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]Université catholique de Louvain, B-1348 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 the "trusted party paradigm" used in computational cryptography models. 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 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 [17], and provided support for protocol design [9]. These techniques also resulted in the creation of powerful automated analysis tools (e.g. [3]), and impacted on several protocol standards used every day, e.g., [8].

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 [7]. Moreover, protocols are often expected to provide more sophisticated security guarantees, which may be difficult to formalize.

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 trusted parties, or ideal functionalities, following the general outline of simulation-based security [5, 12, 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

*This work has been supported by the ANR-07-SESU-002 project. O. Pereira is a Research Associate of the Belgian Fund for Scientific Research FRS-FNRS. Part of this work was done when he was visiting ENS Cachan.

Foundations of Software Technology and Theoretical Computer Science (Kanpur) 2009.
Editors: Ravi Kannan and K. Narayan Kumar; pp 169–180
Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.
Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2009.2316

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 [5]. 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, 5, 6, 14, 18], 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 schedulers 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 fundamental level, one has to adopt a notion of indistinguishability of processes. While the symmetric notions of computational indistinguishability and observational equivalence are commonly used in the cryptographic and symbolic worlds, the symmetry of such relations appeared to be too restrictive for our purpose. For instance, a symmetric equivalence relation makes the addition of an adversary that simply acts as a relay visible. The resulting 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. An association table (plaintext/ciphertext) is then necessary to perform decryption. In our symbolic setting we avoid such a table by using two layers of encryption and a secure key.

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, 5, 6, 16], the joint state composition theorem brings message tagging constraints that, interestingly, are consistent with those obtained by using a completely different symbolic approach (e.g. [13]).

Because of lack of space the proofs are omitted, but can be found in [10].

2 The applied pi calculus

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, and dec for decryption. Terms are defined as names, variables, and function symbols applied to other terms. Terms and function symbols are sorted. While the details of the sort systems are not essential it is important to distinguish sorts of base types and sorts of channel type. Function symbols can only be applied and return terms of base type. By the means of an equational theory E we describe the equations which hold on terms. We denote $=_{\mathsf{E}}$ the equivalence relation induced by E.

Example 1 *In the equational theory* $\{\mathsf{dec}(\mathsf{enc}(x,k),k) = x,\ \mathsf{test}(\mathsf{enc}(x,y),y) = \mathsf{ok}\}$*, 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 (P, Q, R) are built up in a similar way to processes in the pi calculus, except that messages can contain terms (rather than just names). Extended processes add *active substitutions* and restriction on variables. Below, M is a term, n is a name, and x a variable.

$$A, B, C := P \mid A \mid B \mid \nu n.A \mid \nu x.A \mid \{^{M}/_{x}\}$$

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 that are not under a restriction). An *evaluation context* $C[\_]$ is an extended process with a hole instead of an extended process. A context $C[\_]$ *closes* A when $C[A]$ is closed.

2.2 Semantics

The operational semantics of processes in the applied pi calculus is defined by structural rules defining two relations: *structural equivalence*, denoted $\equiv$, and *internal reduction*, denoted $\to$. Structural equivalence takes into account some basic structural rules, e.g. associativity and commutativity of the parallel operator. Internal reduction $\to$ is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that:

$$\begin{array}{llcll}
\text{COMM} & \mathsf{out}(a,M).P \mid \mathsf{in}(a,x).Q & \rightarrow & P \mid Q\{^{M}/_{x}\} & \\
\text{THEN} & \text{if } M = N \text{ then } P \text{ else } Q & \rightarrow & P & \text{where } M =_{\mathsf{E}} N \\
\text{ELSE} & \text{if } M = N \text{ then } P \text{ else } Q & \rightarrow & Q & \\
& \multicolumn{4}{l}{\text{for any terms } M \text{ and } N \text{ without variable such that } M \neq_{\mathsf{E}} N}
\end{array}$$

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 a variable of base type), or $\mathsf{out}(a, c)$ or $\nu c.\mathsf{out}(a, c)$ (c is a channel name).

