concretization of abstract (ground) terms as follows so as to concretize different occurrences of the same abstract name into the same ρ-spi name only when they occur within an integer component, which cannot be forged by the environment. Function $\gamma_{\mathsf{env}} : \mathscr{C}, \mathsf{G} \mapsto \{G_1, \ldots, G_n\}$ is defined as follows:

Table 7 Deductive System and Binding

Notation: In OUT, PAIR DES, DECRYPTION, $\nexists \mathsf{I}$ s.t. $\lceil \mathsf{G} \rceil \in \{\underline{\mathsf{I}}, \mathsf{k}_{\mathsf{I}}^{+}, \mathsf{k}_{\mathsf{E}}^{-}, \mathsf{k}_{\mathsf{IE}}, \mathsf{k}_{\mathsf{EI}}, \mathscr{E}\}$.
$\overline{\mathsf{v}}$ is defined as $\overline{\mathsf{n}} = \mathsf{n}$, $\overline{\mathsf{I}} = \mathsf{I}$, $\overline{\mathscr{E}} = \mathscr{E}$, $\overline{\mathsf{k}_{\mathsf{I}}^{+}} = \mathsf{k}_{\mathsf{I}}^{-}$ and $\overline{\mathsf{k}_{\mathsf{I}}^{-}} = \mathsf{k}_{\mathsf{I}}^{+}$.

$$\text{OUT}\ \frac{\mathsf{G} \in \mathcal{G}}{\mathcal{G} \vdash \mathsf{G}} \qquad \text{ENV MSG}\ \mathcal{G} \vdash \mathscr{E} \qquad \text{PAIR}\ \frac{\mathcal{G} \vdash \mathsf{G}_1 \qquad \mathcal{G} \vdash \mathsf{G}_2}{\mathcal{G} \vdash (\mathsf{G}_1, \mathsf{G}_2)} \qquad \text{PAIR DES}\ \frac{\mathcal{G} \vdash (\mathsf{G}_1, \mathsf{G}_2)}{\mathcal{G} \vdash \mathsf{G}_i \qquad \forall i \in [1,2]}$$

$$\text{ENCRYPTION}\ \frac{\mathcal{G} \vdash \mathsf{G} \qquad \mathcal{G} \vdash \mathsf{v}}{\mathcal{G} \vdash \{\mathsf{G}\}_{\mathsf{v}}} \qquad \text{DECRYPTION}\ \frac{\mathcal{G} \vdash \{\mathsf{G}\}_{\mathsf{v}} \qquad \mathcal{G} \vdash \mathsf{v}' \qquad \mathit{bind}_{\mathsf{abs}}(\mathsf{v}', \overline{\mathsf{v}}) \neq \uparrow \vee \mathit{bind}_{\mathsf{abs}}(\overline{\mathsf{v}'}, \mathsf{v}) \neq \uparrow}{\mathcal{G} \vdash \mathsf{G}}$$

Binding

$\mathit{bind}_{\mathsf{abs}}(\mathsf{v}_1, \mathsf{v}_2) = \mathsf{v}_2$	if either $\lceil \mathsf{v}_1 \rceil = \lceil \mathsf{v}_2 \rceil$ or $(\lceil \mathsf{v}_1 \rceil = \mathscr{E} \wedge \lceil \mathsf{v}_2 \rceil \in \{\mathsf{I}, \mathsf{k}_{\mathsf{I}}^{+}, \mathsf{k}_{\mathsf{IE}}, \mathsf{k}_{\mathsf{EI}}, \mathsf{k}_{\mathsf{E}}^{-}, \mathscr{E}\})$ or $(\lceil \mathsf{v}_2 \rceil = \mathscr{E} \wedge \lceil \mathsf{v}_1 \rceil \in \{\mathsf{I}, \mathsf{k}_{\mathsf{I}}^{+}, \mathsf{k}_{\mathsf{IE}}, \mathsf{k}_{\mathsf{EI}}, \mathsf{k}_{\mathsf{E}}^{-}, \mathscr{E}\})$
$\mathit{bind}_{\mathsf{abs}}(\mathsf{x}, \mathsf{v}) = \mathsf{v}_{(\mathsf{x})}$	
$\mathit{bind}_{\mathsf{abs}}((\mathsf{M}, \mathsf{M}'), (\mathsf{G}, \mathsf{G}')) = (\mathsf{G}_1, \mathsf{G}_2)$	if $\mathit{bind}_{\mathsf{abs}}(\mathsf{M}, \mathsf{G}) = \mathsf{G}_1 \wedge \mathit{bind}_{\mathsf{abs}}(\mathsf{M}', \mathsf{G}') = \mathsf{G}_2$
$\mathit{bind}_{\mathsf{abs}}(\{\mathsf{M}\}_{\mathsf{v}}, \{\mathsf{G}\}_{\mathsf{v}'}) = \{\mathsf{G}'\}_{\mathsf{v}'}$	if $\mathit{bind}_{\mathsf{abs}}(\mathsf{M}, \mathsf{G}) = \mathsf{G}' \wedge$ $(\mathit{bind}_{\mathsf{abs}}(\overline{\mathsf{v}}, \mathsf{v}') \neq \uparrow \vee \mathit{bind}_{\mathsf{abs}}(\mathsf{v}, \overline{\mathsf{v}'}) \neq \uparrow)$
$\mathit{bind}_{\mathsf{abs}}(\mathsf{M}, \mathsf{G}) = \uparrow$	otherwise

$$\gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}) \triangleq \begin{cases} \gamma_{\mathsf{trm}}(\mathsf{v}) & \text{if } \mathsf{G} = \mathsf{v} \\ \{(G_1, G_2) \mid G_1 \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}_1) \wedge G_2 \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}_2)\} & \text{if } \mathsf{G} = (\mathsf{G}_1, \mathsf{G}_2) \\ \{\{G\}_v \mid G \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}') \wedge v \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{v})\} & \text{if } \mathsf{G} = \{\mathsf{G}'\}_{\mathsf{v}} \\ & \wedge\ \mathit{output}(\mathit{nodes}(\mathscr{C})) \vdash \mathsf{v} \\ \gamma_{\mathsf{trm}}(\mathsf{G}) & \text{otherwise} \end{cases}$$

The following lemma says that $\gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G})$ contains $\gamma_{\mathsf{trm}}(\mathsf{G})$.

Lemma 1 (Term Concretization). *For every term* G *and causal graph* $\mathscr{C}$, $\gamma_{trm}(\mathsf{G}) \subseteq \gamma_{env}(\mathscr{C}, \mathsf{G})$.

Proof. We reason by induction on the structure of G.

Base Case The base case is $\mathsf{G} = \mathsf{v}$, which is trivial since, by definition of γ_{env}, we have that $\gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{v}) = \gamma_{\mathsf{trm}}(\mathsf{v})$ for any $\mathscr{C}$.

Inductive Step Let us suppose that $\mathsf{G} = (\mathsf{G}_1, \mathsf{G}_2)$, for some $\mathsf{G}_1, \mathsf{G}_2$. By definition of γ_{trm}, for every $(G_1, G_2) \in \gamma_{\mathsf{trm}}(\mathsf{G})$, there exists σ such that $(G_1, G_2) = (\mathsf{G}_1, \mathsf{G}_2)\sigma$ and thus $G_1 = \mathsf{G}_1\sigma$ and $G_2 = \mathsf{G}_2\sigma$. Therefore, we have that $G_1 \in \gamma_{\mathsf{trm}}(\mathsf{G}_1)$ and $G_2 \in \gamma_{\mathsf{trm}}(\mathsf{G}_2)$. By induction hypothesis, $G_1 \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}_1)$ and $G_2 \in \gamma_{\mathsf{trm}}(\mathscr{C}, \mathsf{G}_2)$ for any $\mathscr{C}$ and, by definition of γ_{env}, $(G_1, G_2) \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G})$, thus getting the result.
The proof for pairs and ciphertexts $\{\mathsf{G}'\}_{\mathsf{v}}$, where $\mathit{output}(\mathit{nodes}(\mathscr{C})) \vdash \mathsf{v}$, follows the same reasoning. If $\mathsf{G} = \{\mathsf{G}'\}_{\mathsf{v}}$ and $\mathit{output}(\mathit{nodes}(\mathscr{C})) \nvdash \mathsf{v}$ then, by definition of γ_{env}, $\gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}) = \gamma_{\mathsf{trm}}(\mathsf{G})$ for any $\mathscr{C}$, as desired.

