Simulation based security in the applied pi calculus

Stéphanie Delaune[1], Steve Kremer[1], and Olivier Pereira[2]

[1] LSV, ENS Cachan & CNRS & INRIA, France
[2] UCL Crypto group, Belgium

Abstract. We present a symbolic framework for refinement and composition of security protocols. The framework uses the notion of ideal functionalities. These are abstract systems which are secure by construction and which can be combined into larger systems. They can be separately refined in order to obtain concrete protocols implementing them. Our work builds on ideas from computational models such as the universally composable security and reactive simulatability frameworks. The underlying language we use is the applied pi calculus which is a general language for specifying security protocols. In our framework we can express the different standard flavours of simulation-based security which happen to all coincide. We illustrate our framework on an authentication functionality which can be realized using the Needham-Schroeder-Lowe protocol. For this we need to define an ideal functionality for asymmetric encryption and its realization. We also show a joint state result for this functionality which allows composition (even though the same key material is reused) using a tagging mechanism.

1 Introduction

Symbolic techniques showed to be a very useful approach for the modeling and analysis of security protocols: for twenty years, they improved our understanding of security protocols, allowed discovering flaws [18], and provided support for protocol design [11]. These techniques also resulted in the creation of powerful automated analysis tools [5, 10, 3], and impacted on several protocol standards used every day, e.g., [9].

Until now, symbolic techniques mostly concentrated on specifying and proving confidentiality and correspondence properties, i.e., showing which symbols are kept secret, and on which session parameters participants agree when a protocol session completes. However, such specifications do not provide any information about the behavior of protocols when they are used in composition with other protocols, and surprising behaviors are well know to happen in such contexts [8]. Moreover, protocols are often expected to provide more sophisticated security guarantees: we can think of privacy-type properties for voting protocols, or input independence for auction protocols.

In this paper, we present a symbolic framework for refinement and composition of security protocols, in which protocols are defined in terms of the behavior of ideal functionalities, following the general outline of simulation-based security [6, 13, 4]. A lower-level protocol is said to securely emulate a higher-level protocol, or ideal functionality, if any behavior that can be observed from the interaction of an adversary with the lower-level protocol can also be observed from the interaction of another adversary (called the simulator) with the higher-level protocol. As a result, ideal functionalities

can be successively refined into more concrete protocols, but also composed to build more complex protocols. Functionalities have been proposed for a wide range of protocol tasks, including general secure multi-party computation [6]. In the spi-calculus [2], Abadi and Gordon also present the idea of a protocol being equivalent to an idealized version. This is however more restrictive as they do not have the notion of a simulator.

Simulation-based security frameworks can typically be decomposed into two "layers": (i) a foundational layer that provides a general model for concurrent computation, and (ii) a security layer that provides general security definitions, most importantly the notion of secure protocol emulation to be used. While the security layer is essentially common to all frameworks [4, 6, 7, 15, 19], including this paper, the foundational layer varies widely. Those variations typically lie in the concurrency model (from the most common token-passing mechanism to the use of scheduler with various powers) and in the definition of computational bounds. These differences typically result in incomparable security notions.

Defining simulation-based security while choosing the applied pi calculus [1] as the foundational layer brings the main benefits of this approach into the symbolic world:

- it provides a powerful machinery that can be used to specify a wide range of sophisticated protocol tasks in terms of the behavior of functionalities, and
- general composition theorems guarantee that protocols keep behaving as expected when executed in arbitrary contexts.

While we tried to stick to the common definitions from the security layer of simulation-based security frameworks, the use of the applied pi calculus as foundational layer raised interesting challenges.

First, at the most foundational level, one has to adopt a notion of indistinguishability of processes. While the symmetric notions of computational indistinguishability and observational equivalence are most commonly used in the cryptographic and symbolic worlds respectively, the symmetry of such relations appeared to be too restrictive for our purpose. For instance, requiring a symmetric equivalence relation makes visible the addition of an adversary that simply acts as a relay. Such undesired behaviors motivate the introduction of new notions of observational preorder and labelled simulation relations in the applied pi calculus.

Next, our attempts at translating ideal functionalities from the computational world into the symbolic world showed to be a non immediate task. For instance, traditional ideal functionalities for asymmetric encryption produce ciphertexts by encrypting random strings unrelated to the original messages. In our setting we use a technique of double-encrypting messages such that a plaintext corresponding to a given ciphertext can only be retrieved through the decryption services offered by the functionalities.

Eventually, we investigate the statement of general composition theorems, and of a specific joint state composition theorem for our asymmetric encryption functionality, as this functionality is typically expected to be used in several protocol sessions. While these theorems appear to be the natural counterpart of their computational versions [4, 6, 7, 17], the joint state composition theorem brings message tagging constraints that are consistent with those obtained by using a completely different symbolic approach (e.g. [14]).

2 The applied pi calculus

The applied pi calculus [1] is a language for describing concurrent processes and their interactions.

2.1 Syntax and informal semantics

To describe processes, one starts with a set of *names* (which are used to name communication channels or other atomic data), a set of *variables*, and a *signature* Σ which consists of the *function symbols* which will be used to define *terms*. In the case of security protocols, typical function symbols will include enc for encryption, which takes plaintext and a key and returns the corresponding ciphertext, and dec for decryption, taking ciphertext and a key and returning the plaintext. Terms are defined as names, variables, and function symbols applied to other terms. Terms and function symbols are sorted, and of course function symbol application must respect sorts and arities. By the means of an equational theory E we describe the equations which hold on terms built from the signature. We denote $=_{\mathsf{E}}$ the equivalence relation induced by E. Two terms are related by $=_{\mathsf{E}}$ only if that fact can be derived from the equations in E. When the set of variables occurring in a term T is empty, we say that T is *ground*.
Example 1. Let $\mathsf{E}_{\mathsf{enc}}$ be the theory made up of the equations $\mathsf{dec}(\mathsf{enc}(x,k),k) = x$ and $\mathsf{test}(\mathsf{enc}(x,y),y) = \mathsf{ok}$. We have that $\mathsf{test}(\mathsf{dec}(\mathsf{enc}(\mathsf{enc}(n,k_1),k_2),k_2),k_1) =_{\mathsf{E}} \mathsf{ok}$.

In the applied pi calculus, one has *plain processes* and *extended processes*. Plain processes are built up in a similar way to processes in the pi calculus, except that messages can contain terms (rather than just names). Below, M and N are terms, n is a name, x a variable and u is a metavariable, standing either for a name or a variable. Extended processes add *active substitutions* and restriction on variables.

$P, Q, R :=$	plain processes	$A, B, C :=$	extended processes
	0		P
	$P \mid Q$		$A \mid B$
	$\nu n.P$		$\nu n.A$
	if $M = N$ then P else Q		$\nu x.A$
	$\mathsf{in}(u, x).P$		$\{^M/_x\}$
	$\mathsf{out}(u, N).P$		

$\{^M/_x\}$ is the substitution that replaces the variable x with the term M. Active substitutions generalise "let". The process $\nu x.(\{^M/_x\} \mid P)$ corresponds exactly to the process "let $x = M$ in P". As usual, names and variables have scopes, which are delimited by restrictions and by inputs. We write $fv(A)$, $bv(A)$, $fn(A)$ and $bn(A)$ for the sets of free and bound variables and free and bound names of A, respectively. We also assume that, in an extended process, there is at most one substitution for each variable, and there is exactly one when the variable is restricted. We say that an extended process is *closed* if all its variables are either bound or defined by an active substitution.

Active substitutions are useful because they allow us to map an extended process A to its *frame* $\phi(A)$ by replacing every plain process in A with 0. A frame is an extended process built up from 0 and active substitutions by parallel composition and restriction. The frame $\phi(A)$ can be viewed as an approximation of A that accounts for the static knowledge A exposes to its environment, but not A's dynamic behaviour. The domain of

a frame φ, denoted by $\mathrm{dom}(\varphi)$, is the set of variables for which φ defines a substitution (those variables x for which φ contains a substitution $\{^M/_x\}$ not under a restriction on x). An *evaluation context* $C[\_]$ is an extended process with a hole instead of an extended process.

Example 2. Consider the following process P:

$$\nu k.\,(\ \mathsf{in}(io_1, x).\mathsf{out}(net, \mathsf{enc}(x, k))$$
$$\mid \mathsf{in}(net, y).\ \text{if } \mathsf{test}(y, k) = \mathsf{ok} \text{ then } \mathsf{out}(io_2, \mathsf{dec}(y, k)) \text{ else } 0).$$

The first component receives a message x on the channel io_1 and sends its encryption with the key k on the channel net. The second is waiting for an input y on net, uses the secret key k to decrypt it. If the decryption succeeds, then it forwards the resulting plaintext on io_2.

2.2 Semantics

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence* (briefly described in Section 2.1) and *internal reduction*, noted $\rightarrow$. Structural equivalence, noted $\equiv$, is the smallest equivalence relation on extended processes that is closed under α-conversion on names and variables, application of evaluation contexts, and some other standard rules such as associativity and commutativity of the parallel operator and commutativity of the bindings. In addition the following three rules are related to active substitutions and equational theories:

$$\nu x.\{^M/_x\} \equiv 0,\ \{^M/_x\} \mid A \equiv \{^M/_x\} \mid A\{^M/_x\},\ \text{and } \{^M/_x\} \equiv \{^N/_x\}\ \text{ if } M =_{\mathsf{E}} N$$

Internal reduction $\rightarrow$ is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that

$$\begin{array}{lll}
\textsc{Comm} & \mathsf{out}(a, M).P \mid \mathsf{in}(a, x).Q \rightarrow P \mid Q\{^M/_x\} & \\
\textsc{Then} & \text{if } M = N \text{ then } P \text{ else } Q \rightarrow P & \text{where } M =_{\mathsf{E}} N \\
\textsc{Else} & \text{if } M = N \text{ then } P \text{ else } Q \rightarrow Q & \\
 & \text{for any ground terms } M \text{ and } N \text{ such that } M \neq_{\mathsf{E}} N &
\end{array}$$

The operational semantics is extended by a *labelled* operational semantics enabling us to reason about processes that interact with their environment. Labelled operational semantics defines the relation $\xrightarrow{\alpha}$ where α is either an input $\mathsf{in}(a, M)$ (a is a channel name and M is a term that can contain names and variables), or $\nu x.\mathsf{out}(a, x)$ (x is variable of base type), or $\mathsf{out}(a, c)$ or $\nu c.\mathsf{out}(a, c)$ (c is a channel name). We adopt the following rules in addition to the internal reduction rules:

$$\textsc{In}\quad \mathsf{in}(a, x).P \xrightarrow{in(a,M)} P\{^M/_x\}$$

$$\textsc{Out-Ch}\quad \mathsf{out}(a, c).P \xrightarrow{out(a,c)} P$$

$$\textsc{Open-Ch}\quad \frac{A \xrightarrow{out(a,c)} A' \qquad c \neq a}{\nu c.A \xrightarrow{\nu c.out(a,c)} A'}$$

$$\textsc{Out-T}\quad \mathsf{out}(a, M).P \xrightarrow{\nu x.out(a,x)} P \mid \{^M/_x\} \qquad x \notin fv(P) \cup fv(M)$$

$$\textsc{Scope}\quad \frac{A \xrightarrow{\alpha} A' \quad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'} \qquad bn(\alpha) \cap fn(B) = \emptyset$$

$$\textsc{Par}\quad \frac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\textsc{Struct}\quad \frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$$

Our rules differ slightly from those described in [1], although we prove in [12] that labelled bisimulation in our system coincides with labelled bisimulation in [1].

Example 3. Consider the process P defined in Example 2. We have that:

$$\begin{aligned} P &\xrightarrow{in(io_1,s)} \nu k.(\mathsf{out}(net, \mathsf{enc}(s,k)) \\ &\qquad\qquad \mid \mathsf{in}(net, y).\ \mathsf{if}\ \mathsf{test}(y,k) = \mathsf{ok}\ \mathsf{then}\ \mathsf{out}(io_2, \mathsf{dec}(y,k))\ \mathsf{else}\ 0) \\ &\longrightarrow \quad \nu k.\ \mathsf{if}\ \mathsf{test}(\mathsf{enc}(s,k),k) = \mathsf{ok}\ \mathsf{then}\ \mathsf{out}(io_2, \mathsf{dec}(\mathsf{enc}(s,k),k))\ \mathsf{else}\ 0) \\ &\longrightarrow \quad \nu k.\mathsf{out}(io_2, s) \\ &\xrightarrow{\nu x.out(io_2,x)} \quad \nu k.\{^s/_x\} \end{aligned}$$

Let A be the resulting process. We have that $\phi(A) \equiv \nu k.\{^s/_x\}$.

2.3 Equivalences

In [1], it is shown that observational equivalence coincides with labelled bisimilarity. This relation is like the usual definition of bisimilarity, except that at each step one additionally requires that the processes are statically equivalent.

Definition 1 (static equivalence ($\approx_s$)). *Two terms M and N are* equal in the frame ϕ*, written $(M =_{\mathsf{E}} N)\phi$, if, and only if there exists $\tilde{n}$ and a substitution σ such that $\phi \equiv \nu\tilde{n}.\sigma$, $M\sigma =_{\mathsf{E}} N\sigma$, and $\tilde{n} \cap (fn(M) \cup fn(N)) = \emptyset$. Two frames ϕ_1 and ϕ_2 are* statically equivalent*, $\phi_1 \approx_s \phi_2$, when $\mathrm{dom}(\phi_1) = \mathrm{dom}(\phi_2)$, and for all terms M, N we have that $(M =_{\mathsf{E}} N)\phi_1$ if and only if $(M =_{\mathsf{E}} N)\phi_2$.*

Example 4. Let $\varphi_0 = \nu s.\{^{\mathsf{enc}(s,k)}/_x\}$ and $\varphi_1 = \nu r.\{^r/_x\}$ where k, s and r are names and E be the theory given in Example 1. We have $(\mathsf{test}(x,k) =_{\mathsf{E}} \mathsf{ok})\varphi_0$ but not $(\mathsf{test}(x,k) =_{\mathsf{E}} \mathsf{ok})\varphi_1$, thus $\varphi_0 \not\approx_s \varphi_1$. However, we have $\nu k.\varphi_0 \approx_s \varphi_1$.