$$\text{IN} \qquad \mathsf{in}(a,x).P \xrightarrow{in(a,M)} P\{^{M}/_{x}\}$$

$$\text{OUT-CH} \qquad \mathsf{out}(a,c).P \xrightarrow{out(a,c)} P$$

$$\text{OPEN-CH} \qquad \frac{A \xrightarrow{out(a,c)} A' \qquad c \neq a}{\nu c.A \xrightarrow{\nu c.out(a,c)} A'}$$

$$\text{OUT-T} \qquad \mathsf{out}(a,M).P \xrightarrow{\nu x.out(a,x)} P \mid \{^{M}/_{x}\} \qquad x \notin fv(P) \cup fv(M)$$

$$\text{SCOPE} \qquad \frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

$$\text{PAR} \qquad \frac{A \xrightarrow{\alpha} A' \qquad \begin{array}{c} bn(\alpha) \cap fn(B) = \emptyset \\ bv(\alpha) \cap fv(B) = \emptyset \end{array}}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\text{STRUCT} \qquad \frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$$

Our rules differ slightly from those described in [1]. It is proved in [11] that labelled bisimulation (see Section 2.3) in our system coincides with labelled bisimulation in [1].

Example 2 *Consider the following process P:*
$\nu k.\ (\mathsf{in}(io_1, x).\mathsf{out}(net, \mathsf{enc}(x, k)) \mid \mathsf{in}(net, y).\ \textit{if}\ \mathsf{test}(y, k) = \mathsf{ok}\ \textit{then}\ \mathsf{out}(io_2, \mathsf{dec}(y, k))\ \textit{else}\ 0)$.

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 one 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. We have that:

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

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

2.3 Indistinguishability relations

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.*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 3 *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 that $(\mathsf{test}(x,k) =_{\mathsf{E}} \mathsf{ok})\varphi_0$ but not $(\mathsf{test}(x,k) =_{\mathsf{E}} \mathsf{ok})\varphi_1$, thus $\varphi_0 \not\approx_s \varphi_1$. However, we have that $\nu k.\varphi_0 \approx_s \varphi_1$.*

Now, 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 *($\preceq$) (resp.* equivalence *($\approx$)) is the largest (resp. largest symmetric) relation on extended processes with same domain s.t. $A \mathcal{R} B$ 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. *A relation $\mathcal{R}$ on closed extended processes is a* simulation *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 simulations we say that $\mathcal{R}$ is a bisimulation. 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 similarity is a precongruence.

PROPOSITION 4. *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. We have also the following lemma:

LEMMA 5. *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 6. *A* functionality *$\mathcal{F}$ is a closed plain process. An* adversary *for $\mathcal{F}$ is an evaluation context $\mathcal{A}[\_]$ of the form: $\nu\widetilde{net}_1.(A_1 \mid \nu\widetilde{net}_2.(A_2 \mid \ldots \mid \nu\widetilde{net}_k.(A_k \mid \_)\ldots))$ where each A_i ($1 \leq i \leq k$) is a closed plain process, $\mathsf{fnet}(\mathcal{F}) \subseteq \bigcup_{1 \leq i \leq k} \widetilde{net}_i \subseteq \mathsf{NET}$, and $fn(\mathcal{A}[\_]) \cap \mathsf{IO} = \emptyset$.*

One may note that the nested form of the adversary process allows to express any arbitrary context while expliciting the restricted names whose scope ranges on the hole. We also note that if $\mathcal{A}[\_]$ is an adversary for $\mathcal{F}$ then $\mathsf{fnet}(\mathcal{A}[\_]) = \mathsf{fnet}(\mathcal{A}[\mathcal{F}])$.

LEMMA 7. *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. If $\mathcal{A}_2[\_]$ is an adversary for $\mathcal{A}_1[\mathcal{F}]$, then $\mathcal{A}_2[\mathcal{A}_1[\_]]$ is an adversary for $\mathcal{F}$.*

While adversaries control the communication of functionalities on NET channels, IO *contexts* model the environment of functionalities, providing inputs and collecting outputs.