Table 8 Result of $paths(graph(\mathsf{Prot}), \mathsf{end}^1_{\mathsf{n}}(\mathsf{A},\mathsf{B},\mathsf{m}), \emptyset)$

$$\left\{\begin{array}{l} \overbrace{new^{\mp}(\mathsf{k_A}) :: new(\mathsf{m}) :: new(\mathsf{n})}^{a} :: out(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+}) :: in(\mathsf{n}) \\ \overbrace{new^{\mp}(\mathsf{k_A}) :: in(\{\mathsf{B},\mathsf{n}_{(\mathsf{x})},\mathsf{m}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: begin^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{m}_{(\mathsf{z})})}^{b} :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ \overbrace{a :: out_{com}(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+},\{\mathsf{B}^{\mathsf{i}_1},\mathsf{n}^{\mathsf{i}_2}_{(\mathsf{x})},\mathsf{m}^{\mathsf{i}_3}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^{+\mathsf{i}_4}}) :: in(\{\mathsf{B},\mathsf{n}_{(\mathsf{x})},\mathsf{m}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: begin^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{m}_{(\mathsf{z})}) :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n})}^{c} \end{array}\right\}$$

$$\left\{\begin{array}{l} a :: out(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+}) :: in(\mathsf{n}) \\ \overbrace{new^{\mp}(\mathsf{k_A}) :: in(\{\mathsf{B},\mathsf{n}_{(\mathsf{x})},\mathscr{E}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: begin^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathscr{E}_{(\mathsf{z})})}^{d} :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: \overbrace{out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\{\mathsf{B},\mathsf{n}_{(\mathsf{x})},\mathscr{E}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: begin^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathscr{E}_{(\mathsf{z})})}^{e} :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: e :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \end{array}\right\}$$

$$\left\{\begin{array}{l} a :: out(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+}) :: in(\mathsf{n}) \\ d :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ \overbrace{new^{\mp}(\mathsf{k_A}) :: in(\{\mathscr{E},\mathsf{n}_{(\mathsf{x})},\mathsf{n}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: begin^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{n}_{(\mathsf{z})})}^{f} :: e :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: \overbrace{out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\{\mathsf{B},\mathsf{n}_{(\mathsf{x})},\mathsf{n}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: begin^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{n}_{(\mathsf{z})})}^{g} :: e :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: \overbrace{out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{z})}) :: in(\{\mathsf{B},\mathsf{n}_{(\mathsf{x})},\mathsf{n}_{(\mathsf{z})}\}_{\mathsf{k}_\mathsf{A}^-}) :: begin^1_{\mathsf{n}_{(\mathsf{x})}}(\mathsf{A},\mathsf{B},\mathsf{n}_{(\mathsf{z})})}^{i} :: e :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: g :: e :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: i :: e :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \end{array}\right\}$$

$$\left\{\begin{array}{l} a :: out(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+}) :: in(\mathsf{n}) \\ f :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \end{array}\right\} \quad \left\{\begin{array}{l} a :: out(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+}) :: in(\mathsf{n}) \\ f :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ d :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ d :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: e :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: e :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: e :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: e :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \end{array}\right\}$$

$$\left\{\begin{array}{l} a :: out(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+}) :: in(\mathsf{n}) \\ f :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ d :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ d :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: e :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: e :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \end{array}\right\} \quad \left\{\begin{array}{l} a :: out(\{\mathsf{B},\mathsf{n},\mathsf{m}\}_{\mathsf{k}_\mathsf{A}^+}) :: in(\mathsf{n}) \\ f :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ d :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ d :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ b :: e :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: g :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \\ c :: e :: i :: out_{com}(\mathsf{n}_{(\mathsf{x})},\mathsf{n}^{\mathsf{i}_5}_{(\mathsf{x})}) :: in(\mathsf{n}) \end{array}\right\}$$

The next lemma says that if there exists a binding between two ρ-spi terms, then there exists a binding between their abstractions.

Lemma 2 (Binding). *Let $\mathscr{C}$ be a causal graph, G and M ρ-spi terms, G and M abstract terms such that $G \in \gamma_{env}(\mathscr{C}, \mathsf{G})$, $M \in \gamma_{trm}(\mathsf{M})$ and $bind(M,G) \neq \uparrow$. Then $\exists \mathsf{G}'$ such that $bind_{\mathsf{abs}}(\mathsf{M},\mathsf{G}) = \mathsf{G}'$, $\sigma = subst(\mathsf{G}')$ and $\overline{G} \in \gamma_{trm}(\mathsf{M}\sigma)$.*

Proof. By induction on the structure of M.

Base Case We have two cases, depending on whether M is a variable or a name:

- Let $M = x$ and $G = v$. By definition of γ_{trm} and γ_{env}, $\mathsf{M} = \mathsf{x}$ and $\mathsf{G} = \mathsf{v}$, for some x and v such that $x \in \gamma_{\mathsf{trm}}(\mathsf{x})$ and $v \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{v}) = \gamma_{\mathsf{trm}}(\mathsf{v})$. By definition of $bind_{\mathsf{abs}}$ and *subst*, $bind_{\mathsf{abs}}(\mathsf{x},\mathsf{v}) = \mathsf{v}_{(\mathsf{x})}$ and $subst(\mathsf{v}_{(\mathsf{x})}) = [\lceil \mathsf{v} \rceil_{(\mathsf{x})}/\mathsf{x}]$. By definition of γ_{trm}, $v \in \gamma_{\mathsf{trm}}(\lceil \mathsf{v} \rceil_{(\mathsf{x})})$, as desired.
- Let $M = a$ and $G = a$. By definition of γ_{trm}, $\mathsf{M} = \mathsf{v}$, for some v such that $a \in \gamma_{\mathsf{trm}}(\mathsf{v})$. Similarly, by definition of γ_{env}, $\mathsf{G} = \mathsf{v}'$, for some v' such that $a \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{v}') = \gamma_{\mathsf{trm}}(\mathsf{v}')$. If $a \notin \mathcal{M}_{\mathcal{E}} \cup I\mathcal{D} \cup \mathcal{K}^{+} \cup \mathcal{K}_{\mathcal{E}}$, then $[\mathsf{v}] = [\mathsf{v}']$ and, by definition of $bind_{\mathsf{abs}}$, $bind_{\mathsf{abs}}(\mathsf{v},\mathsf{v}') = \mathsf{v}'$, as desired. If $a \in \mathcal{M}_{\mathcal{E}} \cup I\mathcal{D} \cup \mathcal{K}^{+} \cup \mathcal{K}_{\mathcal{E}}$, then v or v' may be the message $\mathscr{E}$ and the reasoning is similar.

Inductive step We have two cases, depending on whether M is a pair or a ciphertext:

- Let $M = (M_1, M_2)$ and $G = (G_1, G_2)$. By definition of γ_{trm}, there exist $\mathsf{M}_1, \mathsf{M}_2, \sigma'$ such that $\mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2)$ and $\mathsf{M}\sigma' = M$. By definition of γ_{env}, there exist $\mathsf{G}_1, \mathsf{G}_2$ such that $\mathsf{G} = (\mathsf{G}_1, \mathsf{G}_2)$, $G_1 \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}_1)$ and $G_2 \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}_2)$. By definition of function *bind*, $bind(M,G) = bind(M_1,G_1) \uplus bind(M_2,G_2)$. By induction hypothesis there exist $\mathsf{G}'_1 = bind_{\mathsf{abs}}(\mathsf{M}_1, \mathsf{G}_1)$ and $\mathsf{G}'_2 = bind_{\mathsf{abs}}(\mathsf{M}_2, \mathsf{G}_2)$ such that $\overline{G_1} \in \gamma_{\mathsf{trm}}(\mathsf{M}_1\sigma_1)$ and $\overline{G_2} \in \gamma_{\mathsf{trm}}(\mathsf{M}_2\sigma_2)$, where $\sigma_1 = subst(\mathsf{G}'_1)$ and $\sigma_2 = subst(\mathsf{G}'_2)$. Since $\mathsf{M}\sigma' = M$ and $bind(M,G) \neq \uparrow$, it turns out that for every $\mathsf{v} \in dom(\sigma_1) \cap dom(\sigma_2)$, $\sigma_1(\mathsf{v}) = \sigma_2(\mathsf{v})$. Thus $\sigma = \sigma_1 \uplus \sigma_2 \neq \uparrow$ and $(G_1, G_2) \in \gamma_{\mathsf{trm}}((\mathsf{M}_1, \mathsf{M}_2)\sigma)$, as desired.
- Let $M = \{M'\}_{a_1}$ and $G = \{G'\}_{a_2}$, with $\overline{a_1} = a_2$. By definition of γ_{trm}, there exists $\mathsf{M}', \mathsf{v}_1, \sigma''$ such that $\mathsf{M} = \{\mathsf{M}'\}_{\mathsf{v}_1}$ and $\mathsf{M}\sigma'' = M$. By definition of γ_{env}, there exists $\mathsf{G}', \mathsf{v}_2$ such that $\mathsf{G} = \{\mathsf{G}'\}_{\mathsf{v}_2}$. We have two cases: either $output(nodes(\mathscr{C})) \nvdash \mathsf{v}_2$, and thus $\{G'\}_{a_2} \in \gamma_{\mathsf{trm}}(\{\mathsf{G}'\}_{\mathsf{v}_2})$, or $output(nodes(\mathscr{C})) \vdash \mathsf{v}_2$, and thus $G' \in \gamma_{\mathsf{env}}(\mathsf{G}')$ and $a_2 \in \gamma_{\mathsf{trm}}(\mathsf{v}_2)$. In the former case the result is immediate. In the latter case, the proof is similar to the one for pairs. Notice that if $\overline{a_1} = a_2$ then either $bind_{\mathsf{abs}}(\overline{\mathsf{v}_1}, \mathsf{v}_2) \neq \uparrow$ or $bind_{\mathsf{abs}}(\mathsf{v}_1, \overline{\mathsf{v}_2}) \neq \uparrow$.