We are now defining observational equivalence and observational preorder. For this we introduce the notion of a *barb*. Given an extended process A and a channel name a, we write $A \Downarrow a$ when $A \rightarrow^* C[\mathsf{out}(a, M).P]$ for some term M, plain process P, and evaluation context $C[\_]$ that does not bind a.

Definition 2 (observational preorder, equivalence). Observational preorder *($\preceq$) (resp.* equivalence *($\approx$)) is the largest (resp. largest symmetric) relation on extended processes having same domain such that $A\ \mathcal{R}\ B$ implies*

1. *if $A \Downarrow a$ then $B \Downarrow a$;*
2. *if $A \rightarrow^* A'$, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B';*
3. *$C[A]\ \mathcal{R}\ C[B]$ for all closing evaluation contexts $C[\_]$.*

Definition 3 (labelled (bi)similarity). *A relation $\mathcal{R}$ on closed extended processes is a simulation relation if $A\ \mathcal{R}\ B$ implies*

1. *$\phi(A) \approx_s \phi(B)$,*
2. *if $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B',*
3. *if $A \xrightarrow{\alpha} A'$ and $fv(\alpha) \subseteq \mathrm{dom}(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ and $A'\ \mathcal{R}\ B'$ for some B'.*

If $\mathcal{R}$ and $\mathcal{R}^{-1}$ are both simulation relations we say that $\mathcal{R}$ is a bisimulation relation. Labelled similarity *($\preceq_\ell$), resp.* labelled bisimilarity *($\approx_\ell$), is the largest simulation, resp. bisimulation relation.*

Observational preorder and similarity were not introduced in [1]. However, these definitions seem natural for our purposes. Obviously we have that $\approx \subset \preceq$ and $\approx_\ell \subset \preceq_\ell$. We now show that labelled bisimilarity is a precongruence.

Proposition 1. *Let A and B be two extended processes such that $A \preceq_\ell B$. We have that $C[A] \preceq_\ell C[B]$ for all closing evaluation context $C[\_]$.*

From this proposition it follows that $\preceq_\ell \subseteq \preceq$. Hence, we can use labelled similarity as a convenient proof technique for observational preorder. We actually expect the two relations to coincide but did not prove it as we did not need it. Moreover, the relation $\preceq_\ell$ is stable under application of replication and replication distributes over parallel.

Lemma 1. *Let P and Q be two closed plain processes. We have that: (i) if $P \preceq_\ell Q$ then $!P \preceq_\ell !Q$; (ii) $!(P \mid Q) \preceq_\ell !P \mid !Q$ and $!P \mid !Q \preceq_\ell !(P \mid Q)$.*

3 Simulation based security

3.1 Basic definitions

The simulation-based security approach classically distinguishes between *input-output channels*, which yield the internal interface of a protocol or *functionality* to its environment and *network channels*, which allow the adversary to interact with the functionality. We suppose that all channels are of one of these two sorts: IO or NET. Moreover the sort system ensures that names of sort NET can never be conveyed as data on a channel, i.e. these channels can never be transmitted. We write $\mathsf{fnet}(P)$ for $fn(P) \cap \mathsf{NET}$.

Definition 4 (functionality, adversary). *A* functionality *$\mathcal{F}$ is a closed plain process. An* adversary *for $\mathcal{F}$ is an evaluation context $\mathcal{A}[\_]$ of the form:*
$\nu\widetilde{net}_1.(A_1 \mid \nu\widetilde{net}_2.(A_2 \mid \ldots \mid \nu\widetilde{net}_k.(A_k \mid \_)\ldots))$ *with* $\mathsf{fnet}(\mathcal{F}) \subseteq \bigcup_{1\leq i\leq k} \widetilde{net}_i \subseteq \mathsf{NET}$ *where each A_i ($1 \leq i \leq k$) is a closed plain process, and $fn(\mathcal{A}[\_]) \cap \mathsf{IO} = \emptyset$.*

We may note that if $\mathcal{A}[\_]$ is an adversary for $\mathcal{F}$ then $\mathsf{fnet}(\mathcal{A}[\_]) = \mathsf{fnet}(\mathcal{A}[\mathcal{F}])$.

Lemma 2. *Let $\mathcal{F}$ be a functionality and $\mathcal{A}_1[\_]$ be an adversary for $\mathcal{F}$. Then $\mathcal{A}_1[\mathcal{F}]$ is a functionality. Besides, if $\mathcal{A}_2[\_]$ is an adversary for $\mathcal{A}_1[\mathcal{F}]$, then $A_2[\mathcal{A}_1[\_]]$ is an adversary for $\mathcal{F}$.*

While adversaries can control the communications of functionalities on NET channels, *IO contexts* model the environment of those functionalities, providing them with their inputs and collecting their outputs.

Definition 5 (IO context). *An* IO context *is an evaluation context $C_{io}[\_]$ of the form $\nu\widetilde{io}_1.(C_1 \mid \nu\widetilde{io}_2.(C_2 \mid \ldots \mid \nu\widetilde{io}_k.(C_k \mid \_)\ldots))$ with $\bigcup_{1\leq i\leq k} \widetilde{io}_i \subseteq \mathsf{IO}$ where each C_i ($1 \leq i \leq k$) is a closed plain process.*

Note that if $\mathcal{F}$ is a functionality and $C[\_]$ is an IO context, then $C_{io}[\mathcal{F}]$ is a functionality.

3.2 Strong simulatability

The notion of strong simulatability [16], which is probably the simplest secure emulation notion used in simulation-based security, can be formulated in our setting.

Definition 6 (strong simulatability). *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two functionalities. $\mathcal{F}_1$ emulates $\mathcal{F}_2$ in the sense of* strong simulatability, *written $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$, if there exists an adversary $\mathcal{S}$ for $\mathcal{F}_2$ (the simulator) such that $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{S}[\mathcal{F}_2])$ and $\mathcal{F}_1 \preceq S[\mathcal{F}_2]$.*

The definition ensures that any behavior of $\mathcal{F}_1$ can be matched by $\mathcal{F}_2$ executed in the presence of a specific adversary $\mathcal{S}$. Hence, there are no more attacks on $\mathcal{F}_1$ than attacks on $\mathcal{F}_2$. Moreover, the presence of $\mathcal{S}$ allows abstract definitions of higher-level functionalities, which are independent of a specific realization.

Example 5. Let $\mathcal{F}_{\mathsf{cc}} = \mathsf{in}(io_1, s).\mathsf{out}(net_{\mathsf{cc}}, \mathsf{ok}).\mathsf{in}(net_{\mathsf{cc}}, x).$ if $x = \mathsf{ok}$ then $\mathsf{out}(io_2, s)$. The functionality models a confidential channel and takes a potentially secret value s as input on channel io_1. The adversary is notified via channel net_{cc} that this value is to be transmitted. If the adversary agrees the value is output on channel io_2. This functionality can be realized by the process P introduced in Example 2.
Let $\mathcal{S} = \nu net_{\mathsf{cc}}.\mathsf{in}(net_{\mathsf{cc}}, x).\nu r.\mathsf{out}(net, r).\mathsf{in}(net, x).$ if $x = r$ then $\mathsf{out}(net_{\mathsf{cc}}, \mathsf{ok}) \mid \_)$.
We indeed have that $P \preceq_\ell \mathcal{S}[\mathcal{F}_{\mathsf{cc}}]$ and $\mathsf{fnet}(P) = \mathsf{fnet}(\mathcal{S}[\mathcal{F}_{\mathsf{cc}}])$.

In order to examine the properties of strong simulatability in our specific setting, we now define a particular adversary which is called a *dummy adversary*.

Definition 7 (dummy adversary). *Let $\mathcal{F}$ be a functionality. The* dummy adversary *for $\mathcal{F}$ is the adversary $D[\_] = \nu \widetilde{sim}.(D_1 \mid \nu \widetilde{net}.(D_2 \mid \_))$ where:*

- $\widetilde{net} = \mathsf{fnet}(\mathcal{F}) = \{net_1, \ldots, net_n\}$;
- $\widetilde{sim} = \{sim_1^i, \ldots, sim_n^i, sim_1^o, \ldots, sim_n^o\} \subseteq \mathsf{NET}$;
- $D_1 = !\mathsf{in}(net_1, x).\mathsf{out}(sim_1^i, x) \mid \ldots \mid !\mathsf{in}(net_n, x).\mathsf{out}(sim_n^i, x) \mid$
  $!\mathsf{in}(sim_1^o, x).\mathsf{out}(net_1, x) \mid \ldots \mid !\mathsf{in}(sim_n^o, x).\mathsf{out}(net_n, x)$;
- $D_2 = !\mathsf{in}(sim_1^i, x).\mathsf{out}(net_1, x) \mid \ldots \mid !\mathsf{in}(sim_n^i, x).\mathsf{out}(net_n, x) \mid$
  $!\mathsf{in}(net_1, x).\mathsf{out}(sim_1^o, x) \mid \ldots \mid !\mathsf{in}(net_n, x).\mathsf{out}(sim_n^o, x)$;

By construction we have that $\mathsf{fnet}(D[\mathcal{F}]) = \mathsf{fnet}(\mathcal{F})$.

Lemma 3. *Let $\mathcal{F}$ be a functionality and $D[\_]$ the dummy adversary for $\mathcal{F}$: $\mathcal{F} \preceq D[\mathcal{F}]$.*

However, we do not have that $\mathcal{F} \approx D[\mathcal{F}]$, since $D[\mathcal{F}]$ has more non-determinism than $\mathcal{F}$. A direct consequence of this lemma is that $\mathcal{F}_1 \preceq \mathcal{F}_2$ and $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{F}_2)$ implies that $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$: it is sufficient to observe that $\mathcal{F}_2 \preceq D[\mathcal{F}_2]$ and hence by transitivity $\mathcal{F}_1 \preceq D[\mathcal{F}_2]$. We use these observations to show that $\leq^{\mathsf{SS}}$ is a preorder.

Lemma 4. *The relation $\leq^{\mathsf{SS}}$ is a preorder, that is the following hold: (i)* reflexivity: *$\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_1$; (ii)* transitivity: *$\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$ and $\mathcal{F}_2 \leq^{\mathsf{SS}} \mathcal{F}_3 \Rightarrow \mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_3$.*

Moreover, we show that $\leq^{\mathsf{SS}}$ is closed under application of IO contexts and parallel composition of functionalities.

Proposition 2. *Let $\mathcal{F}_1$, $\mathcal{F}_2$ be functionalities and C_{io} be an IO context.*

$$\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \implies C_{io}[\mathcal{F}_1] \leq^{\mathsf{SS}} C_{io}[\mathcal{F}_2].$$

Now, we prove the following composition results. Actually, closure under parallel composition of functionalities is a direct consequence of the previous proposition by noticing that when $\mathcal{F}$ is a functionality then $\_ \mid \mathcal{F}$ is an IO-context. The proof of the closure under replication is more involved and given in Appendix.

Proposition 3. *Let $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ be three functionalities. We have that:*
(i) $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow \mathcal{F}_1 \mid \mathcal{F}_3 \leq^{\mathsf{SS}} \mathcal{F}_2 \mid \mathcal{F}_3$; and (ii) $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow !\mathcal{F}_1 \leq^{\mathsf{SS}} !\mathcal{F}_2$.

3.3 Other notions of simulation based security

Several other notions of simulation based security appear in the literature. We present these notions, and show that they all coincide in our setting. This coincidence is typically regarded as highly desirable [16, 15], while it does not hold in most simulation-based security frameworks [6, 4].

Definition 8. *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two functionalities and $\mathcal{A}$ be any adversary for $\mathcal{F}_1$.*

- *$\mathcal{F}_1$ emulates $\mathcal{F}_2$ in the sense of* black box simulatability, *written $\mathcal{F}_1 \leq^{\mathsf{BB}} \mathcal{F}_2$, iff $\exists \mathcal{S}. \forall \mathcal{A}. \mathcal{A}[\mathcal{F}_1] \preceq \mathcal{A}[\mathcal{S}[\mathcal{F}_2]]$ where $\mathcal{S}$ is an adversary for $\mathcal{F}_2$ with $\mathsf{fnet}(S[\mathcal{F}_2]) = \mathsf{fnet}(\mathcal{F}_1)$.*
- *$\mathcal{F}_1$ emulates $\mathcal{F}_2$ in the sense of* universally composable simulatability, *written $\mathcal{F}_1 \leq^{\mathsf{UC}} \mathcal{F}_2$, iff $\forall \mathcal{A}. \exists \mathcal{S}. \mathcal{A}[\mathcal{F}_1] \preceq \mathcal{S}[\mathcal{F}_2]$ where $\mathcal{S}$ is an adversary for $\mathcal{F}_2$ such that $\mathsf{fnet}(\mathcal{A}[\mathcal{F}_1]) = \mathsf{fnet}(\mathcal{S}[\mathcal{F}_2])$.*
- *$\mathcal{F}_1$ emulates $\mathcal{F}_2$ in the sense of* universally composable simulatability with dummy adversary, *written $\mathcal{F}_1 \leq^{\mathsf{UCDA}} \mathcal{F}_2$, iff $\exists \mathcal{S}. D[\mathcal{F}_1] \preceq \mathcal{S}[\mathcal{F}_2]$ where D is the dummy adversary for $\mathcal{F}_1$ and $\mathcal{S}$ is an adversary for $\mathcal{F}_2$ such that $\mathsf{fnet}(\mathcal{S}[\mathcal{F}_2]) = \mathsf{fnet}(D[\mathcal{F}_1])$.*

Theorem 1. *We have that $\leq^{\mathsf{SS}} = \leq^{\mathsf{BB}} = \leq^{\mathsf{UC}} = \leq^{\mathsf{UCDA}}$.*

The above security notions can also be defined replacing observational preorder by observational equivalence. We denote the corresponding relations by $\leq^{\mathsf{SS}}_{\approx}, \leq^{\mathsf{BB}}_{\approx}, \leq^{\mathsf{UC}}_{\approx}$ and $\leq^{\mathsf{UCDA}}_{\approx}$. Surprisingly, the use of observational equivalence turns out to be too strong, ruling out natural secure emulation cases: for instance, the $\leq^{\mathsf{SS}}_{\approx}$ relation is not reflexive.