DEFINITION 8. *An* IO context *is an evaluation context $C_{io}[\_]$ of the form $\nu\widetilde{io}_1.(C_1 \mid \nu\widetilde{io}_2.(C_2 \mid \ldots \mid \nu\widetilde{io}_k.(C_k \mid \_) \ldots))$ where each C_i ($1 \leq i \leq k$) is a closed plain process, and $\bigcup_{1 \leq i \leq k} \widetilde{io}_i \subseteq \mathsf{IO}$*

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

3.2 Strong simulatability

The notion of strong simulatability [15], which is probably the simplest secure emulation notion used in simulation-based security, can be formulated in our setting.

DEFINITION 9.*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 $\mathcal{F}_1 \preceq \mathcal{S}[\mathcal{F}_2]$.*

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. One may also note that $0 \leq^{\mathsf{SS}} \mathcal{F}$ for any functionality $\mathcal{F}$. This seems natural in a simulation based framework which only aims at preserving security. Non-triviality conditions may be imposed independently [5].

Example 4 *Let $\mathcal{F}_{\mathsf{cc}} = \mathsf{in}(io_1, x_s).\mathsf{out}(net_{\mathsf{cc}}, \mathsf{ok}).\mathsf{in}(net_{\mathsf{cc}}, x).$ if $x = \mathsf{ok}$ then $\mathsf{out}(io_2, x_s)$. 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 described 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 $D[\_]$ which is called a *dummy adversary*: intuitively, it acts as a relay which forwards all messages. The formal definition is technical because $D[\_]$ needs to both restrict all names in $\mathsf{fnet}(\mathcal{F})$ and ensure that $\mathsf{fnet}(\mathcal{F}) = \mathsf{fnet}(D[\mathcal{F}])$. It therefore relies on two internal channels $sim_j^{i/o}$ for inputs, resp. outputs, for each channel in $\mathsf{fnet}(\mathcal{F})$.

DEFINITION 10. *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 $\widetilde{net} = \mathsf{fnet}(\mathcal{F}) = \{net_1, \ldots, net_n\}$; $\widetilde{sim} = \{sim_1^i, \ldots, sim_n^i, sim_1^o, \ldots, sim_n^o\} \subseteq \mathsf{NET}$; and*

- $D_1 = !\mathsf{in}(net_1, x).\mathsf{out}(sim_1^i, x) \mid \ldots \mid !\mathsf{in}(net_n, x).\mathsf{out}(sim_n^i, x) \mid$
  $!\mathsf{in}(sim_1^o, x).\mathsf{out}(net_1, x) \mid \ldots \mid !\mathsf{in}(sim_n^o, x).\mathsf{out}(net_n, x)$;
- $D_2 = !\mathsf{in}(sim_1^i, x).\mathsf{out}(net_1, x) \mid \ldots \mid !\mathsf{in}(sim_n^i, x).\mathsf{out}(net_n, x) \mid$
  $!\mathsf{in}(net_1, x).\mathsf{out}(sim_1^o, x) \mid \ldots \mid !\mathsf{in}(net_n, x).\mathsf{out}(sim_n^o, x)$.

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

LEMMA 11. *Let $\mathcal{F}$ be a functionality and $D[\_]$ be 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 $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$: as $\mathcal{F}_2 \preceq D[\mathcal{F}_2]$ we have by transitivity that $\mathcal{F}_1 \preceq D[\mathcal{F}_2]$. We use these observations to show that $\leq^{\mathsf{SS}}$ is a preorder (Lemma 12) , which is closed under application of IO contexts (Proposition 13) and parallel composition (Proposition 14).

LEMMA 12. *(i)* $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_1$; *(ii)* $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2$ *and* $\mathcal{F}_2 \leq^{\mathsf{SS}} \mathcal{F}_3 \Rightarrow \mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_3$.

PROPOSITION 13. *Let* $\mathcal{F}_1, \mathcal{F}_2$ *be two functionalities and* C_{io} *be an IO context.*

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

PROPOSITION 14. *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)* $\mathcal{F}_1 \leq^{\mathsf{SS}} \mathcal{F}_2 \Rightarrow !\mathcal{F}_1 \leq^{\mathsf{SS}} !\mathcal{F}_2$.

While, (i) is a direct consequence of Proposition 13 (notice that $\_ \mid \mathcal{F}_3$ is an IO-context) the proof of (ii) is more involved and given in [10].