The following lemma states that causal graphs are a sound abstraction of ρ-spi configurations as far as secrecy is concerned. In particular, if a message G is known to the environment associated with the ρ-spi configuration $\langle s, P \rangle$, namely $s \vdash G$, then an abstraction G of G is known to the environment in the causal graph abstracting $\langle s, P \rangle$. Furthermore, every integer component in G is associated with an output in s and the instantiation of messages occurring therein is consistent.

Lemma 3 (Abstract Environment). *Suppose $\langle s, P \rangle \in \gamma_{net}(\mathscr{C})$ and $s \vdash G$. Then $\exists \mathsf{G}$ such that the following statements hold:*

1. $output(nodes(\mathscr{C})) \vdash \mathsf{G}$ *and* $G \in \gamma_{env}(\mathscr{C}, \mathsf{G})$
2. *if* $bind(G,M) \neq \uparrow$ *and* $M \in \gamma_{trm}(\mathsf{M})$, *then* $\exists \mathsf{G}_{lab}$ *such that* $bind_{\mathsf{abs}}(\mathsf{G},\mathsf{M}) = \mathsf{G}_{lab}$, $\sigma = subst(\mathsf{G}_{lab})$ *and* $\overline{G} \in \gamma_{trm}(\mathsf{M}\sigma)$
3. $\forall \mathsf{G}' \in int(\mathscr{C}, \mathsf{G}_{lab}), \exists \mathsf{out}(\mathsf{G}'').\mathsf{P}'' \in nodes(\mathscr{C})$ *and* G'' *such that*
   (a) $[\mathsf{G}'] \in terms(\lfloor \mathsf{G}'' \rfloor)$
   (b) $s = t_1 :: \ldots :: t_{r-1} :: out(G'') :: t_{r+1} :: \ldots :: t_n$
   (c) $\forall i \in [1,n], \exists \mathsf{Q}_i \in nodes(\mathscr{C}), \mathcal{S}_i \in paths(\mathscr{C}, \mathsf{Q}_i, \emptyset), S_i$ *s.t.* $S_i \in \gamma_{pset}(\mathcal{S}_i) \wedge$
   $(\mathcal{S}_i = \mathcal{S}_j :: \mathsf{t}_i \wedge S_i = S_j :: t_i$, *with* $j < i$,
   $\vee \ (\mathcal{S}_i = \bigcup_{j=1,k} \mathcal{S}_{i'_j} :: out_{com}(\mathsf{G}_j, \mathsf{G}'_j) :: in(\mathsf{G}) \cup \mathcal{S}_q :: in(\mathsf{G}) \ \wedge \ \mathcal{S}_{i_j} = \mathcal{S}_{i'_j} ::$
   $out(\mathsf{G}_j) \wedge$
   $S_i = \bigcup_{j=1,k} S_{i_j} :: t_i \cup S_q :: t_i$ *with* $i'_j < i_j$ *and* $i_j, q \leq i))$
   (d) $\mathsf{Q}_r \in threads(\mathsf{P}'')$
   (e) *Let* σ_1, σ_2 *be two substitutions such that* $\overline{G} = (\mathsf{M}\sigma)\sigma_1$ *and* $G'' = \mathsf{G}''\sigma_2$. *If* $\sigma_1(\mathsf{v}) = n$ *and* $\mathsf{v}^{(\mathsf{out}(\mathsf{G}'').\mathsf{P}, j)} \in terms(\mathsf{G}')$, *then* $\sigma_2(\pi^j(\mathsf{G}'')) = n$

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

Base Case We have four cases depending on the derivation rule applied (OUT, PUBLIC KEYS, ENEMY KEYS or ENV):

OUT The judgement $s \vdash G$ is proved by OUT and $s = t_1 :: \ldots :: t_{r-1} :: out(G) :: \ldots :: t_{r+1} :: \ldots :: t_n$. Since $\langle s, P \rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, the first condition in the definition of γ_{net} proves item 3c. By definition of function *paths*, we have that $\mathsf{out}(\mathsf{G}).\mathsf{P} \in nodes(\mathscr{C})$ and $\mathsf{Q}_r \in threads(\mathsf{P})$, with $G \in \gamma_{\mathsf{trm}}(\mathsf{G})$. Therefore $output(nodes(\mathscr{C})) \vdash \mathsf{G}$ and, by Lemma 1, $G \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G})$, thus proving item 1. By Lemma 2, item 2 holds as well. Finally, item 3 is trivially satisfied since all integer components in G are output by the same process $\mathsf{out}(\mathsf{G}).\mathsf{P}$.

PUBLIC KEYS, ENEMY KEYS, ENV By ENV MSG $output(nodes(\mathscr{C})) \vdash \mathscr{E}$ and $a \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathscr{E})$, whenever a is a public key, an enemy key, an identity or a message generated by the environment. Item 2 is proved by Lemma 2. The proof of the third item is trivial since, for any v such that $[\mathsf{v}] = \mathscr{E}$, $int(\mathscr{C}, \mathsf{v}) = \emptyset$.

Inductive Step We have four cases according to the last derivation rule applied (PAIR, PAIR DES, DECRYPTION or ENCRYPTION):

PAIR, PAIR DES, DECRYPTION The thesis is straightforwardly proved by induction hypothesis.

ENCRYPTION Let us suppose that $G = \{G'\}_{\mathsf{v}'}$, with $s \vdash G'$ and $s \vdash \mathsf{v}'$. By induction hypothesis, there exist G' and v' such that $G' \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}')$, $\mathsf{v}' \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{v}')$, $output(nodes(\mathscr{C})) \vdash \mathsf{G}'$ and $output(nodes(\mathscr{C})) \vdash \mathsf{v}'$. By ENCRYPTION, we have that $output(nodes(\mathscr{C})) \vdash \{\mathsf{G}'\}_{\mathsf{v}'}$ and, by definition of γ_{env}, we get $\{G'\}_{\mathsf{v}'} \in \gamma_{\mathsf{env}}(\mathscr{C}, \{\mathsf{G}'\}_{\mathsf{v}'})$, thus proving item 1. Item 2 is proved by Lemma 2. Let $\mathsf{G}_{lab} = \{\mathsf{G}_1\}_{\mathsf{v}_1}$: since $output(nodes(\mathscr{C})) \vdash \mathsf{v}_1$, $int(\mathscr{C}, \{\mathsf{G}_1\}_{\mathsf{v}_1}) = int(\mathscr{C}, \mathsf{G}_1) \cup int(\mathscr{C}, \mathsf{v}_1)$ and item 3 is trivially proved by induction hypothesis.

The next lemma says that the occurrences of a session message in a path are instantiated by the same ρ-spi message.

Lemma 4 (Session Messages). *Let* s *be a path. For every* $\mathsf{v} \in sm(\mathsf{s})$*, there exists i s.t.*

1. $\pi^i(\mathsf{s}) = new(\lfloor \mathsf{v} \rfloor)$ *and,*
2. *For every* $s \in \gamma_{path}(\mathsf{s})$*, there exists n such that* $\pi^i(s) = new(n)$ *and* $\sigma(\mathsf{v}) = n$*, with* $\sigma = bsm(s, \mathsf{s})$.

Proof. By induction on the length of s. If s is the empty trace, then $sm(\mathsf{s}) = \emptyset$ and the thesis is trivially satisfied. Let us suppose that s is not the empty trace and proceed by cases on the last action:

$\mathsf{s} = \mathsf{s}'::new(\mathsf{m})$ By definition 7,$sm(\mathsf{s}) = sm(\mathsf{s}') \cup \{\mathsf{m}\}$: the thesis derives directly from the induction hypothesis and the definition of bsm.