4 Applications

We illustrate our framework by showing the secure emulation of a mutual authentication functionality by the Needham-Shroeder-Lowe (NSL) protocol [18].As the NSL protocol uses public key encryption we first introduce in Section 4.1 functionalities for asymmetric encryption: the ideal functionality and its realization together with a joint state composition result. Then, in Section 4.2, we present the mutual authentication functionality and its realization through the NSL protocol. Finally we use the joint state composition result in Section 4.3 to obtain a result for an unbounded number of concurrent sessions.

We use the notation $\mathsf{in}(u, =M)$ to test whether the input on u is equal (modulo E) to the term M (if not, the process blocks). We sometimes use tuples of terms, denoted by $\langle M_1, \ldots, M_n \rangle$, while keeping the equational theory for these tuples implicit. Lastly, we omit "else Q" in a conditional when $Q = 0$.

4.1 Asymmetric encryption with joint state

In this section we introduce a functionality for asymmetric encryption together with a *joint state composition result* which is crucial for composition of protocols that share key material. Even though encryption in a Dolev-Yao model is already idealized we will see that we nevertheless need to introduce an *ideal functionality* for encryption in order to obtain the joint state composition result. Throughout this section we rely on the following equational theory:

$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y), z), y) = x \qquad \mathsf{testdec}(\mathsf{aenc}(x, \mathsf{pk}(y), z), y) = \mathsf{ok}.$$

The first equation models randomized asymmetric encryption whereas the second one allows testing whether decryption with a given key succeeds or not.

Real encryption. The real encryption functionality is described in Figure 1:

- *Initialisation*: the functionality receives a channel name io^1_{pke}, which will be used for all sensitive information exchanges, i.e. when this functionality is used the channel io^1_{pke} should be restricted. A fresh private key sk is generated and the corresponding public key, i.e. $\mathsf{pk}(sk)$, is sent on io^1_{pke}. Then the process is ready to receive encryption or decryption requests. Note that encryption requests can be sent on the sensitive channel io^1_{pke} or on the public channel io^2_{pke} which is the channel the environment will typically use. Decryption requests are only available through the sensitive channel io^1_{pke} and thus will not be used by the attacker.
- *Encryption* $\mathcal{P}_{\mathsf{enc}}$: each time this process receives a request on the channel io^1_{pke} (resp. io^2_{pke}), it computes the corresponding ciphertext (probabilistic encryption) and outputs the ciphertext on the channel io^1_{pke} (resp. io^2_{pke}).
- *Decryption* $\mathcal{P}_{\mathsf{dec}}$: each time this process receives a request on the channel io^1_{pke}, it tries to decrypt the ciphertext and checks whether the tag is TAG_0. If so, it outputs the plaintext on the channel io^1_{pke}. Otherwise, it does nothing.

$$\begin{aligned}
\mathcal{P}_{\mathsf{pke}} &:= \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\nu sk.\mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{KEY}, \mathsf{pk}(sk)\rangle).\\
&\quad (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{P}_{\mathsf{enc}} \mid \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{P}_{\mathsf{enc}} \mid !\mathcal{P}_{\mathsf{dec}})\\
\mathcal{P}_{\mathsf{enc}} &:= \mathsf{in}(io^i_{\mathsf{pke}}, \langle = \text{ENC}, m\rangle).\\
&\quad \nu r_2.\ \text{let } menc = \mathsf{aenc}(\langle \text{TAG}_0, m\rangle, \mathsf{pk}(sk), r_2) \text{ in } \mathsf{out}(io^i_{\mathsf{pke}}, \langle \text{CIPHER}, menc\rangle)\\
\mathcal{P}_{\mathsf{dec}} &:= \mathsf{in}(io^1_{\mathsf{pke}}, \langle = \text{DEC}, m\rangle).\\
&\quad \text{let } \langle = \text{TAG}_0, m_1\rangle = \mathsf{adec}(m, sk) \text{ in } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, m\rangle)
\end{aligned}$$

Fig. 1: Real encryption functionality

Ideal functionality. We now propose, in Figure 2, an idealized version $\mathcal{F}_{\mathsf{pke}}$ of the real encryption functionality, which guarantees that the confidentiality of messages is preserved independently of any cryptanalytic effort that could be performed on ciphertexts from the knowledge of public keys. In various cryptographic settings [4, 6, 17], this is achieved by computing ciphertexts as the encryption of random messages instead of the actual plaintext. To be able to perform decryption, a table for plaintext/ciphertext associations is maintained. The burden of this association table is avoided in our symbolic specification by using two layers of encryption: messages are first encrypted using a secure key $\mathsf{pk}(ssk)$, then tagged and encrypted with the public key $\mathsf{pk}(sk)$ that is published during the initialization step. We stress that neither $\mathsf{pk}(ssk)$ nor ssk are ever transmitted by $\mathcal{F}_{\mathsf{pke}}$, guaranteeing that it is impossible to decrypt such a ciphertext outside the functionality, even if the key sk is adversarially chosen, which will be a crucial feature for our joint state composition theorem.

The ideal functionality behaves as follows:

- *Initialisation*: the attacker chooses the secret key sk and the tag that will be added in each encryption. Then a secure key ssk is generated and now the process is ready to receive encryption or decryption requests.
- *Encryption* $\mathcal{F}_{\mathsf{enc}}$: each time the process receives a request on the channel io^1_{pke} (resp. io^2_{pke}), it computes the corresponding ciphertext and outputs the ciphertext on the channel io^1_{pke} (resp. io^2_{pke}). As explained above, the plaintext m is first encrypted using $\mathsf{pk}(ssk)$ before being tagged and encrypted with $\mathsf{pk}(sk)$.
- *Decryption* $\mathcal{F}_{\mathsf{dec}}$: each time the process receives a request on the channel io^1_{pke}, it tries to decrypt the ciphertext and checks if the tag is the tag provided during the initialization. Then, it checks if the resulting plaintext is encrypted under $\mathsf{pk}(ssk)$. If so, this means that this ciphertext has been produced by the encryption functionality and thus has to be decrypted twice. Otherwise, the ciphertext has been produced by the attacker and the plaintext is sent on the channel io^1_{pke}.

$$\mathcal{F}_{\mathsf{pke}} := \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{out}(net, \text{INIT}).\mathsf{in}(net, \langle = \text{ALGO}, sk, tag\rangle).\mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{KEY}, \mathsf{pk}(sk)\rangle).$$
$$\nu ssk.\ (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{F}_{\mathsf{enc}} \mid \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{F}_{\mathsf{enc}} \mid !\mathcal{F}_{\mathsf{dec}})$$

$$\mathcal{F}_{\mathsf{enc}} := \mathsf{in}(io^i_{\mathsf{pke}}, \langle = \text{ENC}, m\rangle).\nu r_1.\nu r_2.$$
$$\text{let } alea = \mathsf{aenc}(m, \mathsf{pk}(ssk), r_1) \text{ in let } menc = \mathsf{aenc}(\langle tag, alea\rangle, \mathsf{pk}(sk), r_2) \text{ in}$$
$$\mathsf{out}(io^i_{\mathsf{pke}}, \langle \text{CIPHER}, menc\rangle)$$

$$\mathcal{F}_{\mathsf{dec}} := \mathsf{in}(io^1_{\mathsf{pke}}, \langle = \text{DEC}, m\rangle).\ \text{let } \langle = tag, m_1\rangle = \mathsf{adec}(m, sk) \text{ in}$$
$$\text{if } \mathsf{testdec}(m_1, ssk) = \mathsf{ok} \text{ then } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, \mathsf{adec}(m_1, ssk)\rangle)$$
$$\text{else } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, m_1\rangle)$$

Fig. 2: Ideal encryption functionality

Realization. We indeed have that the real encryption functionality realizes the ideal one, i.e., $\mathcal{P}_{\mathsf{pke}} \leq^{\mathsf{SS}} \mathcal{F}_{\mathsf{pke}}$. This is witnessed by the adversary:

$$\mathcal{A}_{\mathsf{pke}} = \nu net.(\mathsf{in}(net, = \text{INIT}).\ \nu sk.\ \mathsf{out}(net, (\text{ALGO}, sk, \text{TAG}_0)) \mid \_).$$

Composition with joint state. While $\leq^{\mathsf{SS}}$ is stable under replication this is not always sufficient to obtain composition guarantees. Indeed replication of a process also replicates all key generation operations. In order to obtain self-composition and inter-protocol composition with common key material we need a *joint state functionality* $\mathcal{P}_{\mathsf{js}}$, i.e. a functionality that realizes $!\mathcal{F}_{\mathsf{pke}}$ while reusing the same key material. We actually show such a functionality for the functionality $\underline{\mathcal{F}_{\mathsf{pke}}}$, which is a variant of $\mathcal{F}_{\mathsf{pke}}$ in which each message is tagged. More precisely, the process $\underline{\mathcal{F}_{\mathsf{pke}}}$ is defined as $\mathcal{F}_{\mathsf{pke}}$, except that: (i) the functionality begins with the instructions $\mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{in}(io^1_{\mathsf{pke}}, sid)$ instead of $\mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}})$, (ii) each input of the form $\mathsf{in}(c, m)$ is replaced by $\mathsf{in}(c, \langle = sid, m\rangle)$, and (iii) each output of the form $\mathsf{out}(c, m)$ is replaced by $\mathsf{out}(c, \langle sid, m\rangle)$.

The $\mathcal{P}_{\mathsf{js}}$ functionality process launches one instance of the $\underline{\mathcal{F}_{\mathsf{pke}}}$ functionality that will be used in all protocol sessions. All the requests to the joint state functionality are received on the public channel io_{pke} in process $\mathcal{P}^1_{\mathsf{js}}$. They are then forwarded using the private IO channel $cont$ to $\mathcal{P}^2_{\mathsf{js}}$. The process $\mathcal{P}^2_{\mathsf{js}}$ shares the private channel io_{pke} with $\underline{\mathcal{F}_{\mathsf{pke}}}$ and forwards all the requests after concatenating the session identifier to the plaintext. Then the response is again forwarded to the process $\mathcal{P}^1_{\mathsf{js}}$ which outputs the result on the public channel io_{pke}.

$$
\begin{array}{lll}
\mathcal{P}_{\mathsf{js}} & := \nu cont.(\mathcal{P}^1_{\mathsf{js}} \mid \nu io_{\mathsf{pke}}, io^2_{\mathsf{pke}}.(P^2_{\mathsf{js}} \mid \_)) \\
\\
\mathcal{P}^1_{\mathsf{js}} & := \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{in}(io_1, sid).\mathsf{out}(cont, \langle sid, \text{INIT}\rangle). \\
& \quad \mathsf{in}(cont, (= \text{KEY}, pk)).\mathsf{out}(io^1_{\mathsf{pke}}, \langle sid, \text{KEY}, pk\rangle). \\
& \quad (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid \text{ let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid !\mathcal{P}^1_{\mathsf{js-dec}} \mid \\
& \quad !\mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{in}(io_1, sid).\mathsf{out}(io^1_{\mathsf{pke}}, \langle sid, \text{KEY}, pk\rangle). \\
& \quad (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid \text{ let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{P}^1_{\mathsf{js-enc}} \mid !\mathcal{P}^1_{\mathsf{js-dec}})) \\
\mathcal{P}^1_{\mathsf{js-enc}} & := \mathsf{in}(io^i_{\mathsf{pke}}, \langle = sid, = \text{ENC}, m\rangle).\ \mathsf{out}(cont, \langle sid, \text{ENC}, m\rangle). \\
& \quad \mathsf{in}(cont, \langle = \text{CIPHER}, c\rangle).\ \mathsf{out}(io^i_{\mathsf{pke}}, \langle sid, \text{CIPHER}, c\rangle) \\
\mathcal{P}^1_{\mathsf{js-dec}} & := \mathsf{in}(io^1_{\mathsf{pke}}, \langle = sid, = \text{DEC}, c\rangle).\ \mathsf{out}(cont, \langle sid, dec, c\rangle). \\
& \quad \mathsf{in}(cont, \langle = \text{PLAIN}, m\rangle).\ \mathsf{out}(io_{\mathsf{pke}}, \langle sid, \text{PLAIN}, m\rangle) \\
\\
\mathcal{P}^2_{\mathsf{js}} & := \mathsf{in}(cont, \langle sid, = \text{INIT}\rangle).\nu io'_{\mathsf{pke}}.\mathsf{out}(io_{\mathsf{pke}}, io'_{\mathsf{pke}}). \\
& \quad \mathsf{in}(io'_{\mathsf{pke}}, \langle = \text{KEY}, pk\rangle).\mathsf{out}(cont, \langle \text{KEY}, pk\rangle).\ (!\mathcal{P}^2_{\mathsf{js-enc}} \mid !\mathcal{P}^2_{\mathsf{js-dec}}) \\
\mathcal{P}^2_{\mathsf{js-enc}} & := \mathsf{in}(cont, \langle = sid, = \text{ENC}, m\rangle).\ \mathsf{out}(io'_{\mathsf{pke}}, \langle \text{ENC}, \langle sid, m\rangle\rangle). \\
& \quad \mathsf{in}(io'_{\mathsf{pke}}, \langle = \text{CIPHER}, c\rangle).\ \mathsf{out}(cont, \langle \text{CIPHER}, c\rangle) \\
\mathcal{P}^2_{\mathsf{js-dec}} & := \mathsf{in}(cont, \langle = sid, = \text{DEC}, c\rangle).\ \mathsf{out}(io'_{\mathsf{pke}}, \langle \text{DEC}, c\rangle). \\
& \quad \mathsf{in}(io'_{\mathsf{pke}}, \langle = \text{PLAIN}, m\rangle).\ \mathsf{out}(cont, \langle \text{PLAIN}, m\rangle)
\end{array}
$$

Fig. 3: Joint state IO-context

We now observe that the following joint state composition result holds. One instance of the encryption functionality can be used to emulate an unbounded number of such instances using the joint state process: $\mathcal{P}_{\mathsf{js}}[\mathcal{F}_{\mathsf{pke}}] \leq^{\mathsf{SS}} !\underline{\mathcal{F}_{\mathsf{pke}}}$.