3.3 Other notions of simulation based security

Several other notions of simulation based security appear in the literature. We present them, and show that they all coincide in our setting. This coincidence is regarded as highly desirable [15, 14], while it does not hold in most simulation-based security frameworks [5, 4].

DEFINITION 15. *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, $\mathcal{F}_1 \leq^{\mathsf{BB}} \mathcal{F}_2$, *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, $\mathcal{F}_1 \leq^{\mathsf{UC}} \mathcal{F}_2$, *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* $\mathcal{F}_2$ *s.t.* $\mathsf{fnet}(\mathcal{A}[\mathcal{F}_1]) = \mathsf{fnet}(\mathcal{S}[\mathcal{F}_2])$.
- $\mathcal{F}_1$ *emulates* $\mathcal{F}_2$ *in the sense of* universally composable simulatability with dummy adversary, $\mathcal{F}_1 \leq^{\mathsf{UCDA}} \mathcal{F}_2$, *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 16. *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 denoted $\leq^{\mathsf{SS}}_{\approx}, \leq^{\mathsf{BB}}_{\approx}, \leq^{\mathsf{UC}}_{\approx}$ 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, due to the extra non-determinism that the simulator introduces. While symbolic analysis techniques typically rely on bisimulation relations, this is however consistent with the definitions proposed in the task-PIOA framework [6], which also allows non-deterministic executions for simulation based security.

4 Applications

We illustrate our framework by showing the secure emulation of a mutual authentication functionality by the Needham-Shroeder-Lowe (NSL) protocol [17]. As the NSL protocol uses public key encryption we first introduce in Section 4.1 functionalities for asymmetric

$$
\begin{array}{lll}
\mathcal{P}_{\mathsf{pke}} & := & \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\nu sk.\mathsf{out}(io^1_{\mathsf{pke}}, \langle \textsc{key}, \mathsf{pk}(sk)\rangle). \\
 & & (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{P}_{\mathsf{enc}} \ \mid \ \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{P}_{\mathsf{enc}} \ \mid !\mathcal{P}_{\mathsf{dec}}) \\
\mathcal{P}_{\mathsf{enc}} & := & \mathsf{in}(io^i_{\mathsf{pke}}, \langle = \textsc{enc}, m\rangle). \\
 & & \nu r_2.\ \text{let } menc = \mathsf{aenc}(\langle \textsc{tag}_0, m\rangle, \mathsf{pk}(sk), r_2) \text{ in } \mathsf{out}(io^i_{\mathsf{pke}}, \langle \textsc{cipher}, menc\rangle) \\
\mathcal{P}_{\mathsf{dec}} & := & \mathsf{in}(io^1_{\mathsf{pke}}, \langle = \textsc{dec}, m\rangle). \\
 & & \text{let } \langle = \textsc{tag}_0, m_1\rangle = \mathsf{adec}(m, sk) \text{ in } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \textsc{plain}, m\rangle)
\end{array}
$$

Figure 1: Real encryption functionality

encryption. Then, we briefly 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.

4.1 Asymmetric encryption with joint state

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 by introducing an *ideal functionality* for encryption we are able to obtain a joint state composition result. Throughout this section we rely on the following equational theory allowing us to model randomized asymmetric encryption:

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

Real encryption. This functionality (decribed in Figure 1) receives a channel name io^1_{pke}, which will be used for all sensitive information exchanges. 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. Each time a decryption request is received on the channel io^1_{pke}, it tries to decrypt the ciphertext and checks whether the tag is $\textsc{tag}_0$. If so, it outputs the plaintext on the channel io^1_{pke}. Otherwise, it does nothing.

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, 5, 16], 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.