$\mathsf{s} = \mathsf{s}'::out_{com}(\mathsf{G}_1, \mathsf{G}_2) :: in(\mathsf{G})$ By definition 7, we have that $sm(\mathsf{s}) = bound(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2) \cup bymatch(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2)$. By definition of function *bound*, for every $\mathsf{v} \in bound(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2)$, there exist $\mathsf{v}', \mathsf{P}, j$ such that $\mathsf{v}^{(\mathsf{P},\mathsf{j})} \in \mathsf{msgs}(\mathsf{G}_2)$, $\pi^j(\mathsf{G}_1) = \mathsf{v}'$ and $\mathsf{v}' \in sm(\mathsf{s}')$. Similarly, by definition of function *bymatch*, for every $\mathsf{v} \in bymatch(\mathsf{s}', \mathsf{G}_1, \mathsf{G}_2)$, there exists some prefix s'' of s' such that $\mathsf{v} \in sm(\mathsf{s}'')$. The thesis follows straightforwardly from the induction hypothesis and the definition of bsm.

$\mathsf{s} = \mathsf{s}'::\mathsf{t}$ (otherwise) By definition 7, $sm(\mathsf{s}) = sm(\mathsf{s}')$: the thesis derives directly from the induction hypothesis and the definition of bsm.

The next definition introduces the notion of path set approximation, which is used in Lemma 5 for proving that cycles may be soundly abstracted away in cycle-invariant causal graphs.

Definition 12 (Path set approximation). *We say that* $\mathcal{S}'$ *is an approximation of* $\mathcal{S}$*, written* $\mathcal{S} \preceq \mathcal{S}'$*, if and only if*

1. *for every* $\mathsf{s}' :: out_{com}(\mathsf{G}_1, \mathsf{G}_2) :: in(\mathsf{G}) \in \mathcal{S}'$ *there exists* $\mathsf{s} \in \mathcal{S}$ *such that:*
   - *either* $\mathsf{s} = \mathsf{s}'$
   - *or* $\mathsf{s} = \mathsf{s}'' :: out_{com}(\mathsf{G}_1, \mathsf{G}_3) :: in(\mathsf{G}) :: \mathsf{s}''$*, for some* s''*, and* $sm(\mathsf{s}') \cap \mathsf{msgs}(\mathsf{G}_1) \subseteq sm(\mathsf{s})$.
2. *for every* $\mathsf{s}' :: in(\mathsf{G}) \in \mathcal{S}'$*, with* $\mathsf{s}' \neq \mathsf{s}'' :: out_{com}(\mathsf{G}_1, \mathsf{G}_2)$*, there exists* $\mathsf{s} \in \mathcal{S}$ *such that:*
   - *either* $\mathsf{s} = \mathsf{s}'$
   - *or* $\mathsf{s} = \mathsf{s}'' :: in(\mathsf{G}) :: \mathsf{s}''$*, for some* s''*, and* $sm(\mathsf{s}') \subseteq sm(\mathsf{s})$.

Lemma 5 (Cycle-invariant causal graphs). *Let* $\mathscr{C}$ *be a cycle-invariant causal graph and* $\mathsf{out}(\mathsf{G}_1).\mathsf{P}_1 \xrightarrow{\mathsf{G}'_1} \mathsf{com}(\mathsf{G})$, $\mathsf{com}(\mathsf{G}) \to \mathsf{Q}$ *and* $\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})$ *be edges in* $\mathscr{C}$.

For every $\mathcal{S} \in paths(\mathscr{C}, \mathsf{out}(\mathsf{G}_1).\mathsf{P}, \emptyset)$ *there exists* $\mathcal{S}' \in paths(\mathscr{C}, \mathsf{Q}, \emptyset)$ *such that* $\mathcal{S}' \preceq \mathcal{S}$.
For every $\mathcal{S} \in paths(\mathscr{C}, \mathsf{in}(\mathsf{M}).\mathsf{P}, \emptyset)$ *there exists* $\mathcal{S}' \in paths(\mathscr{C}, \mathsf{Q}, \emptyset)$ *such that* $\mathcal{S}' \preceq \mathcal{S}$.

Proof. We prove the first item as the proof for the second one is similar. Let us suppose that there exists a cycle in the causal graph containing the edge $\mathsf{in}(\mathsf{M}).\mathsf{P} \to \mathsf{com}(\mathsf{G})$. Otherwise, the thesis is trivially satisfied. More formally, there exists $\mathcal{S} \in paths(\mathscr{C}, \mathsf{out}(\mathsf{G}_1).\mathsf{P}, \emptyset)$ such that $\mathcal{S}' = \{\mathsf{s} \in \mathcal{S} \mid \exists \mathsf{s}_1, \mathsf{s}_2 \text{ s.t. } \mathsf{s} = \mathsf{s}_1 :: out_{com}(\mathsf{G}_1, \mathsf{G}'_1) :: in(\mathsf{G}) :: \mathsf{s}_2\} \neq \emptyset$.

By definition of function *paths*, $\mathsf{s}_1 :: \mathit{out}_{com}(\mathsf{G}_1, \mathsf{G}_1') :: \mathit{in}(\mathsf{G}) :: \mathsf{s}_2 \in \mathcal{S}'$ if and only if there exists $\mathcal{E}, \mathcal{S}'' \in \mathit{paths}(\mathscr{C}, \mathsf{Q}, \mathcal{E})$ such that $\mathsf{s}_1 :: \mathit{out}_{com}(\mathsf{G}_1, \mathsf{G}_1') :: \mathit{in}(\mathsf{G}) \in \mathcal{S}''$. By definition of function *paths*, $\mathcal{S}'' \in \mathit{paths}(\mathscr{C}, \mathsf{Q}, \emptyset)$. Since $\mathcal{G}$ is cycle-invariant, there exists $\mathsf{s}' = \mathsf{s}_1 :: \mathit{out}(\mathsf{G}_1, \mathsf{G}_1'') :: \mathit{in}(\mathsf{G}) :: \mathsf{s}_2' \in \mathcal{S}'$ such that $\mathsf{s}_2 \sim \mathsf{s}_2'$ and $\mathit{sm}(\mathsf{s}_1) \cap \mathsf{msgs}(\mathsf{G}) \subseteq \mathit{sm}(\mathsf{s}')$, thus proving the thesis.

The next proposition states that if a configuration is an instance of a causal graph guaranteeing authenticity, the trace in the configuration guarantees weak authenticity. In the following, we write $t_1 :: \ldots :: t_n \sqsubseteq s$ to say that there exist $s_1, \ldots, s_{n+1}$ such that $s = s_1 :: t_1 :: \ldots :: s_n :: t_n :: s_{n+1}$.

Lemma 6 (Authenticity). *If $\langle s, P \rangle \in \gamma_{net}(\mathscr{C})$ and $\mathscr{C}$ guarantees authenticity, then s guarantees weak authenticity.*

Proof. By condition (i) in the definition of γ_{net}, if $s = s_1 :: \mathit{end}_N^i(A, B, M) :: s_2$ then there exist $\mathsf{Q} \in \mathit{nodes}(\mathscr{C})$, $\mathcal{S} \in \mathit{paths}(\mathscr{C}, \mathsf{Q}, \emptyset)$ such that, for every $\mathsf{s} \in \mathcal{S}$, there exists $s' \sqsubseteq s_1$ such that $s' :: \mathit{end}_N^i(A, B, M) \in \gamma_{\mathsf{path}}(\mathsf{s})$. Since the causal graph guarantees authenticity, there exist N, M such that $\mathsf{s}' :: \mathit{end}_{\mathsf{N}}^i(\mathsf{A}, \mathsf{B}, \mathsf{M}) \in \mathcal{S}$ guarantees authenticity. Thus there exist $\mathsf{N}', \mathsf{M}', \mathsf{s}_1', \mathsf{s}_2'$ such that $\mathsf{s}' = \mathsf{s}_1' :: \mathit{begin}_{\mathsf{N}'}^i(\mathsf{A}, \mathsf{B}, \mathsf{M}') :: \mathsf{s}_2'$, $\lfloor \mathsf{N} \rfloor = \lfloor \mathsf{N}' \rfloor$, $\lfloor \mathsf{M} \rfloor = \lfloor \mathsf{M}' \rfloor$, $\mathsf{msgs}(\mathsf{M}', \mathsf{N}') \subseteq \mathit{sm}(\mathsf{s}_1')$ and $\mathsf{msgs}(\mathsf{M}, \mathsf{N}) \subseteq \mathit{sm}(\mathsf{s}')$. By Lemma 4, $\mathit{begin}_N^i(A, B, M) \in s'$ as desired.