This relation is witnessed by the adversary $\mathcal{A}_{\mathsf{js}}$ described in Figure 4 as we have that: $\mathcal{P}_{\mathsf{js}}[\mathcal{F}_{\mathsf{pke}}] \preceq \mathcal{A}_{\mathsf{js}}[!\underline{\mathcal{F}_{\mathsf{pke}}}]$. This adversary launches several functionalities with the

same key sk. However, note that the session identifier sid used to tag each encryption associated could be different. The value of these session identifiers is selected by the attacker.

$$\mathcal{A}_{\mathsf{js}} := \nu cs.(\mathcal{A}^1_{\mathsf{js}} \mid \nu net.(\mathcal{A}^2_{\mathsf{js}} \mid \_))$$

$$\mathcal{A}^1_{\mathsf{js}} := \mathsf{in}(cs, \text{INIT}).\mathsf{out}(net, \text{INIT}).\mathsf{in}(net, \langle = \text{ALGO}, sk, tag\rangle).\mathsf{out}(cs, \langle \text{ALGO}, sk, tag\rangle)$$

$$\begin{aligned}\mathcal{A}^2_{\mathsf{js}} := {} & \mathsf{in}(net, \langle sid, = \text{INIT}\rangle).\ \mathsf{out}(cs, \text{INIT}).\\ & \mathsf{in}(cs, \langle = \text{ALGO}, sk, tag\rangle).\ \mathsf{out}(net, \langle sid, \text{ALGO}, sk, \langle sid, tag\rangle\rangle).\\ & !\mathsf{in}(net, \langle sid', = \text{INIT}\rangle).\mathsf{out}(net, \langle sid', \text{ALGO}, sk, \langle sid', tag\rangle\rangle)\end{aligned}$$

Fig. 4: Joint state adversary

Note that it is crucial to introduce the ideal functionality. We indeed have that $\mathcal{P}_{\mathsf{js}}[\mathcal{P}_{\mathsf{pke}}] \leq^{\mathsf{SS}} \mathcal{P}_{\mathsf{js}}[\mathcal{F}_{\mathsf{pke}}] \leq^{\mathsf{SS}} !\underline{\mathcal{F}_{\mathsf{pke}}}$ as well as $!\underline{\mathcal{P}_{\mathsf{pke}}} \leq^{\mathsf{SS}} !\underline{\mathcal{F}_{\mathsf{pke}}}$ (where $\underline{\mathcal{P}_{\mathsf{pke}}}$ is defined from $\mathcal{P}_{\mathsf{pke}}$ in the same way as $\underline{\mathcal{F}_{\mathsf{pke}}}$ from $\mathcal{F}_{\mathsf{pke}}$). However, $\mathcal{P}_{\mathsf{js}}[\mathcal{P}_{\mathsf{pke}}] \not\leq^{\mathsf{SS}} !\underline{\mathcal{P}_{\mathsf{pke}}}$. In particular $!\underline{\mathcal{P}_{\mathsf{pke}}}$ will provide multiple public keys while $\mathcal{P}_{\mathsf{js}}[\mathcal{P}_{\mathsf{pke}}]$ only provides a single one. Taking the more abstract ideal functionality allows this to be avoided by a simulator that chooses the same secret key for each instance of the functionality.

4.2 Mutual authentication

Ideal functionality for mutual authentication. The $\mathcal{F}_{\mathsf{auth}}$ functionality is described in Figure 5 and works as follows. Both the initiator ($\mathcal{F}_{\mathsf{init}}$) and the responder ($\mathcal{F}_{\mathsf{resp}}$) receive a request for mutual authentication on their io channel. They forward this request to the adversary and, if both parties are honest, to a trusted host $\mathcal{F}_{\mathsf{th}}$ which compares these requests and authorizes going further if they match. Eventually, when the adversary asks to finish the protocol, then both participants complete the protocol session.

Realization of mutual authentication. The realization of $\mathcal{F}_{\mathsf{auth}}$ based on the Needham-Schroeder-Lowe protocol is described in Figure 6. For simplicity we consider only two honest identities (ID-A and ID-B) and one adversary identity (ID-I). We suppose that these are the only terms of sort id and that the type system only allows these values for the variables id_i. The public key infrastructure is modelled as local tables of the participants which are used to retrieve the channel names associated to the $\mathcal{F}_{\mathsf{pke}}$ functionality of a given identity. We define F^X_{pke} to be $F^X_{\mathsf{pke}}[io_{\mathsf{pke}} \mapsto io^X_{\mathsf{pke}}]$ where $[io_{\mathsf{pke}} \mapsto io^X_{\mathsf{pke}}]$ denotes the replacement of io_{pke} by io^X_{pke}.

We have that $\mathcal{P}_{\mathsf{nsl}} \leq^{\mathsf{SS}} \mathcal{F}_{\mathsf{auth}}$ by showing that $\mathcal{P}_{\mathsf{nsl}} \preceq \mathcal{S}[\mathcal{F}_{\mathsf{auth}}]$ where $\mathcal{S} = \nu net.(\_ \mid \nu io^2_{\mathsf{pke}}.\mathcal{P}_{\mathsf{nsl}}[io_1 \mapsto net][io_2 \mapsto net])$. Intuitively, when $\mathcal{F}_{\mathsf{auth}}$ sends the initialization on channel net then the $\mathcal{P}_{\mathsf{nsl}}$ protocol is executed. If the protocol succeeds then it sends the message $\langle \text{FINISH}, sid, id_1, id_2\rangle$ on channel net. The restriction on io^2_{pke} is to avoid that the environment uses the encryption functionality and it ensures that $fn(\mathcal{S}) \cap \mathsf{IO} = \emptyset$.

$$
\begin{aligned}
\mathcal{F}_{\mathsf{auth}} &:= \nu c_1.\nu c_2.(\mathcal{F}_{\mathsf{init}} \mid \mathcal{F}_{\mathsf{resp}} \mid \mathcal{F}_{\mathsf{th}}) \\
\mathcal{F}_{\mathsf{init}} &:= \mathsf{in}(io_1, \langle \text{INIT}, sid, id_1, id_2\rangle).\mathsf{out}(net, \langle \text{INIT}, sid, id_1, id_2\rangle). \\
&\quad \text{if } id_2 = \text{ID-I} \text{ then } \mathsf{in}(net, \langle \text{FINISH}, = sid, = id_1, = id_2\rangle). \\
&\qquad\qquad \mathsf{out}(io_1, \langle finish, sid, id_1, id_2\rangle) \\
&\qquad \text{else } \mathsf{out}(c_1, \langle \text{COMPARE}, sid, id_1, id_2\rangle). \\
&\qquad\quad \mathsf{in}(c_1, \langle = \text{OK}, = sid, = id_1, = id_2\rangle). \\
&\qquad\quad \mathsf{in}(net, \langle \text{FINISH}, = sid, = id_1, = id_2\rangle). \\
&\qquad\quad \mathsf{out}(io_1, \langle \text{FINISH}, sid, id_1, id_2\rangle) \\
\mathcal{F}_{\mathsf{resp}} &:= \mathsf{in}(io_2, \langle \text{INIT}, sid, id_1, id_2\rangle).\mathsf{out}(net, \langle \text{INIT}, sid, id_1, id_2\rangle). \\
&\quad \text{if } id_1 = \text{ID-I} \text{ then } \mathsf{in}(net, \langle \text{FINISH}, = sid, = id_1, = id_2\rangle). \\
&\qquad\qquad \mathsf{out}(io_2, \langle \text{FINISH}, sid, id_1, id_2\rangle) \\
&\qquad \text{else } \mathsf{out}(c_2, \langle \text{COMPARE}, sid, id_1, id_2\rangle). \\
&\qquad\quad \mathsf{in}(c_2, \langle = \text{OK}, = sid, = id_1, = id_2\rangle). \\
&\qquad\quad \mathsf{in}(net, \langle \text{FINISH}, = sid, = id_1, = id_2\rangle). \\
&\qquad\quad \mathsf{out}(io_2, \langle \text{FINISH}, sid, id_1, id_2\rangle) \\
\mathcal{F}_{\mathsf{th}} &:= \mathsf{in}(c_1, \langle = \text{COMPARE}, sid, id_1, id_2\rangle).\mathsf{in}(c_2, \langle = \text{COMPARE}, = sid, = id_1, = id_2\rangle). \\
&\quad \mathsf{out}(c_1, \langle \text{OK}, sid, id_1, id_2\rangle).\mathsf{out}(c_2, \langle \text{OK}, sid, id_1, id_2\rangle)
\end{aligned}
$$

Fig. 5: Mutual authentication functionality

4.3 From one to many sessions

We have shown that $\mathcal{P}_{\mathsf{nsl}} \leq^{\mathsf{SS}} \mathcal{F}_{\mathsf{auth}}$. This result only shows that $\mathcal{P}_{\mathsf{nsl}}$ is as secure as $\mathcal{F}_{\mathsf{auth}}$ for a single session of the protocol. By Proposition 3 we have that $!\mathcal{P}_{\mathsf{nsl}} \leq^{\mathsf{SS}} !\mathcal{F}_{\mathsf{auth}}$ but this does not correspond to the expected security for an unbounded number of sessions, as each session uses a different key. To show that $!\mathcal{F}_{\mathsf{auth}}$ can be realized with shared key material we use our joint state result. To apply this result we need the following technical lemma.

Lemma 5. *Let n be a name and c be a channel name such that $c \notin fn(P) \cup fn(Q)$.*

$$\nu c.\,![\nu n.(\mathsf{out}(c, n) \mid P) \mid \mathsf{in}(c, x).Q] \preceq_\ell !\,\nu c.[\nu n.(\mathsf{out}(c, n) \mid P) \mid \mathsf{in}(c, x).Q].$$

Applying this lemma twice on $!\mathcal{P}_{\mathsf{nsl}}$ we obtain that

$$\nu io^A_{\mathsf{pke}}, io^B_{\mathsf{pke}}.!\ (\underline{\mathcal{F}^A_{\mathsf{pke}}} \mid \underline{\mathcal{F}^B_{\mathsf{pke}}} \mid \nu io^a_{\mathsf{pke}}, io^b_{\mathsf{pke}}. \\ (\mathsf{out}(io_{\mathsf{pke}}, io^a_{\mathsf{pke}}) \mid \mathsf{out}(io_{\mathsf{pke}}, io^b_{\mathsf{pke}}) \mid \mathcal{P}_{\mathsf{init}} \mid \mathcal{P}_{\mathsf{resp}})) \leq^{\mathsf{SS}} !\mathcal{P}_{\mathsf{nsl}}$$

Applying Lemma 1 we have that

$$\nu io^A_{\mathsf{pke}}, io^B_{\mathsf{pke}}.\ (!\underline{\mathcal{F}^A_{\mathsf{pke}}} \mid !\underline{\mathcal{F}^B_{\mathsf{pke}}} \mid !\nu io^a_{\mathsf{pke}}, io^b_{\mathsf{pke}}. \\ (\mathsf{out}(io_{\mathsf{pke}}, io^a_{\mathsf{pke}}) \mid \mathsf{out}(io_{\mathsf{pke}}, io^b_{\mathsf{pke}}) \mid \mathcal{P}_{\mathsf{init}} \mid \mathcal{P}_{\mathsf{resp}})) \leq^{\mathsf{SS}} !\mathcal{P}_{\mathsf{nsl}}$$