$$\begin{array}{lll}
\mathcal{F}_{\mathsf{pke}} & := & \mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{out}(net, \text{INIT}).\mathsf{in}(net, \langle = \text{ALGO}, sk, tag\rangle).\mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{KEY}, \mathsf{pk}(sk)\rangle). \\
 & & \nu ssk.\ (\text{let } io^i_{\mathsf{pke}} = io^1_{\mathsf{pke}} \text{ in } !\mathcal{F}_{\mathsf{enc}} \ \mid\ \text{let } io^i_{\mathsf{pke}} = io^2_{\mathsf{pke}} \text{ in } !\mathcal{F}_{\mathsf{enc}} \ \mid !\mathcal{F}_{\mathsf{dec}}) \\
\mathcal{F}_{\mathsf{enc}} & := & \mathsf{in}(io^i_{\mathsf{pke}}, \langle = \text{ENC}, m\rangle).\nu r_1.\nu r_2. \\
 & & \text{let } alea = \mathsf{aenc}(m, \mathsf{pk}(ssk), r_1) \text{ in let } menc = \mathsf{aenc}(\langle tag, alea\rangle, \mathsf{pk}(sk), r_2) \text{ in} \\
 & & \mathsf{out}(io^i_{\mathsf{pke}}, \langle \text{CIPHER}, menc\rangle) \\
\mathcal{F}_{\mathsf{dec}} & := & \mathsf{in}(io^1_{\mathsf{pke}}, \langle = \text{DEC}, m\rangle).\ \text{let } \langle = tag, m_1\rangle = \mathsf{adec}(m, sk) \text{ in} \\
 & & \text{if } \mathsf{testdec}(m_1, ssk) = \mathsf{ok} \text{ then } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, \mathsf{adec}(m_1, ssk)\rangle) \\
 & & \qquad\qquad\qquad\qquad\qquad \text{else } \mathsf{out}(io^1_{\mathsf{pke}}, \langle \text{PLAIN}, m_1\rangle)
\end{array}$$

Figure 2: Ideal encryption functionality

During the initialization, 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. Each time the process receives an encryption request, it computes the corresponding ciphertext and outputs the corresponding ciphertext. As explained above, the plaintext m is first encrypted using $\mathsf{pk}(ssk)$ before being tagged and encrypted with $\mathsf{pk}(sk)$. When the process receives a decryption request, 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.

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

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*, i.e. a functionality that realizes $!\mathcal{F}_{\mathsf{pke}}$ while reusing the same key material. We actually consider 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 joint state functionality $\mathcal{P}_{\mathsf{js}}[\underline{\mathcal{F}_{\mathsf{pke}}}]$ (see Figure 3) uses a single instance of $\underline{\mathcal{F}_{\mathsf{pke}}}$ for 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}.

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: $\mathcal{P}_{\mathsf{js}}[\mathcal{F}_{\mathsf{pke}}] \leq^{\mathsf{SS}} !\underline{\mathcal{F}_{\mathsf{pke}}}$.

$$\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, \textsc{init}\rangle). \\
 & & \mathsf{in}(cont, (= \textsc{key}, pk)).\mathsf{out}(io^1_{\mathsf{pke}}, \langle sid, \textsc{key}, pk\rangle). \\
 & & (\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 \\
 & & \;!\mathsf{in}(io_{\mathsf{pke}}, io^1_{\mathsf{pke}}).\mathsf{in}(io_1, sid).\mathsf{out}(io^1_{\mathsf{pke}}, \langle sid, \textsc{key}, pk\rangle). \\
 & & \;(\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, = \textsc{enc}, m\rangle).\ \mathsf{out}(cont, \langle sid, \textsc{enc}, m\rangle). \\
 & & \mathsf{in}(cont, \langle = \textsc{cipher}, c\rangle).\ \mathsf{out}(io^i_{\mathsf{pke}}, \langle sid, \textsc{cipher}, c\rangle) \\
\mathcal{P}^1_{\mathsf{js-dec}} & := & \mathsf{in}(io^1_{\mathsf{pke}}, \langle = sid, = \textsc{dec}, c\rangle).\ \mathsf{out}(cont, \langle sid, dec, c\rangle). \\
 & & \mathsf{in}(cont, \langle = \textsc{plain}, m\rangle).\ \mathsf{out}(io_{\mathsf{pke}}, \langle sid, \textsc{plain}, m\rangle) \\
\mathcal{P}^2_{\mathsf{js}} & := & \mathsf{in}(cont, \langle sid, = \textsc{init}\rangle).\nu io'_{\mathsf{pke}}.\mathsf{out}(io_{\mathsf{pke}}, io'_{\mathsf{pke}}). \\
 & & \mathsf{in}(io'_{\mathsf{pke}}, \langle = \textsc{key}, pk\rangle).\mathsf{out}(cont, \langle \textsc{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, = \textsc{enc}, m\rangle).\ \mathsf{out}(io'_{\mathsf{pke}}, \langle \textsc{enc}, \langle sid, m\rangle\rangle). \\
 & & \mathsf{in}(io'_{\mathsf{pke}}, \langle = \textsc{cipher}, c\rangle).\ \mathsf{out}(cont, \langle \textsc{cipher}, c\rangle) \\
\mathcal{P}^2_{\mathsf{js-dec}} & := & \mathsf{in}(cont, \langle = sid, = \textsc{dec}, c\rangle).\ \mathsf{out}(io'_{\mathsf{pke}}, \langle \textsc{dec}, c\rangle). \\
 & & \mathsf{in}(io'_{\mathsf{pke}}, \langle = \textsc{plain}, m\rangle).\ \mathsf{out}(cont, \langle \textsc{plain}, m\rangle)
\end{array}$$