Theorem 1 (Soundness) If $\mathit{graph}(P) = \mathscr{C}$, then $\langle \varepsilon, P \rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$.

Proof. Trivial, by definition of γ_{net} and by observing that the trace in the configuration is empty.

The preservation theorem states that the set of configurations abstracted by a causal graph is closed under process reduction. This means that causal graphs are a sound model for statically reasoning on security properties of ρ-spi processes.

Theorem 2 (Preservation) Let $\mathscr{C}$ be a cycle-invariant graph. If $\langle s, P \rangle \rightarrow \langle s', P' \rangle$ and $\langle s, P \rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, then $\langle s', P' \rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$.

Proof. The proof proceeds by induction on the length of the derivation for $\langle s, P \rangle \rightarrow \langle s', P' \rangle$: if P is a parallel composition or a process with identifier this may take several steps. The base cases are given by the remaining transition rules. Since $\langle s, P \rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, with $s = t_1 :: \ldots :: t_n$, we have that

1. $\forall i \in [1, n], \exists \mathsf{Q}_i \in \mathit{nodes}(\mathscr{C}), \mathcal{S}_i \in \mathit{paths}(\mathscr{C}, \mathsf{Q}_i, \emptyset), S_i$ s.t. $S_i \in \gamma_{\mathsf{pset}}(\mathcal{S}_i) \wedge$
   $(\mathcal{S}_i = \mathcal{S}_j :: \mathsf{t}_j \wedge S_i = S_j :: t_j$, with $j < i$,
   $\vee\ (\mathcal{S}_i = \bigcup_{j=1,k} \mathcal{S}_{i_j'} :: \mathit{out}_{com}(\mathsf{G}_j, \mathsf{G}_j') :: \mathit{in}(\mathsf{G}) \cup \mathcal{S}_r :: \mathit{in}(\mathsf{G}) \wedge \mathcal{S}_j = \mathcal{S}_{i_j'} :: \mathit{out}(\mathsf{G}_j) \wedge$
   $S_i = \bigcup_{j=1,k} S_{i_j} :: t_i \cup S_r :: t_i$ with $i_j' < i_j$ and $i_j, r \leq i))$
2. $\forall Q \in \mathit{threads}(P),\ \exists i$ s.t. $Q = \mathsf{Q}_i\sigma,$ with $\sigma \in \mathsf{ext}(\mathsf{bsm}(S_i, \mathcal{S}_i))$

New $\langle s, \mathsf{new}(n).P\rangle \rightarrow \langle s :: new(n), P\rangle$

For proving $\langle s :: new(n), P\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, we have to prove the two conditions in the definition of γ_{net}.

Condition *(i)* Since $\langle \mathsf{new}(n).P, s\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, there exist $i \in [1,n]$ and $\mathsf{Q}_i = \mathsf{new(n).P} \in nodes(\mathscr{C})$ such that $\mathsf{new}(n).P = \mathsf{new(n).P}\sigma$, with $\sigma \in \mathsf{ext}(\mathsf{bsm}(S_i, \mathcal{S}_i))$. By definition of causal graph, if $\mathsf{new(n).P} \in nodes(\mathscr{C})$, then there exists $\mathsf{Q} \in nodes(\mathscr{C})$ such that $\mathsf{Q} \in threads(\mathsf{P})$ and $\mathsf{new(n).P} \rightarrow \mathsf{Q} \in edges(\mathscr{C})$. By definition of function *paths*, $\mathcal{S}_i :: new(\mathsf{n}) \in paths(\mathscr{C}, \mathsf{Q}, \emptyset)$. By definition of function *sm* and function bsm, $\mathsf{bsm}(\mathcal{S}_i :: new(\mathsf{n}), S_i :: new(n)) = \mathsf{bsm}(\mathcal{S}_i, S_i) \cup [n/\mathsf{n}]$. By function γ_{pset}, $S_i :: new(n) \in \gamma_{\mathsf{pset}}(\mathcal{S}_i :: new(\mathsf{n}))$, thus proving condition (i).

Condition *(ii)* Since $\mathsf{new}(n).P = \mathsf{new(n).P}\sigma$, we get $P = \mathsf{P}\sigma'$, with $\sigma' \in \mathsf{ext}(\mathsf{bsm}(\mathcal{S}_i, S_i) \cup [n/\mathsf{n}])$, as desired.

Thus $\langle s :: new(n), P\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, as desired. The reasoning for key-pair restrictions is similar.

Struct $\langle s, p.P\rangle \rightarrow \langle s :: p, P\rangle$,
with $p \in \{\mathsf{out}(G), \mathsf{begin}^i_{G_1}(A,I,G_2), \mathsf{end}^i_{G_1}(A,I,G_2)\}$

As before, we prove distinctly the two conditions for $\langle s :: p, P\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$.

Condition*(i)* Since $\langle s, p.P\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, there exist $i \in [1,n]$ and $\mathsf{Q}_i = \mathsf{p.P} \in nodes(\mathscr{C})$ such that $p.P = \mathsf{p.P}\sigma$, with $\sigma \in \mathsf{ext}(\mathsf{bsm}(S_i, \mathcal{S}_i))$. By definition of causal graph, if $\mathsf{p.P} \in nodes(\mathscr{C})$, then there exists $\mathsf{Q} \in nodes(\mathscr{C})$ such that $\mathsf{Q} \in threads(\mathsf{P})$ and $\mathsf{p.P} \rightarrow \mathsf{Q} \in edges(\mathscr{C})$. By definition of function *paths*, $\mathcal{S}_i :: \mathsf{p} \in paths(\mathscr{C}, \mathsf{Q}, \emptyset)$. By definition of function *sm* and function bsm, $\mathsf{bsm}(\mathcal{S}_i :: \mathsf{p}, S_i :: p) = \mathsf{bsm}(\mathcal{S}_i, S_i)$. Since $p.P = \mathsf{p.P}\sigma$, we have that $p = \mathsf{p}\sigma$: by definition of function γ_{pset}, $S_i :: p \in \gamma_{\mathsf{pset}}(\mathcal{S}_i :: \mathsf{p})$, thus proving condition (i).

Condition *(ii)* Since $p.P = \mathsf{p.P}\sigma$, we have that $P = \mathsf{P}\sigma$, as desired.

Thus $\langle s :: p, P\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, as desired.

In $\langle s, \mathsf{in}(M).P\rangle \rightarrow \langle s :: in(M\sigma), P\sigma\rangle$, with $\sigma = bind(M,G)$

Since $\langle s, \mathsf{in}(M).P\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, by condition (ii) there exist M and P such that $\mathsf{in(M).P} \in nodes(\mathscr{C})$ and $M \in \gamma_{\mathsf{trm}}(\mathsf{M})$. In the following, we reason on an arbitrary $Q \in threads(P\sigma)$. By Lemma 3, if $s \vdash G$, then there exists a term G such that

1. $output(nodes(\mathscr{C})) \vdash \mathsf{G}$ and $G \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G})$
2. $\exists \sigma, \mathsf{G}_{lab}$ such that $bind_{\mathsf{abs}}(\mathsf{G}, \mathsf{M}) = \mathsf{G}_{lab}$, $\sigma' = subst(\mathsf{G}_{lab})$ and $\overline{G} \in \gamma_{\mathsf{trm}}(\mathsf{M}\sigma')$
3. $\forall \mathsf{G}' \in int(\mathscr{C}, \mathsf{G}_{lab})$, $\exists \mathsf{out(G'').P''} \in nodes(\mathscr{C})$, G'' and S such that
   (a) $\lfloor \mathsf{G}' \rfloor \in terms(\lfloor \mathsf{G}'' \rfloor)$
   (b) $s = t_1 :: \ldots :: t_{r-1} :: out(G'') :: t_{r+1} :: \ldots :: t_n$

(c) $\forall i \in [1,n], \exists \mathsf{Q}_i \in \mathit{nodes}(\mathscr{C}), \mathcal{S}_i \in \mathit{paths}(\mathscr{C}, \mathsf{Q}_i, \emptyset), S_i$ s.t. $S_i \in \gamma_{\mathsf{pset}}(\mathcal{S}_i) \wedge$
$(\mathcal{S}_i = \mathcal{S}_j :: \mathsf{t}_j \wedge S_i = S_j :: t_j$, with $j \leq i$,
$\vee\ (\mathcal{S}_i = \bigcup_{j=1,k} \mathcal{S}'_j :: \mathit{out}_{com}(\mathsf{G}_j, \mathsf{G}'_j) :: \mathit{in}(\mathsf{G}) \cup \mathcal{S}_q :: \mathit{in}(\mathsf{G}) \wedge \mathcal{S}_j = \mathcal{S}'_j :: \mathit{out}(\mathsf{G}_j) \wedge$
$S_i = \bigcup_{j=1,k} S_j :: t_i \cup S_q :: t_i$ with $j,q \leq i))$