Now we can use the joint state result to obtain that:

$$\nu io^A_{\mathsf{pke}}, io^B_{\mathsf{pke}}.\ (\mathcal{P}^A_{\mathsf{js}}[\mathcal{F}^A_{\mathsf{pke}}] \mid \mathcal{P}^B_{\mathsf{js}}[\mathcal{F}^B_{\mathsf{pke}}] \mid !\nu io^a_{\mathsf{pke}}, io^b_{\mathsf{pke}}. \\ (\mathsf{out}(io_{\mathsf{pke}}, io^a_{\mathsf{pke}}) \mid \mathsf{out}(io_{\mathsf{pke}}, io^b_{\mathsf{pke}}) \mid \mathcal{P}_{\mathsf{init}} \mid \mathcal{P}_{\mathsf{resp}})) \leq^{\mathsf{SS}} !\mathcal{P}_{\mathsf{nsl}}$$

where $\mathcal{P}^X_{\mathsf{js}} = \mathcal{P}_{\mathsf{js}}[io_{\mathsf{pke}} \mapsto io^X_{\mathsf{pke}}]$. This corresponds to a result for unbounded number of sessions with shared key material. Note that the joint state context uses a tagging mechanism and adds a tag to each encryption.

$$
\begin{aligned}
\mathcal{P}_{\mathsf{nsl}} \;&:= \nu io^{A}_{\mathsf{pke}}, io^{B}_{\mathsf{pke}}, io^{a}_{\mathsf{pke}}, io^{b}_{\mathsf{pke}}.\\
&\quad (\underline{\mathcal{F}_{\mathsf{pke}}}^{A} \mid \underline{\mathcal{F}_{\mathsf{pke}}}^{B} \mid \mathsf{out}(io_{\mathsf{pke}}, io^{a}_{\mathsf{pke}}) \mid \mathsf{out}(io_{\mathsf{pke}}, io^{b}_{\mathsf{pke}}) \mid \mathcal{P}_{\mathsf{init}} \mid \mathcal{P}_{\mathsf{resp}})\\
\mathcal{P}_{\mathsf{init}} \;&:= \mathsf{in}(io_1, \langle \text{INIT}, sid, id_1, id_2\rangle).\\
&\quad \text{if } id_1 = \text{ID-A then let } io^{\mathsf{init}}_{\mathsf{pke}} = io^{a}_{\mathsf{pke}} \text{ in } \mathcal{P}^{1}_{\mathsf{init}}\\
&\qquad\qquad \text{else if } id_1 = \text{ID-B then let } io^{\mathsf{init}}_{\mathsf{pke}} = io^{b}_{\mathsf{pke}} \text{ in } \mathcal{P}^{1}_{\mathsf{init}}\\
\mathcal{P}^{1}_{\mathsf{init}} \;&:= \text{if } id_2 = \text{ID-A then let } io^{\mathsf{resp}}_{\mathsf{pke}} = io^{a}_{\mathsf{pke}} \text{ in } \mathcal{P}^{2}_{\mathsf{init}}\\
&\qquad\qquad \text{else if } id_2 = \text{ID-B then let } io^{\mathsf{resp}}_{\mathsf{pke}} = io^{b}_{\mathsf{pke}} \text{ in } \mathcal{P}^{2}_{\mathsf{init}}\\
&\qquad\qquad\qquad\qquad \text{else if } id_2 = \text{ID-I then let } io^{\mathsf{resp}}_{\mathsf{pke}} = io^{i}_{\mathsf{pke}} \text{ in } \mathcal{P}^{2}_{\mathsf{init}}\\
\mathcal{P}^{2}_{\mathsf{init}} \;&:= \mathsf{out}(io^{\mathsf{init}}_{\mathsf{pke}}, sid).\\
&\quad (\text{*Msg 1*}) \quad \nu na.\mathsf{out}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle sid, \text{ENC}, \langle na, id_1\rangle\rangle).\ \mathsf{in}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle = sid, = \text{CIPHER}, x_1\rangle).\\
&\quad \mathsf{out}(net_{\mathsf{nsl}}, x_1).\\
&\quad (\text{*Msg 2*}) \quad \mathsf{in}(net_{\mathsf{nsl}}, x_2).\\
&\quad \mathsf{out}(io^{\mathsf{init}}_{\mathsf{pke}}, \langle sid, \text{DEC}, x_2\rangle).\ \mathsf{in}(io^{\mathsf{init}}_{\mathsf{pke}}, \langle = sid, = \text{PLAIN}, \langle = na, y_{nb}, = id_2\rangle\rangle).\\
&\quad (\text{*Msg 3*}) \quad \mathsf{out}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle sid, \text{ENC}, y_{nb}\rangle).\mathsf{in}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle = sid, = \text{CIPHER}, x_3\rangle).\\
&\quad \mathsf{out}(net_{\mathsf{nsl}}, x_3).\\
&\quad \mathsf{out}(io_1, \langle \text{FINISH}, sid, id_1, id_2\rangle)\\
\mathcal{P}_{\mathsf{resp}} \;&:= \mathsf{in}(io_2, \langle = \text{INIT}, sid, id_1, id_2\rangle).\\
&\quad \text{if } id_1 = \text{ID-A then let } io^{\mathsf{init}}_{\mathsf{pke}} = io^{a}_{\mathsf{pke}} \text{ in } \mathcal{P}^{1}_{\mathsf{resp}}\\
&\qquad\qquad \text{else if } id_1 = \text{ID-B then let } io^{\mathsf{init}}_{\mathsf{pke}} = io^{b}_{\mathsf{pke}} \text{ in } \mathcal{P}^{1}_{\mathsf{resp}}\\
&\qquad\qquad\qquad\qquad \text{else if } id_1 = \text{ID-I then let } io^{i}_{\mathsf{pke}} = io^{i}_{\mathsf{pke}} \text{ in } \mathcal{P}^{1}_{\mathsf{resp}}\\
\mathcal{P}^{1}_{\mathsf{resp}} \;&:= \text{if } id_2 = \text{ID-A then let } io^{\mathsf{resp}}_{\mathsf{pke}} = io^{a}_{\mathsf{pke}} \text{ in } \mathcal{P}^{2}_{\mathsf{resp}}\\
&\qquad\qquad \text{else if } id_2 = \text{ID-B then let } io^{\mathsf{resp}}_{\mathsf{pke}} = io^{b}_{\mathsf{pke}} \text{ in } \mathcal{P}^{2}_{\mathsf{resp}}\\
\mathcal{P}^{2}_{\mathsf{resp}} \;&:= \mathsf{out}(io^{\mathsf{resp}}_{\mathsf{pke}}, sid).\\
&\quad (\text{*Msg 1*}) \quad \mathsf{in}(net_{\mathsf{nsl}}, x_1).\mathsf{out}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle sid, \text{DEC}, x_1\rangle).\mathsf{in}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle = sid, = \text{PLAIN}, \langle x_{na}, = id_1\rangle\rangle).\\
&\quad (\text{*Msg 2*}) \quad \nu nb.\mathsf{out}(io^{\mathsf{init}}_{\mathsf{pke}}, \langle sid, \text{ENC}, \langle x_{na}, nb, id_1\rangle\rangle).\mathsf{in}(io^{\mathsf{init}}_{\mathsf{pke}}, \langle = sid, = \text{CIPHER}, x_2\rangle).\\
&\quad \mathsf{out}(net_{\mathsf{nsl}}, x_2).\\
&\quad (\text{*Msg 3*}) \quad \mathsf{in}(net_{\mathsf{nsl}}, x_3).\mathsf{out}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle sid, \text{DEC}, x_3\rangle).\mathsf{in}(io^{\mathsf{resp}}_{\mathsf{pke}}, \langle = sid, = \text{PLAIN}, y_{nb}\rangle).\\
&\quad \text{if } y_{nb} = nb \text{ then } \mathsf{out}(io_2, \langle \text{FINISH}, sid, id_1, id_2\rangle)
\end{aligned}
$$

Fig. 6: Mutual authentication realization

5 Conclusions

This paper proposes a symbolic framework for the analysis of security protocols along the lines of the simulation based security approach, while adopting the applied pi calculus as basic layer. We state central definitions and security notions, show general composition theorems and specific joint-state composition results for asymmetric encryption, and illustrate their use in the analysis of a mutual authentication protocol.

This framework brings the benefits of the secure composition theorems associated to simulation based security into the symbolic world, and opens the path to the analysis of more sophisticated protocols that can naturally specified by the behavior of an ideal functionality, e.g., electronic commerce or voting protocols. At a more fundamental level, our framework makes use of preorder notions, which can be established by labeled simulation. While the use of labeled bisimulations is quite common in the applied pi

calculus and has been integrated in automatic provers, the automation of proofs relying on labeled simulation appears as an interesting challenge for future works.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symp. on Principles of Programming Languages (POPL'01)*. ACM, 2001.
2. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report 149, SRC, 1998.
3. A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th Int. Conference on Computer Aided Verification (CAV'05)*, LNCS. Springer, 2005.
4. M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.
5. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, 2001.
6. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, 2001.
7. R. Canetti, L. Cheung, D. Kaynar, N. Lynch, and O. Pereira. Compositional security for Task-PIOAs. In *Proc. 20th Computer Security Foundations Symposium (CSF'07)*, 2007.
8. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. Theory of Cryptography Conference (TCC'06)*, LNCS. Springer, 2006.
9. I. Cervesato, A. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key kerberos. *Information and Computation*, 206(2-4):402–424, 2008.
10. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. 20th Int. Conference on Computer Aided Verification (CAV'08)*, LNCS, 2008.
11. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, 2004.
12. S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, LNCS. Springer, 2007.
13. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game: A completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC'87)*. ACM Press, 1987.
14. J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, 2000.
15. R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006.
16. R. Küsters, A. Datta, J. C. Mitchell, and A. Ramanathan. On the relationships between notions of simulation-based security. *Journal of Cryptology*, 21(4):492–546, 2008.
17. R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digitial Signature Functionalities with Local Computation. In *Proc. 21st IEEE Computer Security Foundations Symposium (CSF'08)*, 2008.
18. G. Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.
19. P. Mateus, J. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time calculus. In *Proc. 14th Conference on Concurrency Theory (CONCUR'03)*, LNCS. Springer, 2003.

A Proof of Proposition 1

The proof relies on the following two lemmas.

Lemma 6. *Let $\mathcal{R}$ be the relation on closed extended processes defined as follows:*

$$\mathcal{R} = \preceq_\ell \cup \{(A, B) \mid A \equiv \tilde{A} \mid \{M/x\},\ B \equiv \tilde{B} \mid \{M/x\},\ \textit{and}\ \tilde{A} \preceq_\ell \tilde{B}\}.$$

We have that $\mathcal{R}$ is a labelled simulation.

Proof. Let A and B be two closed extended processes such that $A\ \mathcal{R}\ B$. Either $A \preceq_\ell B$ and we easily conclude. Otherwise, we have that there exist two closed extended processes $\tilde{A}$, $\tilde{B}$, an active substitution $\{M/x\}$ such that:

$$A \equiv \tilde{A} \mid \{M/x\}, B \equiv \tilde{B} \mid \{M/x\}, \text{and } \tilde{A} \preceq_\ell \tilde{B}.$$

We show that the 3 points of the definition of labelled simulation hold.

1. $A \approx_s B$. Indeed, we have that $\tilde{A} \approx_s \tilde{B}$, thus $\tilde{A} \mid \{M/x\} \approx_s \tilde{B} \mid \{M/x\}$. The result easily follows.
2. If $A \to A'$ then $B \to^* B'$ for some B' such that $A'\ \mathcal{R}\ B'$.
   Since $A \equiv \tilde{A} \mid \{M/x\}$, we have that $A' \equiv \tilde{A'} \mid \{M/x\}$ for some closed extended process $\tilde{A'}$ such that $\tilde{A} \to \tilde{A'}$. Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B'}$ such that $\tilde{B} \to^* \tilde{B'}$ and $\tilde{A'} \preceq_\ell \tilde{B'}$. Let $B' = \tilde{B'} \mid \{M/x\}$. We have that $B \equiv \tilde{B} \mid \{M/x\} \to^* \tilde{B'} \mid \{M/x\} \stackrel{\mathsf{def}}{=} B'$ and $A'\ \mathcal{R}\ B'$ by definition of $\mathcal{R}$.
3. If $A \xrightarrow{\alpha} A'$ with $fv(\alpha) \subseteq \mathrm{dom}(A)$ and $bn(\alpha) \cap fn(B) = \emptyset$, then $B \to^* \xrightarrow{\alpha} \to^* B'$ for some B' such that $A'\ \mathcal{R}\ B'$.
   Since $A \equiv \tilde{A} \mid \{M/x\}$, we have that $A' \equiv \tilde{A'} \mid \{M/x\}$ for some closed extended process $\tilde{A'}$ such that $\tilde{A} \xrightarrow{\alpha'} \tilde{A'}$ with $\alpha' = \alpha[x \mapsto M]$. Note that $fv(\alpha') \subseteq \mathrm{dom}(\tilde{A})$ and we can assume that $bn(\alpha') \cap fn(\tilde{B}) = \emptyset$. Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B'}$ such that $\tilde{B} \to^* \xrightarrow{\alpha'} \to^* \tilde{B'}$ and $\tilde{A'} \preceq_\ell \tilde{B'}$. Let $B' = \tilde{B'} \mid \{M/x\}$. We have that $B \equiv \tilde{B} \mid \{M/x\} \to^* \xrightarrow{\alpha} \to^* \tilde{B'} \mid \{M/x\} \stackrel{\mathsf{def}}{=} B'$ and $A'\ \mathcal{R}\ B'$ by definition of $\mathcal{R}$. This allows us to conclude. □

Lemma 7. *Let $\mathcal{R}$ be the relation on closed extended processes defined as follows:*

$$\mathcal{R} = \preceq_\ell \cup \{(A, B) \mid A \equiv \nu\tilde{u}.(\tilde{A} \mid P),\ \nu\tilde{u}.(\tilde{B} \mid P),\ \textit{and}\ \tilde{A} \preceq_\ell \tilde{B}\}.$$

We have that $\mathcal{R}$ is a labelled simulation.