Figure 3: Joint state IO-context

$$\begin{array}{lll}
\mathcal{A}_{\mathsf{js}} & := & \nu cs.(\mathcal{A}^1_{\mathsf{js}} \mid \nu net.(\mathcal{A}^2_{\mathsf{js}} \mid \_)) \\
\mathcal{A}^1_{\mathsf{js}} & := & \mathsf{in}(cs, \textsc{init}).\mathsf{out}(net, \textsc{init}).\mathsf{in}(net, \langle = \textsc{algo}, sk, tag\rangle).\mathsf{out}(cs, \langle \textsc{algo}, sk, tag\rangle) \\
\mathcal{A}^2_{\mathsf{js}} & := & \mathsf{in}(net, \langle sid, = \textsc{init}\rangle).\ \mathsf{out}(cs, \textsc{init}). \\
 & & \mathsf{in}(cs, \langle = \textsc{algo}, sk, tag\rangle).\ \mathsf{out}(net, \langle sid, \textsc{algo}, sk, \langle sid, tag\rangle\rangle). \\
 & & !\mathsf{in}(net, \langle sid', = \textsc{init}\rangle).\mathsf{out}(net, \langle sid', \textsc{algo}, sk, \langle sid', tag\rangle\rangle)
\end{array}$$

Figure 4: Joint state adversary

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.

Note that it is crucial to introduce an ideal encryption 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

Because of lack of space we only briefly sketch the mutual authentication functionality. The details of the functionalities and the simulator are given in [10].

Ideal functionality for mutual authentication. The functionality $\mathcal{F}_{\mathsf{auth}}$ roughly 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 functionality $\mathcal{F}_{\mathsf{auth}}$ can be realized by a functionality $\mathcal{P}_{\mathsf{nsl}}$ implementing the well-known Needham-Schroeder-Lowe protocol [17]. We have that $\mathcal{P}_{\mathsf{nsl}} \leq^{\mathsf{SS}} \mathcal{F}_{\mathsf{auth}}$ by showing that $\mathcal{P}_{\mathsf{nsl}} \preceq \mathcal{S}[\mathcal{F}_{\mathsf{auth}}]$ for some $\mathcal{S}$.

4.3 From one to many sessions

We have 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 14 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 which allows pushing the replication under the restricted channel c.

LEMMA 17. *Let n be a name and c be a channel name such that $c \notin fn(P) \cup fn(Q)$.*

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

This lemma allows us to apply the joint state result and obtain a result for an unbounded number of sessions sharing keys. Note that the joint state context uses a tagging mechanism.

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 its 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 be specified by the behavior of an ideal functionality. At a more fundamental level, we use 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. Another direction for future work is to give a precise characterization of what properties are preserved by strong simulatability.

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] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, 2001.

[6] 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.

[7] 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.

[8] 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.

[9] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th Comp. Security Foundations Workshop (CSFW'04)*, 2004.

[10] S. Delaune, S. Kremer, and O. Pereira. Simulation based security in the applied pi calculus. Cryptology ePrint Archive, Report 2009/267. `http://eprint.iacr.org/`.

[11] 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.

[12] 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.

[13] J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, 2000.

[14] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006.

[15] 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.

[16] 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.

[17] G. Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[18] 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.