(d) $\mathsf{Q}_r \in \mathit{threads}(\mathsf{P}'')$

(e) Let σ_1, σ_2 such that $G = (\mathsf{M}\sigma')\sigma_1$ and $G'' = \mathsf{G}''\sigma_2$. If $\sigma_1(\mathsf{v}) = n$ and $\mathsf{v}^{(\mathsf{out}(\mathsf{G}''), \mathsf{P}, j)} \in \mathit{terms}(\mathsf{G}')$, then $\sigma_2(\pi^j(\mathsf{G}'')) = n$

By definition of causal graph, there exists $\mathsf{Q} \in \mathit{nodes}(\mathscr{C})$ such that $\mathsf{Q} \in \mathit{threads}(\mathsf{P}\sigma')$, $\mathsf{in}(\mathsf{M}).\mathsf{P} \rightarrow \mathsf{com}(\mathsf{G}) \in \mathit{edges}(\mathscr{C})$ and $\mathsf{com}(\mathsf{G}) \rightarrow \mathsf{Q} \in \mathit{edges}(\mathscr{C})$. As before, we prove the two conditions for $\langle s :: \mathit{in}(M\sigma), P\sigma \rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$.

Condition *(i)* Since $\langle s, \mathsf{in}(M).P \rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, by condition (ii) applied on the source configuration, there exist $\mathcal{S}_i \in \mathit{paths}(\mathscr{C}, \mathsf{in}(\mathsf{M}).\mathsf{P}, \emptyset)$, S_i such that $S_i \in \gamma_{\mathsf{pset}}(\mathcal{S}_i)$ and $\mathsf{in}(M).P = \mathsf{in}(\mathsf{M}).\mathsf{P}\sigma_i$, with $\sigma_i \in \mathsf{ext}(\mathsf{bsm}(S_i, \mathcal{S}_i))$. By definition of function bsm, $\mathsf{bsm}(S_i, \mathcal{S}_i) = \mathsf{bsm}(S_i :: \mathit{in}(M\sigma), \mathcal{S}_i :: \mathit{in}(\mathsf{M}\sigma'))$ and thus $S_i :: \mathit{in}(M\sigma) \in \gamma_{\mathsf{pset}}(\mathcal{S}_i :: \mathit{in}(\mathsf{M}\sigma'))$.
We now reason on the arbitrary $\mathsf{G}' \in \mathit{int}(\mathscr{C}, \mathsf{G}_{lab})$ of item 3 and we prove $S_r :: \mathit{in}(M\sigma) \in \gamma_{\mathsf{pset}}(\mathcal{S}_{r'} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma'))$, where $\mathcal{S}_r = \mathcal{S}_{r'} :: \mathit{out}(\mathsf{G}'')$, with $r' < r$.
By definition of γ_{net}, $S_r \in \gamma_{\mathsf{pset}}(\mathcal{S}_r)$. By item 3e of Lemma 3, the instantiation of the session messages yielded by $\mathit{bound}(\mathsf{s}, \mathsf{G}'', \mathsf{G}')$ is consistent and this suffices for proving that $S_r :: \mathit{in}(M\sigma) \in \gamma_{\mathsf{pset}}(\mathcal{S}_{r'} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma'))$.
Notice that $\mathcal{S}_{r'} \in \mathit{paths}(\mathscr{C}, \mathsf{out}(\mathsf{G}'').\mathsf{Q}_r, \emptyset)$ and it might be the case that $\mathcal{S}_{r'} \notin \mathit{paths}(\mathscr{C}, \mathsf{out}(\mathsf{G}'').\mathsf{Q}_r, \{\mathsf{in}(\mathsf{M}).\mathsf{P} \rightarrow \mathsf{com}(\mathsf{G})\})$. However, by Lemma 5, we can take some $\mathcal{S}' \in \mathit{paths}(\mathscr{C}, \mathsf{Q}, \emptyset)$ such that $\mathcal{S}' \preceq \mathcal{S}_{r'}$. By Definition 12, for every $\mathsf{s} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma') \in \mathcal{S}'$, either $\mathsf{s} \in \mathcal{S}_{r'}$ or $\mathsf{s} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}''') :: \mathit{in}(\mathsf{M}\sigma') :: \mathsf{s}' \in \mathcal{S}_{r'}$ and $\mathit{sm}(\mathsf{s}) \cap \mathsf{msgs}(\mathsf{G}'') \subseteq \mathit{sm}(\mathsf{s} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma') :: \mathsf{s}')$. Thus we have that $S_r :: \mathit{in}(G) \in \gamma_{\mathsf{pset}}(\mathcal{S}_{r'} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma'))$.
Let us fix $\sigma'_i = \mathsf{bsm}(S_i :: \mathit{in}(G), \mathcal{S}_i :: \mathit{in}(\mathsf{M}\sigma'))$ and $\sigma'_r = \mathsf{bsm}(S_r :: \mathit{in}(G), \mathcal{S}_{r'} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma'))$. Suppose by contradiction that there exists v such that $\sigma'_i(\mathsf{v}) \neq \sigma'_r(\mathsf{v})$. Since $G \in \gamma_{\mathsf{trm}}(\mathsf{M}\sigma')$, it must be the case that $\mathsf{v} \notin \mathsf{msgs}(\mathsf{M}\sigma')$. Thus there exists $\mathsf{s}_i \in \mathcal{S}_i$ such that $\mathsf{v} \in \mathit{sm}(\mathsf{s}_i)$ and $\mathsf{s}_r \in \mathcal{S}'_r$ such that $\mathsf{v} \in \mathit{bymatch}(\mathsf{s}_r, \mathsf{G}'', \mathsf{G}')$. By definition of function *bymatch*, there exists $\mathsf{n} \in \mathit{sm}(\mathsf{s}_r :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma'))$ such that $\mathsf{s}_r = \mathsf{s}'_r :: \mathit{new}(\mathsf{n}) :: \mathsf{s}''_r$ and $\mathsf{v} \in \mathit{sm}(\mathsf{s}'_r)$. Since $\mathsf{v} \in \mathit{sm}(\mathsf{s}_i)$, $\mathsf{n} \in \mathit{sm}(\mathsf{s}_i :: \mathit{in}(\mathsf{M}\sigma'))$. Lemma 4 proves that $\mathsf{s}_i = \mathsf{s}'_i :: \mathit{new}(\mathsf{n}) :: \mathsf{s}''_i$. Since $G \in \gamma_{\mathsf{trm}}(\mathsf{M}\sigma')$, $\sigma_r(\mathsf{n}) = \sigma_i(\mathsf{n})$. By definition of γ_{net}, $\mathsf{s}'_r :: \mathit{new}(\mathsf{n}) :: \mathsf{s}''_i \in \mathcal{S}_i$ and $\sigma_r(\mathsf{v}) = \sigma_i(\mathsf{v})$, thus giving a contradiction. Thus the instantiation of the session messages is consistent and by similar reasoning on the other integer components of G_{lab} we get the result.

Condition *(ii)* Let σ_{n+1} be the substitution obtained by the union of $\mathsf{bsm}(S_i :: \mathit{in}(G), \mathcal{S}_i :: \mathit{in}(\mathsf{M}\sigma'))$ (intra-thread causality) and $\mathsf{bsm}(S_r :: \mathit{in}(G), \mathcal{S}_{r'} :: \mathit{out}_{com}(\mathsf{G}'', \mathsf{G}') :: \mathit{in}(\mathsf{M}\sigma'))$ (inter-thread causality). We have to prove that $Q = \mathsf{Q}\sigma'_{n+1}$, with $\sigma'_{n+1} \in \mathsf{ext}(\sigma_{n+1})$.
The proof follows by observing that $\mathsf{in}(M).P = \mathsf{in}(\mathsf{M}).\mathsf{P}\sigma'_i$, with $\sigma'_i \in \mathsf{ext}(\sigma_i)$, $\overline{G} \in (\mathsf{M}\sigma')\sigma'_{n+1}$ and $\sigma_i \subseteq \sigma_{n+1}$.

Thus $\langle s :: in(G), P\sigma\rangle \in \gamma_{\mathsf{net}}(\mathscr{C})$, as desired.