Proof. Let A and B be two closed extended processes such that $A\ \mathcal{R}\ B$. Either $A \preceq_\ell B$ and we easily conclude. Otherwise, we have that there exist two closed extended processes $\tilde{A}$, $\tilde{B}$, a plain process P (with $fv(P) \subseteq \mathrm{dom}(A)$) and a sequence of metavariables $\tilde{u}$ such that:

$$A \equiv \nu\tilde{u}.(\tilde{A} \mid P), B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \text{ and } \tilde{A} \preceq_\ell \tilde{B}.$$

We show that the 3 points of the definition of labelled simulation hold. First, we have that $A \approx_s B$. Indeed, we have that $\tilde{A} \approx_s \tilde{B}$, thus since $\approx_s$ is closed by application of evaluation context, we deduce that $\nu\tilde{u}.(\tilde{A} \mid P) \approx_s \nu\tilde{u}.(\tilde{B} \mid P)$, and thus $A \approx_s B$. Now, we distinguish several cases depending on the form of the label α involved in the reduction $A \xrightarrow{\alpha} A'$.

1. $\alpha = \nu x.\mathsf{out}(c, x)$. We distinguish two cases:
   (a) $A' \equiv \nu\tilde{u}.(\tilde{A'} \mid P)$ for some closed extended process $\tilde{A'}$ and $\tilde{A} \xrightarrow{\nu x.out(c,x)} \tilde{A'}$ with $c, x \notin \tilde{u}$. (Note that $c, x \notin \tilde{u}$ can be assumed w.l.o.g.: from $A \equiv \nu\tilde{u}.(\tilde{A} \mid P)$, $B \equiv \nu\tilde{u}.(\tilde{B} \mid P)$ and $\tilde{A} \preceq_\ell \tilde{B}$ we obtain by α-conversion that $A \equiv \nu\tilde{u}_1.(\tilde{A}_1 \mid P_1)$, $B \equiv \nu\tilde{u}_1.(\tilde{B}_1 \mid P_1)$ for some $\tilde{u}_1$ with $c, x \notin \tilde{u}_1$ and as $\preceq_\ell$ is closed under injective renaming of free names we have that $\tilde{A}_1 \preceq_\ell \tilde{B}_1$).
   Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B'}$ such that $\tilde{B} \rightarrow^* \xrightarrow{\nu x.out(c,x)} \rightarrow^* \tilde{B'}$ and $\tilde{A'} \preceq_\ell \tilde{B'}$. Let $B' = \nu\tilde{u}.(\tilde{B'} \mid P)$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{\nu x.out(c,x)} \rightarrow^* \nu\tilde{u}.(\tilde{B'} \mid P) \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (b) $A' \equiv \nu\tilde{u}.\nu\tilde{n}.(\tilde{A} \mid P' \mid \{M/x\})$ for some plain process P' and some sequence of names $\tilde{n}$ such that $\tilde{n} \cap fn(\tilde{B}) = \emptyset$ and $P \xrightarrow{\nu x.out(c,x)} \nu\tilde{n}.(P' \mid \{M/x\})$ with $c, x \notin \tilde{u} \cup \tilde{n}$.
   Let $B' = \nu\tilde{u}.\nu\tilde{n}.(\tilde{B} \mid P' \mid \{M/x\})$. We have that $\tilde{A} \mid \{M/x\} \preceq_\ell \tilde{B} \mid \{M/x\}$ thanks to Lemma 6 and the fact that $\tilde{A} \preceq_\ell \tilde{B}$. Thus $\tilde{A'} \mathcal{R} \tilde{B'}$ by definition of $\mathcal{R}$. We have also that:
   $$B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{\nu x.out(c,x)} \rightarrow^* \nu\tilde{u}.\nu\tilde{n}.(\tilde{B} \mid P' \mid \{M/x\}) \stackrel{\mathsf{def}}{=} B'.$$
2. $\alpha = \mathsf{out}(c, a)$. We distinguish two cases:
   (a) $A' \equiv \nu\tilde{u}.(\tilde{A'} \mid P)$ and $\tilde{A} \xrightarrow{out(c,a)} \tilde{A'}$ with $a, c \notin \tilde{u}$.
   Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B'}$ such that $\tilde{B} \rightarrow^* \xrightarrow{out(c,a)} \rightarrow^* \tilde{B'}$ and $\tilde{A'} \preceq_\ell \tilde{B'}$. Let $B' = \nu\tilde{u}.(\tilde{B'} \mid P)$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{out(c,a)} \rightarrow^* \nu\tilde{u}.(\tilde{B'} \mid P) \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (b) $A' \equiv \nu\tilde{u}.(\tilde{A} \mid P')$ and $P \xrightarrow{out(c,a)} P'$ with $a, c \notin \tilde{u}$.
   Let $B' = \nu\tilde{u}.(\tilde{B} \mid P')$. By definition of $\mathcal{R}$, we have that $\tilde{A'} \mathcal{R} \tilde{B'}$. We have also that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{out(c,a)} \rightarrow^* \nu\tilde{u}.(\tilde{B} \mid P') \stackrel{\mathsf{def}}{=} B'$.
3. $\alpha = \mathsf{in}(c, M)$. We distinguish two cases:
   (a) $A' \equiv \nu\tilde{u}.(\tilde{A'} \mid P)$ and $\tilde{A} \xrightarrow{in(c,M)} \tilde{A'}$ with $c \notin \tilde{u}$ and $\tilde{u}$ do not occur in M.
   Since $\tilde{A} \preceq_\ell \tilde{B}$ and $fv(M) \subseteq \mathrm{dom}(A) \subseteq \mathrm{dom}(\tilde{A})$, we know that there exists a closed extended process $\tilde{B'}$ such that $\tilde{B} \rightarrow^* \xrightarrow{in(c,M)} \rightarrow^* \tilde{B'}$ and $\tilde{A'} \preceq_\ell \tilde{B'}$. Let $B' = \nu\tilde{u}.(\tilde{B'} \mid P)$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{in(c,M)} \rightarrow^* \nu\tilde{u}.(\tilde{B'} \mid P) \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (b) $A' \equiv \nu\tilde{u}.(\tilde{A} \mid P')$ and $P \xrightarrow{in(c,M)} P'$ with $c \notin \tilde{u}$ and $\tilde{u}$ do not occur in M.
   Let $B' = \nu\tilde{u}.(\tilde{B} \mid P')$. By definition of $\mathcal{R}$, we have that $\tilde{A'} \mathcal{R} \tilde{B'}$. We have also that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{in(c,M)} \rightarrow^* \nu\tilde{u}.(\tilde{B} \mid P') \stackrel{\mathsf{def}}{=} B'$.

4. $\alpha = \nu a.\mathsf{out}(c, a)$ and $a \notin fn(B)$. We distinguish four cases:
   (a) $A' \equiv \nu\tilde{u}.(\tilde{A}' \mid P)$ and $\tilde{A} \xrightarrow{\nu a.out(c,a)} \tilde{A}'$ with $c \notin \tilde{u}$. We have also to assume that $a \notin fn(\tilde{B})$. (Note again that we can assume this w.l.o.g.: if $a \notin fn(B)$ and $a \notin \tilde{u}$ then we can assume that $a \notin fn(\tilde{B})$. We can always assume $a \notin \tilde{u}$ as explained previously.)
   Since $\tilde{A} \preceq_\ell \tilde{B}$ and $a \notin fn(\tilde{B})$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{\nu a.out(c,a)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}.(\tilde{B}' \mid P)$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{\nu a.out(c,a)} \rightarrow^* \nu\tilde{u}.(\tilde{B}' \mid P) \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (b) $A' \equiv \nu\tilde{u}.(\tilde{A} \mid P')$ and $P \xrightarrow{\nu a.out(c,a)} P'$ with $c \notin \tilde{u}$.
   Let $B' = \nu\tilde{u}.(\tilde{B} \mid P')$. By definition of $\mathcal{R}$, we have that $\tilde{A}' \mathcal{R} \tilde{B}'$. We have also that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{\nu a.out(c,a)} \rightarrow^* \nu\tilde{u}.(\tilde{B} \mid P') \stackrel{\mathsf{def}}{=} B'$.
   (c) $A' \equiv \nu\tilde{u}'.(\tilde{A}' \mid P)$ and $\tilde{A} \xrightarrow{out(c,a)} \tilde{A}'$ with $c \notin \tilde{u}$, $a \in \tilde{u}$ and $\tilde{u}' = \tilde{u} \setminus \{a\}$.
   Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{out(c,a)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}'.(\tilde{B}' \mid P)$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{\nu a.out(c,a)} \rightarrow^* \nu\tilde{u}'.(\tilde{B}' \mid P) \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (d) $A' \equiv \nu\tilde{u}'.(\tilde{A} \mid P')$ and $P \xrightarrow{out(c,a)} P'$ with $c \notin \tilde{u}$, $a \in \tilde{u}$ and $\tilde{u}' = \tilde{u} \setminus \{a\}$.
   Let $B' = \nu\tilde{u}'.(\tilde{B} \mid P')$. By definition of $\mathcal{R}$, we have that $A' \mathcal{R} B'$. We have also that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \xrightarrow{\nu a.out(c,a)} \rightarrow^* \nu\tilde{u}'.(\tilde{B} \mid P') \stackrel{\mathsf{def}}{=} B'$.

5. $\alpha = \tau$. We distinguish 8 cases:
   (a) $A' \equiv \nu\tilde{u}.(\tilde{A}' \mid P)$ and $\tilde{A} \rightarrow \tilde{A}'$.
   Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}.(\tilde{B}' \mid P)$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow \nu\tilde{u}.(\tilde{B}' \mid P) \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (b) $A' \equiv \nu\tilde{u}.(\tilde{A} \mid P')$ and $P \rightarrow P'$.
   Let $B' = \nu\tilde{u}.(\tilde{B} \mid P')$. By definition of $\mathcal{R}$, we have that $A' \mathcal{R} B'$. We have also that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow \nu\tilde{u}.(\tilde{B} \mid P') \stackrel{\mathsf{def}}{=} B'$.
   (c) $A' \equiv \nu\tilde{u}.(\tilde{A}' \mid P')$ with $\tilde{A} \xrightarrow{out(c,a)} \tilde{A}'$ and $P \xrightarrow{in(c,a)} P'$.
   Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{out(c,a)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}.(\tilde{B}' \mid P')$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \nu\tilde{u}.(\tilde{B}' \mid P') \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (d) $A' \equiv \nu\tilde{u}, a.(\tilde{A}' \mid P')$ with $\tilde{A} \xrightarrow{\nu a.out(c,a)} \tilde{A}'$ and $P \xrightarrow{in(c,a)} P'$.
   Since $\tilde{A} \preceq_\ell \tilde{B}$ and $a \notin fn(\tilde{B})$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{\nu a.out(c,a)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}, a.(\tilde{B}' \mid P')$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \nu\tilde{u}, a.(\tilde{B}' \mid P') \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.
   (e) $A' \equiv \nu\tilde{u}.(\tilde{A}' \mid P')$ with $\tilde{A} \xrightarrow{in(c,a)} \tilde{A}'$ and $P \xrightarrow{out(c,a)} P'$.

Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{in(c,a)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}.(\tilde{B}' \mid P')$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \nu\tilde{u}.(\tilde{B}' \mid P') \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.

(f) $A' \equiv \nu\tilde{u}, a.(\tilde{A}' \mid P')$ with $\tilde{A} \xrightarrow{in(c,a)} \tilde{A}'$ and $P \xrightarrow{\nu a.out(c,a)} P'$.
Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{in(c,a)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}, a.(\tilde{B}' \mid P')$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \nu\tilde{u}.(\tilde{B}' \mid P') \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.

(g) $A' \equiv \nu\tilde{u}, x.(\tilde{A}' \mid P')$ with $\tilde{A} \xrightarrow{\nu x.out(c,x)} \tilde{A}'$ and $P \xrightarrow{in(c,x)} P'$.
Since $\tilde{A} \preceq_\ell \tilde{B}$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{\nu x.out(c,x)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}, x.(\tilde{B}' \mid P')$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \nu\tilde{u}, x.(\tilde{B}' \mid P') \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$.

(h) $A' \equiv \nu\tilde{u}.\nu\tilde{n}.(\tilde{A}' \mid P')$ with $\tilde{A} \xrightarrow{in(c,M)} \tilde{A}'$ and $P \xrightarrow{\nu x.out(c,x)} \nu\tilde{n}.(P' \mid \{M/x\})$.
Since $\tilde{A} \preceq_\ell \tilde{B}$ and $fv(M) \subseteq fv(P) \subseteq \mathrm{dom}(A) \subseteq \mathrm{dom}(\tilde{A})$, we know that there exists a closed extended process $\tilde{B}'$ such that $\tilde{B} \rightarrow^* \xrightarrow{in(c,M)} \rightarrow^* \tilde{B}'$ and $\tilde{A}' \preceq_\ell \tilde{B}'$. Let $B' = \nu\tilde{u}.\nu\tilde{n}.(\tilde{B}' \mid P')$. We have that $B \equiv \nu\tilde{u}.(\tilde{B} \mid P) \rightarrow^* \nu\tilde{u}.\nu\tilde{n}.(\tilde{B}' \mid P') \stackrel{\mathsf{def}}{=} B'$ and $A' \mathcal{R} B'$ by definition of $\mathcal{R}$. □

Proposition 1. *Let A and B be two extended processes such that $A \preceq_\ell B$. We have that $C[A] \preceq_\ell C[B]$ for all closing evaluation context $C[\_]$.*

Proof. We prove this result by structural induction on $C[\_]$.
Base case: $C = \_$ In such a case we easily conclude.
Induction step. We distinguish several cases depending on the form of C.