The proof for the replication is straightforward since $\langle s, !P\rangle \rightarrow \langle s, P|!P\rangle$ and $threads(P) = threads(P|!P)$. The proof for the inductive step follows straightforwardly from the induction hypothesis.

Finally, we give the theorems stating that causal graphs constitute an abstract model where secrecy and authenticity can be soundly verified. The next theorem says that if $\mathscr{C}$ is the causal graph associated with P and $\mathscr{C}$ guarantees the secrecy of v, then P guarantees the secrecy of any concretization of v.

Theorem 3 (Secrecy) Let P be a process and $\mathscr{C}$ a cycle-invariant causal graph s.t. $\mathscr{C} = graph(P)$. If $\mathscr{C}$ guarantees the secrecy of v then P guarantees the secrecy of any $v \in \gamma_{\mathsf{trm}}(\mathsf{v})$.

Proof. Straightforwardly by Lemma 3 and Theorem 2.

As mentioned in Section 2.3, we assume that each bound name in the protocol specification has a distinct canonical representative; this helps verifying the secrecy of messages as, fixed a protocol session, every message has a different abstraction in the causal graph. For instance, the causal graph of Example 3 guarantees the secrecy of m and, consequently, the protocol of Example 1 guarantees the secrecy of the authenticated message in every protocol session. The next theorem states that causal graphs are a safe abstract interpretation of authenticity.

Theorem 4 (Authenticity) Let P be a process and $\mathscr{C}$ a cycle-invariant causal graph s.t. $\mathscr{C} = graph(P)$.

- If $\mathscr{C}$ guarantees authenticity, then P guarantees weak authenticity.
- If $\mathscr{C}$ guarantees authenticity and P is nonce linear, then P guarantees strong authenticity.

Proof. Directly by Lemma 6 and Theorem 2. Notice that since P is nonce linear, then for every $s \in T(P)$, $end^i_n(I,J,M) \in s$ and $end^i_{n'}(I',J',M') \in s$ imply $n \neq n'$, thus getting the desired injectivity property.

This means that causal graphs, which express the causality among process events, are a sound (and decidable) model for proving weak authenticity while the freshness of authentication requests, namely the condition distinguishing weak from strong authenticity, may be directly verified on the syntax of ρ-spi processes.

D Variable Instantiation with Ciphertexts

In this section we relax the assumption that variables are only instantiated by names. The reason for this assumption is preventing the number of nodes in the causal graphs from diverging, which happens if variables are instantiated with a possibly infinite number of ciphertexts. An effective solution to this problem is to guarantee that the number of ciphertexts generated by trusted principals is finite and to abstract away from the ones

generated by the environment. We introduce the new syntactic category of ciphertext variables, ranged over by ξ, ϕ, ψ. These variables can only be instantiated by ciphertexts: as mentioned in the paper, we assume that names and ciphertexts are tagged so as to distiguish them. If ciphertext variables are not encrypted within the process, then it is easy to see that the number of ciphertexts generated by the process is finite. We do not find this over-approximation too inconvenient since nested encryptions are expensive and typically avoided and, in fact, in most protocols principals simply forward ciphertexts since the binding between different ciphertexts is given by some secret such as a nonce or a key (e.g., Kerberos [28]). Furthermore, this approximation does not rule out protocols where a principal generates a nested encryption that does not contain any ciphertext (e.g., Needham Schroeder symmetric-key [27]).

In order to abstract away from the ciphertexts generated by the environment, the special message $\mathscr{E}$ now abstracts over all terms possibly known to the environment, not only the initially known ones.

D.1 ρ-spi calculus and causal graphs

We start by extending function *bind* in order to allow for the instantiation of ρ-spi variables with either names or ciphertexts.

$$
\begin{array}{ll}
bind(a,a) = [] & \\
bind(x,a) = [a/x] & \\
bind(\xi,\{G\}_a) = [\{G'\}_a/\xi] & \\
bind((M,M'),(G,G')) = bind(M,G) \uplus bind(M',G') & \\
bind(\{M\}_{\overline{a}},\{G\}_a) = bind(M,G) & \\
bind(M,G) = \uparrow & \text{otherwise}
\end{array}
$$

Second, we do the same for function $bind_{\mathsf{abs}}$:

$$
\begin{array}{ll}
bind_{\mathsf{abs}}(\mathsf{v}_1,\mathsf{v}_2) = \mathsf{v}_2 & \text{if either } [\mathsf{v}_1] = [\mathsf{v}_2] \\
 & \text{or } ([\mathsf{v}_1] = \mathscr{E} \wedge [\mathsf{v}_2] \in \{\mathsf{I}, \mathsf{k}_\mathsf{I}^+, \mathsf{k}_{\mathsf{IE}}, \mathsf{k}_{\mathsf{EI}}, \mathsf{k}_\mathsf{E}^-, \mathscr{E}\}) \\
 & \text{or } ([\mathsf{v}_2] = \mathscr{E} \wedge [\mathsf{v}_1] \in \{\mathsf{I}, \mathsf{k}_\mathsf{I}^+, \mathsf{k}_{\mathsf{IE}}, \mathsf{k}_{\mathsf{EI}}, \mathsf{k}_\mathsf{E}^-, \mathscr{E}\}) \\
bind_{\mathsf{abs}}(\mathsf{x},\mathsf{a}) = \mathsf{a}_{(\mathsf{x})} & \\
bind_{\mathsf{abs}}(\xi,\{\mathsf{G}\}_\mathsf{v}) = \{\mathsf{G}\}_\mathsf{v}\sigma & \sigma \text{ maps each name a in } \{\mathsf{G}\}_\mathsf{v} \text{ to } \mathsf{a}_{(\xi_i)} \text{ for some fresh } \xi_i \\
bind_{\mathsf{abs}}((\mathsf{M},\mathsf{M}'),(\mathsf{G},\mathsf{G}')) = (\mathsf{G}_1,\mathsf{G}_2) & \text{if } bind_{\mathsf{abs}}(\mathsf{M},\mathsf{G}) = \mathsf{G}_1 \wedge bind_{\mathsf{abs}}(\mathsf{M}',\mathsf{G}') = \mathsf{G}_2 \\
bind_{\mathsf{abs}}(\{\mathsf{M}\}_\mathsf{v},\{\mathsf{G}\}_{\mathsf{v}'}) = \{\mathsf{G}_1\}_{\mathsf{v}_2} & \text{if } bind_{\mathsf{abs}}(\mathsf{M},\mathsf{G}) = \mathsf{G}_1 \wedge \\
 & \qquad (bind_{\mathsf{abs}}(\overline{\mathsf{v}},\mathsf{v}') \neq \uparrow \vee bind_{\mathsf{abs}}(\mathsf{v},\overline{\mathsf{v}'}) \neq \uparrow) \\
bind_{\mathsf{abs}}(\mathsf{M},\mathsf{G}) = \uparrow & \text{otherwise}
\end{array}
$$

Notice that each name a in the ciphertext replacing variable ξ is mapped to $a_{(\xi_i)}$, for some fresh variable ξ_i. This allows for tracking session messages within the received ciphertext. Function *subst* is extended accordingly so as to yield a substitution mapping each ciphertext variable to the corresponding ciphertext. For preventing the number of nodes from diverging, we require in function *graph* that ciphertext variables are only instantiated by ciphertexts whose encryption key is unknown to the environment. This way we prevent variables from being instantiated by the infinite number of ciphertexts generated by the environment, yet allowing for their instantiation by the finite number of ciphertexts generated by trusted processes.

$$\mathit{threads}(\mathsf{P}) \subseteq \mathcal{K}$$

$$\mathsf{p}.\mathsf{P} \in \mathcal{K} \ \wedge\ \mathsf{Q} \in \mathit{threads}(\mathsf{P}) \ \wedge\ \mathsf{p} \neq \mathsf{in}(\cdot)$$
$$\Rightarrow \mathsf{Q} \in \mathcal{K} \ \wedge\ \mathsf{p}.\mathsf{P} \rightarrow \mathsf{Q} \in \mathcal{E}$$

$$\mathsf{in}(\mathsf{M}).\mathsf{P} \in \mathcal{K} \ \wedge\ \mathit{output}(\mathcal{K}) \vdash \mathsf{G} \ \wedge\ \mathit{bind}_{\mathsf{abs}}(\mathsf{M},\mathsf{G}) = \mathsf{G}' \ \wedge\ \sigma = \mathit{subst}(\mathsf{G}') \ \wedge\ \mathsf{Q} \in \mathit{threads}(\mathsf{P}\sigma) \wedge$$
$$\mathsf{G}_1 \in \mathit{int}(\mathcal{K},\mathsf{G}') \ \wedge\ \{\mathsf{P}_1\} = \mathit{index}(\mathsf{G}_1) \ \wedge\ \{\mathsf{G}_2\}_{\mathsf{v}_2} \in \mathit{range}(\sigma) \Rightarrow \mathit{output}(\mathcal{K}) \nvdash \mathsf{v}_2$$
$$\Rightarrow \{\mathsf{Q}, \mathsf{com}(\mathsf{G}')\} \subseteq \mathcal{K} \ \wedge\ \{\mathsf{in}(\mathsf{M}).\mathsf{P} \rightarrow \mathsf{com}(\mathsf{G}'),\ \mathsf{P}_1 \xrightarrow{\mathsf{G}_1} \mathsf{com}(\mathsf{G}'),\ \mathsf{com}(\mathsf{G}') \rightarrow \mathsf{Q}\} \subseteq \mathcal{E}$$