- $C[\_] = \nu u.C'[\_]$. In such a case, thanks to our induction hypothesis, we have that $C'[A] \preceq_\ell C'[B]$. Then, thanks to Lemma 7, we easily deduce that $C[A] \preceq_\ell C[B]$.
- $C[\_] = P \mid C'[\_]$. We conclude as in the previous case.
- $C[\_] = \{M/x\} \mid C'[\_]$. In such a case, thanks to our induction hypothesis, we have that $C'[A] \preceq_\ell C'[B]$. Then, thanks to Lemma 6, we easily deduce that $C[A] \preceq_\ell C[B]$.

This allows us to conclude the proof. □

B Proofs of Section 3

Lemma 3. *Let $\mathcal{F}$ be a functionality and $D[\_]$ the dummy adversary for $\mathcal{F}$: $\mathcal{F} \preceq D[\mathcal{F}]$.*

Proof. We define the following relation $\mathcal{R}$ on closed extended processes

$$\mathcal{R} \;=\; \preceq_\ell \;\cup\; \{(A, D[A]) \mid \exists \mathcal{F}.\ \mathsf{fnet}(A) \subseteq \mathsf{fnet}(\mathcal{F}) \text{ and, } D[\_] \text{ is a dummy adversary for } \mathcal{F}\}.$$

We now show that $\mathcal{R}$ is a labelled simulation. If $A \preceq_\ell B$ we trivially conclude. Suppose that $B = D[A]$ and $D[\_]$ is a dummy adversary for some functionality $\mathcal{F}$ such that $\mathsf{fnet}(A) \subseteq \mathsf{fnet}(\mathcal{F})$. We have that $D[\_] = \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(D_2 \mid \_))$ where D_1 and D_2 are closed plain processes as described in Definition 7.

We note that by the type system, for any label α we have that $bn(\alpha) \cap \mathsf{NET} = \emptyset$. Hence, if $A(\rightarrow^* \xrightarrow{\alpha} \rightarrow^*)^* A'$ then $\mathsf{fnet}(A') \subseteq \mathsf{fnet}(A)$. We now show the 3 points of the definition of a labelled simulation.

1. By construction of $D[\_]$, we have that $\phi(A) \equiv \phi(D[A])$, thus $\phi(A) \approx_s \phi(D[A])$.
2. Suppose that $A \rightarrow A'$. As $\rightarrow$ is closed under application of evaluation contexts, we have that $D[A] \rightarrow D[A']$. Moreover, $\mathsf{fnet}(A') \subseteq \mathsf{fnet}(A)$. We conclude that $A' \;\mathcal{R}\; D[A']$.
3. Suppose that $A \xrightarrow{\alpha} A'$ with $fv(\alpha) \subseteq \mathrm{dom}(A)$ and $bn(\alpha) \cap fn(D[A]) = \emptyset$. We have to consider different cases.
   - Names in $\widetilde{net}$ do not occur in α. In this case $D[A] \xrightarrow{\alpha} D[A']$. Moreover, $\mathsf{fnet}(A') \subseteq \mathsf{fnet}(A)$. We conclude that $A' \;\mathcal{R}\; D[A']$.
   - $\alpha = \mathsf{in}(net_k, M)$. We have that

$$\begin{array}{rl} D[A] \;\equiv & \nu\widetilde{sim}.(\mathsf{in}(net_k, x).\mathsf{out}(sim^i_k, x) \mid D_1 \mid \nu\widetilde{net}.(D_2 \mid A)) \\ \xrightarrow{in(net_k, M)} & \nu\widetilde{sim}.(\mathsf{out}(sim^i_k, M) \mid D_1 \mid \nu\widetilde{net}.(D_2 \mid A)) \\ \equiv & \nu\widetilde{sim}.(\mathsf{out}(sim^i_k, M) \mid D_1 \mid \nu\widetilde{net}.(\mathsf{in}(sim^i_k, x).\mathsf{out}(net_k, x) \mid D_2 \mid A)) \\ \rightarrow & \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(\mathsf{out}(net_k, M) \mid D_2 \mid A)) \\ \rightarrow & \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(D_2 \mid A')) \\ \equiv & D[A'] \end{array}$$

     Moreover, $\mathsf{fnet}(A') \subseteq \mathsf{fnet}(A)$. We conclude that $A' \;\mathcal{R}\; D[A']$.
   - $\alpha = (\nu u.)\mathsf{out}(net_k, u)$. We have that

$$\begin{array}{rl} D[A] \;\equiv & \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(\mathsf{in}(net_k, x).\mathsf{out}(sim^o_k, x) \mid D_2 \mid A)) \\ \rightarrow & \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(\nu u.)(\mathsf{out}(sim^o_k, u) \mid D_2 \mid A')) \\ \equiv & \nu\widetilde{sim}.(\mathsf{in}(sim^o_k, x).\mathsf{out}(net_k, x) \mid D_1 \mid \nu\widetilde{net}.(\nu u.)(\mathsf{out}(sim^o_k, u) \mid D_2 \mid A')) \\ \rightarrow & \nu\widetilde{sim}.(\nu u.)(\mathsf{out}(net_k, u) \mid D_1 \mid \nu\widetilde{net}.(D_2 \mid A')) \\ \xrightarrow{(\nu u.)out(net_k, u)} & \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(D_2 \mid A')) \\ \equiv & D[A'] \end{array}$$

     Moreover, $\mathsf{fnet}(A') \subseteq \mathsf{fnet}(A)$. We conclude that $A' \;\mathcal{R}\; D[A']$.

Lemma 4. *The relation $\leq^{\mathsf{SS}}$ is a preorder, that is the following hold: (i)* reflexivity: *$\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_1$; (ii)* transitivity: *$\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$ and $\mathcal{F}_2 \leq^{\mathsf{SS}} \mathcal{F}_3 \;\Rightarrow\; \mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_3$.*

Proof. Reflexivity holds thanks to Lemma 3. Now, it remains to establish transitivity. As $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$ and $\mathcal{F}_2 \leq^{\mathsf{SS}} \mathcal{F}_3$, we have that there exist an adversary $\mathcal{S}^1$ for $\mathcal{F}_2$ and an adversary $\mathcal{S}^2$ for $\mathcal{F}_3$ such that:

- $\mathcal{F}_1 \preceq \mathcal{S}^1[\mathcal{F}_2]$ and $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{S}^1[\mathcal{F}_2])$;
- $\mathcal{F}_2 \preceq \mathcal{S}^2[\mathcal{F}_3]$ and $\mathsf{fnet}(\mathcal{F}_2) = \mathsf{fnet}(\mathcal{S}^2[\mathcal{F}_3])$.

As $\preceq$ is closed under application of evaluation contexts (Proposition 1) we also have that $\mathcal{S}^1[\mathcal{F}_2] \preceq \mathcal{S}^1[\mathcal{S}^2[\mathcal{F}_3]]$. By transitivity of $\preceq$ we have that $\mathcal{F}_1 \preceq \mathcal{S}^1[\mathcal{S}^2[\mathcal{F}_3]]$.

As $\mathsf{fnet}(\mathcal{F}_2) = \mathsf{fnet}(S^2[\mathcal{F}_3])$ and $\mathcal{S}^1$ is an adversary for $\mathcal{F}_2$, we deduce that $\mathcal{S}_1$ is also an adversary for $S^2[\mathcal{F}_3]$ and thus, thanks to Lemma 2, we deduce that $\mathcal{S}^1[\mathcal{S}^2[\_]]$ is an adversary for $\mathcal{F}_3$. In order to conclude, it remains to show that $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{S}^1[\mathcal{S}^2[\mathcal{F}_3]])$.

As $\mathsf{fnet}(\mathcal{F}_2) = \mathsf{fnet}(\mathcal{S}^2[\mathcal{F}_3])$, we deduce that $\mathsf{fnet}(\mathcal{S}^1[\mathcal{F}_2]) = \mathsf{fnet}(\mathcal{S}^1[S^2[\mathcal{F}_3]])$ and we conclude thanks to the fact that $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{S}^1[\mathcal{F}_2])$. □

Lemma 8. *Let c be a channel of type* NET *and A be an extended process: $\nu c.A \preceq A$.*

Proof. Actually, we prove a stronger statement. We show that:

$$A \preceq_\ell B \text{ implies } \nu c.A \preceq_\ell B \quad \text{for any } c \in \mathsf{NET}.$$

Let $\mathcal{R} = \preceq_\ell \cup \{(A, B) \mid A \equiv \nu c.\bar{A},\ \bar{A} \preceq_\ell B\}$. We show that $\mathcal{R}$ is a labelled simulation. If $A \preceq_\ell B$ then we trivially conclude. Suppose $A\ \mathcal{R}\ B$ and $A \equiv \nu c.\bar{A}$ with $\bar{A} \preceq_\ell B$. We need to show the 3 points of the definition of labelled simulation.

1. As $c \in \mathsf{NET}$, we have that $\phi(\nu c.\bar{A}) \equiv \phi(\bar{A})$. As $\bar{A} \preceq_\ell B$ we have that $\bar{A} \approx_s B$ and hence we conclude that $A \approx_s B$.
2. Suppose $A \to A'$. Hence $\nu c.\bar{A} \to A'$. By inspection of the reduction rules we have that $A' \equiv \nu c.\bar{A}'$ and $\bar{A} \to \bar{A}'$ for some closed extended process $\bar{A}'$. As $\bar{A} \preceq_\ell B$ we have that there exists B' such that $B \to^* B'$ and $\bar{A}' \preceq_\ell B'$. Hence $A'\ \mathcal{R}\ B'$.
3. Suppose $A \xrightarrow{\alpha} A'$. Hence, $\nu c.\bar{A} \xrightarrow{\alpha} A'$. By the type system we have that $\alpha \neq \nu d.out(a, d)$ for any $d \in \mathsf{NET}$. By inspection of the labelled rules we have that $A' \equiv \nu c.\bar{A}'$ and $\bar{A} \xrightarrow{\alpha} \bar{A}'$ for some closed extended process $\bar{A}'$. As $\bar{A} \preceq_\ell B$ we have that there exists B' such that $B \to^* \xrightarrow{\alpha} \to^* B'$ and $\bar{A}' \preceq_\ell B'$. Hence $A'\ \mathcal{R}\ B'$.

To prove Lemma 8 we observe that $A \preceq_\ell A$. From the above statement we have that $\nu c.A \preceq_\ell A$ and hence $\nu c.A \preceq A$. □

Note that Lemma 8 relies on the type system and the fact that channels of type NET only appear in "channel position". In particular this avoids a counterexample where $A = \mathsf{out}(a, c)$. In such a case, we have that $A \xrightarrow{out(a,c)} 0$, whereas $\nu c.A$ can only moves with a label of the form $\nu d.\mathsf{out}(a, d)$.

Proposition 2. *Let $\mathcal{F}_1$, $\mathcal{F}_2$ be functionalities and C_{io} be an IO context.*

$$\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \implies C_{io}[\mathcal{F}_1] \leq^{\mathsf{SS}} C_{io}[\mathcal{F}_2].$$

Proof. As $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$ we have that there exists a simulator S for $\mathcal{F}_2$ such that $\mathcal{F}_1 \preceq S[\mathcal{F}_2]$ and $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(S[\mathcal{F}_2])$. As $\preceq$ is closed under application of evaluation contexts we also have that $C_{io}[\mathcal{F}_1] \preceq C_{io}[S[\mathcal{F}_2]]$. By Definition 5 and Definition **??**, we have that C_{io} and S are of the form:

- $C_{io} = \nu\widetilde{io}_1.(C_1 \mid \nu\widetilde{io}_2.(C_2 \mid \ldots \mid \nu\widetilde{io}_\ell.(C_\ell \mid \_)\ldots))$ with $\bigcup_{1\leq i\leq \ell} \widetilde{io}_i \subseteq \mathsf{IO}$ and where each C_i ($1 \leq i \leq \ell$) is a closed plain process.
- $S = \nu\widetilde{net}_1.(S_1 \mid \nu\widetilde{net}_2.(S_2 \mid \ldots \mid \nu\widetilde{net}_k.(S_k \mid \_)\ldots))$, $\mathsf{fnet}(\mathcal{F}_2) \subseteq \bigcup_{1\leq j\leq k} \widetilde{net}_j \subseteq \mathsf{NET}$, $fn(S[\_]) \cap \mathsf{IO} = \emptyset$ and where each S_j ($1 \leq j \leq k$) is a closed plain process.

Let $D[\_] = \nu\widetilde{sim}.(D_1 \mid \nu\widetilde{net}.(D_2 \mid \_))$ be the dummy adversary for $C_{io}[S[\mathcal{F}_2]]$. Thanks to Lemma 3 we have that $C_{io}[S[\mathcal{F}_2]] \preceq D[C_{io}[S[\mathcal{F}_2]]]$. Now, thanks to Lemma 8, we have that

$$D[C_{io}[S[\mathcal{F}_2]]] \preceq D[C_{io}[S'[\mathcal{F}_2]]]$$

where:

- $S' = \nu\widetilde{net}'_1.(S_1 \mid \nu\widetilde{net}'_2.(S_2 \mid \ldots \mid \nu\widetilde{net'}_k.(S_k \mid \_)\ldots))$, and
- $\widetilde{net}'_i = \widetilde{net}_i \setminus \widetilde{net}$.