Finally, we need to refine the concretization functions γ_{trm} and γ_{env}. Their definition is the same as in Table 3, apart from the concretization of $\mathscr{E}$:

$$\gamma_{\mathsf{trm}}(\mathscr{C},\mathscr{E}) = \mathit{ID} \cup \mathcal{K}^+ \cup \mathcal{M}_{\mathcal{I}} \cup \mathcal{K}_{\mathcal{I}} \cup \{G \mid G \in \gamma_{\mathsf{env}}(\mathscr{C},\mathsf{G}) \ \wedge\ \mathit{output}(\mathit{nodes}(\mathscr{C})) \vdash \mathsf{G} \ \wedge\ \mathsf{G} \neq \mathscr{E}\}$$
$$\gamma_{\mathsf{env}}(\mathscr{C},\mathscr{E}) = \mathit{ID} \cup \mathcal{K}^+ \cup \mathcal{M}_{\mathcal{I}} \cup \mathcal{K}_{\mathcal{I}} \cup \{G \mid G \in \gamma_{\mathsf{env}}(\mathscr{C},\mathsf{G}) \ \wedge\ \mathit{output}(\mathit{nodes}(\mathscr{C})) \vdash \mathsf{G} \ \wedge\ \mathsf{G} \neq \mathscr{E}\}$$

D.2 Proofs of soundness and safety

Lemma 1 is modified according to the definition of γ_{trm}.

Lemma 7 (Term Concretization). *For every term* G *and causal graph* $\mathscr{C}$, $\gamma_{\mathit{trm}}(\mathscr{C},\mathsf{G}) \subseteq \gamma_{\mathit{env}}(\mathscr{C},\mathsf{G})$.

Proof. The proof is similar to the one of Lemma 1.

In the following, we say that a term M is $\mathscr{E}$-free if and only if $\mathsf{v} \in \mathsf{payload}(\mathsf{M})$ implies $\lfloor \mathsf{v} \rfloor \neq \mathscr{E}$, where payload is defined as follows:

$$\mathsf{payload}(\mathsf{M}) \triangleq \begin{cases} \{\mathsf{u}\} & \text{if } \mathsf{M} = \mathsf{u} \\ \mathsf{payload}(\mathsf{M}_1) \cup \mathsf{payload}(\mathsf{M}_2) & \text{if } \mathsf{M} = (\mathsf{M}_1, \mathsf{M}_2) \\ \mathsf{payload}(\mathsf{M}') & \text{if } \mathsf{M} = \{\mathsf{M}'\}_u \end{cases}$$

Furthermore, we say that a term G' is a $\mathscr{E}$-unfolding of G in $\mathscr{C}$, written $\mathsf{G}' \in \mathsf{unfold}(\mathscr{C},\mathsf{G})$, if G' is obtained from G by replacing some occurrences of $\mathscr{E}$ in G with some message $\lceil \mathsf{G}'' \rceil$ such that $\mathit{output}(\mathit{nodes}(\mathscr{C})) \vdash \mathsf{G}''$. Lemma 8 is similar to Lemma 2: the only difference is that we require the abstract term M to be $\mathscr{E}$-free. Notice that input patterns in causal graph generated from a ρ-spi process are always $\mathscr{E}$-free.

Lemma 8 (Binding). *Let* $\mathscr{C}$ *be a causal graph,* G *and* M ρ*-spi terms,* G *and* M *abstract terms such that* $G \in \gamma_{\mathit{env}}(\mathscr{C},\mathsf{G})$, $M \in \gamma_{\mathit{trm}}(\mathsf{M})$, $\mathit{bind}(M,G) \neq \uparrow$ *and* M *is* $\mathscr{E}$*-free. Then* $\exists \mathsf{G}' \in \mathit{unfold}(\mathscr{C},\mathsf{G})$ *such that* $\mathit{bind}_{\mathsf{abs}}(\mathsf{M},\mathsf{G}') = \mathsf{G}''$, $\sigma = \mathit{subst}(\mathsf{G}'')$, *and* $\overline{G} \in \gamma_{\mathit{trm}}(\mathsf{M}\sigma)$.

Proof. By induction on the structure of M.

Base Case We have two cases, depending on whether M is a variable or a name:

- Let $M = x$ and $G = v$. By definition of function γ_{trm}, $\mathsf{M} = \mathsf{x}$, for some x. We have two cases, depending on whether G is a name or a ciphertext. If G is a name, then either $\mathsf{G} = \mathsf{v}$, for some v such that $n \in \gamma_{\mathsf{env}}(\mathscr{C},\mathsf{v})$, or $\mathsf{G} = \mathscr{E}$. The thesis derives directly from the definition of function $\mathit{bind}_{\mathsf{abs}}$. If G is a ciphertext, then the reasoning is similar.

– Let $M = a$ and $G = a$. We have two cases, depending on whether $\mathsf{G} = \mathsf{v}$, for some v such that $a \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{v})$, or $\mathsf{G} = \mathscr{E}$. In the former case, the proof is as in Lemma 2. Since $n \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G})$, if $\mathsf{G} = \mathscr{E}$, then there exists $\mathsf{v} \in \mathsf{unfold}(\mathscr{C}, \mathscr{E})$ such that $n \in \gamma_{\mathsf{trm}}(\mathscr{C}, \mathsf{v})$, as desired.

Inductive Step We show the proof for the case where M is a ciphertext in that the remaining cases are similar.

– Let $M = \{M'\}_{a_1}$ and $G = \{G'\}_{a_2}$, with $\overline{a_1} = a_2$. Since M is $\mathscr{E}$-free, by definition of γ_{trm} there exist $\mathsf{M}', \mathsf{v}_1, \sigma''$ such that $\mathsf{M} = \{\mathsf{M}'\}_{\mathsf{v}_1}$ and $M \in \gamma_{\mathsf{trm}}(\mathscr{C}, \mathsf{M})$. We have two cases, depending on whether G is a ciphertext or $\mathscr{E}$. Suppose that there exist $\mathsf{G}', \mathsf{v}_2$ such that $\mathsf{G} = \{\mathsf{G}'\}_{\mathsf{v}_2}$, $G' \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}')$ and $a_2 \in \gamma_{\mathsf{trm}}(\mathscr{C}, \mathsf{v}_2)$. (If $\mathsf{G} = \mathscr{E}$, then there exists $\mathsf{G}' \neq \mathscr{E}$ such that $G \in \gamma_{\mathsf{env}}(\mathscr{C}, \mathsf{G}')$ and thus G' is a ciphertext as well.) It is easy to see that if $\overline{a_1} = a_2$ then either $\mathit{bind}_{\mathsf{abs}}(\overline{\mathsf{v}_1}, \mathsf{v}_2) \neq \uparrow$ or $\mathit{bind}_{\mathsf{abs}}(\mathsf{v}_1, \overline{\mathsf{v}_2}) \neq \uparrow$. By induction hypothesis, there exists $\mathsf{G}'' \in \mathsf{unfold}(\mathscr{C}, \mathsf{G}')$ such that $\mathit{bind}_{\mathsf{abs}}(\mathsf{M}', \mathsf{G}'') = \mathsf{G}'''$, $\sigma' = \mathit{subst}(\mathsf{G}''')$ and $G \in \gamma_{\mathsf{trm}}(\mathscr{C}, \mathsf{M}'\sigma')$. The thesis follows directly from the definition of $\mathit{bind}_{\mathsf{abs}}$ and γ_{trm}.

Since processes do not encrypt ciphertexts received from the network, it is easy to prove that if $\mathit{output}(\mathit{nodes}(\mathscr{C})) \vdash \mathsf{G}$ then $\mathit{output}(\mathit{nodes}(\mathscr{C})) \vdash \mathsf{G}'$ for every $\mathsf{G}' \in \mathsf{unfold}(\mathscr{C}, \mathsf{G})$. The rest of the proof is similar to the one in Section C.