Since $\widetilde{net}'_i \cap \mathsf{fnet}(C_{io}[\_]) = \emptyset$ ($1 \leq i \leq k$) and $\mathsf{fnet}(S'[\_]) \cap \mathsf{IO} = \emptyset$, we have that $C_{io}[S'[\mathcal{F}_2]] \equiv S'[C_{io}[\mathcal{F}_2]]$ and thus $D[C_{io}[S'[\mathcal{F}_2]]] \equiv D[S'[C_{io}[\mathcal{F}_2]]]$. In order to conclude, it remains to show that $D[S'[\_]]$ is a simulator for $C_{io}[\mathcal{F}_2]$ such that $\mathsf{fnet}(C_{io}[\mathcal{F}_1]) = \mathsf{fnet}(D[S'[C_{io}[\mathcal{F}_2]]])$.

First note that $D[S'[\_]]$ is of the right form and $\mathsf{fnet}(D[S'[\_]]) \cap \mathsf{IO} = \emptyset$. Moreover, since D is a dummy adversary for $C_{io}[\mathcal{F}_2]$, we have $\widetilde{net} = \mathsf{fnet}(C_{io}[S[\mathcal{F}_2]])$ and thus we have that:

$$\begin{array}{rcl}
\mathsf{fnet}(C_{io}[\mathcal{F}_2]) & = & \mathsf{fnet}(C_{io}[_]) \cup \mathsf{fnet}(\mathcal{F}_2) \\
& \subseteq & \mathsf{fnet}(C_{io}[_]) \cup \bigcup_{1\leq j\leq k} \widetilde{net}_j \\
& \subseteq & \mathsf{fnet}(C_{io}[S[\mathcal{F}_2]]) \cup \bigcup_{1\leq j\leq k} \widetilde{net}_j \\
& = & \widetilde{net} \cup \bigcup_{1\leq j\leq k} \widetilde{net}_j \\
& = & \widetilde{net} \cup \bigcup_{1\leq j\leq k} \widetilde{net}'_j \\
& \subseteq & \widetilde{sim} \cup \widetilde{net} \cup \bigcup_{1\leq j\leq k} \widetilde{net}'_j
\end{array}$$

Thus, $D[S'[\_]]$ is a simulator for $C_{io}[\mathcal{F}_2]$. Moreover, we have that $\mathsf{fnet}(C_{io}[\mathcal{F}_1]) = \mathsf{fnet}(D[S'[C_{io}[\mathcal{F}_2]]])$. Indeed, we have that:

- $\mathsf{fnet}(C_{io}[\mathcal{F}_1]) = \mathsf{fnet}(C_{io}[S[\mathcal{F}_2]]) = \widetilde{net}$, and
- $\mathsf{fnet}(D[S'[C_{io}[\mathcal{F}_2]]]) = \mathsf{fnet}(D[C_{io}[S'[\mathcal{F}_2]]]) = \mathsf{fnet}(D[C_{io}[S[\mathcal{F}_2]]]) = \mathsf{fnet}(C_{io}[S[\mathcal{F}_2]]) = \widetilde{net}$.

This allows us to conclude. □

Proposition 3. *Let $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ be three functionalities. We have that:*
(i) $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow \mathcal{F}_1 \mid \mathcal{F}_3 \leq^{\mathsf{SS}} \mathcal{F}_2 \mid \mathcal{F}_3$; and (ii) $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow !\mathcal{F}_1 \leq^{\mathsf{SS}} !\mathcal{F}_2$.

Proof. We show that $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow !\mathcal{F}_1 \leq^{\mathsf{SS}} !\mathcal{F}_2$. We have that there exists an adversary $\mathcal{S}$ for $\mathcal{F}_1$ such that $\mathcal{F}_1 \preceq_\ell \mathcal{S}[\mathcal{F}_2]$. By Lemma 1 we have that $!\mathcal{F}_1 \preceq_\ell !\mathcal{S}[\mathcal{F}_2]$. Moreover, we have

$$\begin{array}{lll} !\mathcal{S}[\mathcal{F}_2] = !(\widetilde{\nu net}_1.(S_1 \mid \widetilde{\nu net}_2.(S_2 \mid \ldots \mid \widetilde{\nu net}_k.(S_k \mid \mathcal{F}_2)\ldots))) & & \\ \quad \preceq_\ell !(S_1 \mid S_2 \mid \ldots \mid S_k \mid \mathcal{F}_2) & \text{by Lemmas 8 and 1} \\ \quad \preceq_\ell !S_1 \mid !S_2 \mid \ldots \mid !S_k \mid !\mathcal{F}_2) & \text{by Lemma 1} \\ \quad \preceq_\ell D[!S_1 \mid !S_2 \mid \ldots \mid !S_k \mid !\mathcal{F}_2)] & \text{by Lemma 3} \end{array}$$

where $D[\_]$ is the dummy adversary for $!S_1 \mid !S_2 \mid \ldots \mid !S_k \mid !\mathcal{F}_2$. Defining $\mathcal{S}' = D[!S_1 \mid !S_2 \mid \ldots \mid !S_k \mid \_)]$ we obtain that $!\mathcal{F}_1 \preceq_\ell !\mathcal{S}[\mathcal{F}_2] \preceq_\ell \mathcal{S}'[!\mathcal{F}_2]$ and conclude that $!\mathcal{F}_1 \leq^{\mathsf{SS}} !\mathcal{F}_2$.

Theorem 1. *We have that* $\leq^{\mathsf{SS}} = \leq^{\mathsf{BB}} = \leq^{\mathsf{UC}} = \leq^{\mathsf{UCDA}}$.

Proof. We prove the following inclusions:

- $\leq^{\mathsf{SS}} \subseteq \leq^{\mathsf{BB}}$.
  Suppose that $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$. Hence, there exists an adversary S_{ss} for $\mathcal{F}_2$ such that $\mathcal{F}_1 \preceq S_{\mathsf{ss}}[\mathcal{F}_2]$ and $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{S}_{\mathsf{ss}}[\mathcal{F}_2])$. We have to show that there exists an adversary $\mathcal{S}_{\mathsf{bb}}$ for $\mathcal{F}_2$ such that for all adversary $\mathcal{A}$ for $\mathcal{F}_1$ we have that $\mathsf{fnet}(S_{\mathsf{bb}}[\mathcal{F}_2]) = \mathsf{fnet}(\mathcal{F}_1)$ and $A[\mathcal{F}_1] \preceq \mathcal{A}[\mathcal{S}_{\mathsf{bb}}[\mathcal{F}_2]]$.
  Let $\mathcal{S}_{\mathsf{bb}} \stackrel{\mathsf{def}}{=} \mathcal{S}_{\mathsf{ss}}$. Hence, we have that $\mathcal{F}_1 \preceq S_{\mathsf{bb}}[\mathcal{F}_2]$ and $\mathsf{fnet}(\mathcal{S}_{\mathsf{bb}}[\mathcal{F}_2]) = \mathsf{fnet}(\mathcal{F}_1)$. As $\preceq$ is closed under application of evaluation contexts, for any adversary $\mathcal{A}$ for $\mathcal{F}_1$ we obtain that $\mathcal{A}[\mathcal{F}_1] \preceq \mathcal{A}[\mathcal{S}_{\mathsf{bb}}[\mathcal{F}_2]]$.
- $\leq^{\mathsf{BB}} \subseteq \leq^{\mathsf{UC}}$.
  Suppose that $\mathcal{F}_1 \leq^{\mathsf{BB}} \mathcal{F}_2$. In such a case there exists an adversary $\mathcal{S}_{\mathsf{bb}}$ for $\mathcal{F}_2$ such that $\mathsf{fnet}(\mathcal{S}_{\mathsf{bb}}[\mathcal{F}_2]) = \mathsf{fnet}(\mathcal{F}_1)$ and for all adversary $\mathcal{A}$ for $\mathcal{F}_1$ we have that $\mathcal{A}[\mathcal{F}_1] \preceq \mathcal{A}[\mathcal{S}_{\mathsf{bb}}[\mathcal{F}_2]]$. We have to show that for all adversary $\mathcal{A}$ for $\mathcal{F}_2$, there exists an adversary $\mathcal{S}_{\mathsf{uc}}$ for $\mathcal{F}_1$ such that $\mathsf{fnet}(\mathcal{A}[\mathcal{F}_1]) = \mathsf{fnet}(S_{\mathsf{uc}}[\mathcal{F}_2])$ and $A[\mathcal{F}_1] \preceq \mathcal{S}_{\mathsf{uc}}[\mathcal{F}_2]$. Let $\mathcal{A}$ be an adversary $\mathcal{F}_2$ and $\mathcal{S}_{\mathsf{uc}}[\_] \stackrel{\mathsf{def}}{=} \mathcal{A}[\mathcal{S}_{\mathsf{bb}}[\_]]$. As $\mathsf{fnet}(\mathcal{F}_1) = \mathsf{fnet}(\mathcal{S}_{\mathsf{bb}}[\mathcal{F}_2])$, we have also $\mathsf{fnet}(\mathcal{A}[\mathcal{F}_1]) = \mathsf{fnet}(\mathcal{A}[\mathcal{S}_{\mathsf{bb}}[\mathcal{F}_2]]) = \mathsf{fnet}(\mathcal{S}_{\mathsf{uc}}[\mathcal{F}_2])$. In order to conclude, it remains to show that $\mathcal{S}_{\mathsf{uc}}$ is an adversary for $\mathcal{F}_2$. This is indeed the case since $\mathcal{S}_{\mathsf{uc}}[\_] = \mathcal{A}[\mathcal{S}_{\mathsf{bb}}[\_]]$ and $\mathcal{S}_{\mathsf{bb}}$ is an adversary for $\mathcal{F}_2$ and $\mathcal{A}$ is also an adversary. This allows us to conclude.
- $\leq^{\mathsf{UC}} \subseteq \leq^{\mathsf{UCDA}}$.
  Suppose that $\mathcal{F}_1 \leq^{\mathsf{UC}} \mathcal{F}_2$. Hence, for all adversary $\mathcal{A}$ for $\mathcal{F}_1$ there exists an adversary $\mathcal{S}_{\mathsf{uc}}$ for $\mathcal{F}_2$ such that $A[\mathcal{F}_1] \preceq S[\mathcal{F}_2]$ and $\mathsf{fnet}(A[\mathcal{F}_1]) = \mathsf{fnet}(S_{\mathsf{uc}}[\mathcal{F}_2])$. We have to show that there exists an adversary $\mathcal{S}$ for $\mathcal{F}_2$ such that $F[\mathcal{F}_1] \preceq S[\mathcal{F}_2]$ and $\mathsf{fnet}(S[\mathcal{F}_2]) = \mathsf{fnet}(D[\mathcal{F}_1])$ where D is the dummy adversary for $\mathcal{F}_1$. Taking D for $\mathcal{A}$ we easily conclude by applying our hypothesis.
- $\leq^{\mathsf{UCDA}} \subseteq \leq^{\mathsf{SS}}$.
  Suppose that $\mathcal{F}_1 \leq^{\mathsf{UCDA}} \mathcal{F}_2$. Hence, there exists an adversary $\mathcal{S}$ for $\mathcal{F}_2$ such that $D[\mathcal{F}_1] \preceq S[\mathcal{F}_2]$ and $\mathsf{fnet}(S[\mathcal{F}_2]) = \mathsf{fnet}(D[\mathcal{F}_1])$ where D is the dummy adversary for $\mathcal{F}_1$. We have that $\mathsf{fnet}(D[\mathcal{F}_1]) = \mathsf{fnet}(\mathcal{F}_1)$. By Lemma 3, we have that $\mathcal{F}_1 \preceq D[\mathcal{F}_1]$. We conclude by transitivity of $\preceq$. □

C Proof of Section 4

Lemma 5. *Let n be a name and c be a channel name such that $c \notin fn(P) \cup fn(Q)$.*
$\nu c.\,![\nu n.(\mathsf{out}(c,n) \mid P) \mid \mathsf{in}(c,x).Q] \preceq_\ell !\,\nu c.[\nu n.(\mathsf{out}(c,n) \mid P) \mid \mathsf{in}(c,x).Q]$.

Proof. Consider the processes P_1, P_2 and $\mathcal{P}$ defined as follows:

- $P_1 = \nu c.\,![\nu n.(\mathsf{out}(c,n) \mid P) \mid \mathsf{in}(c,x).Q]$, and
- $P_2 = !\nu c.[\nu n.(\mathsf{out}(c,n) \mid P) \mid \mathsf{in}(c,x).Q]$, and
- $\mathcal{P} = \nu n.(\mathsf{out}(c,n) \mid P) \mid \mathsf{in}(c,x).Q$.

Let k be an integer. We consider the context $C_k^A[\_]$ and $C_k^B[\_]$ defined as follows:

- $C_k^A[\_] = \nu n_1, \ldots, n_k.\,[(\mathsf{out}(c,n_1) \mid \mathsf{in}(c,x).Q) \mid \cdots \mid (\mathsf{out}(c,n_k) \mid \mathsf{in}(c,x).Q) \mid \_]$.
- $C_k^B[\_] = \nu n_1, \ldots, n_k.\,[\nu c.(\mathsf{out}(c,n_1) \mid \mathsf{in}(c,x).Q) \mid \cdots \mid \nu c.(\mathsf{out}(c,n_k) \mid \mathsf{in}(c,x).Q) \mid \_]$.

By convention $C_k^A[\_] = C_k^B[\_] = \_$.

We define the following relation $\mathcal{R}$ on closed extended processes

$$\mathcal{R} = \preceq_\ell \cup \{(A,B) \mid A \equiv \nu c.(!\mathcal{P} \mid C_k^A[D]) \text{ and } B \equiv (!\nu c.\mathcal{P}) \mid C_k^B[D] \text{ for some extended process } D \text{ and some } k\}$$

Note that $P_1 \;\mathcal{R}\; P_2$ and we can show that $\mathcal{R}$ is a labelled simulation by showing that the 3 points of the definition of labelled simulation hold. □