Composing security protocols: from confidentiality to privacy [⋆]

Myrto Arapinis[1], Vincent Cheval[2,3], and Stéphanie Delaune[4]

[1] School of Informatics, University of Edinburgh, UK
[2] LORIA, CNRS, France
[3] School of Computing, University of Kent, UK
[4] LSV, CNRS & ENS Cachan, France

Abstract. Security protocols are used in many of our daily-life applications, and our privacy largely depends on their design. Formal verification techniques have proved their usefulness to analyse these protocols, but they become so complex that modular techniques have to be developed. We propose several results to safely compose security protocols. We consider arbitrary primitives modeled using an equational theory, and a rich process algebra close to the applied pi calculus.
Relying on these composition results, we derive some security properties on a protocol from the security analysis performed on each of its sub-protocols individually. We consider parallel composition and the case of key-exchange protocols. Our results apply to deal with confidentiality but also privacy-type properties (*e.g.* anonymity) expressed using a notion of equivalence. We illustrate the usefulness of our composition results on protocols from the 3G phone application and electronic passport.

1 Introduction

Privacy means that one can control when, where, and how information about oneself is used and by whom, and it is actually an important issue in many modern applications. For instance, nowadays, it is possible to wave an electronic ticket, a building access card, a government-issued ID, or even a smartphone in front of a reader to go through a gate, or to pay for some purchase. Unfortunately, as often reported by the media, this technology also makes it possible for anyone to capture some of our personal information. To secure the applications mentioned above and to protect our privacy, some specific cryptographic protocols are deployed. For instance, the 3G telecommunication application allows one to send SMS encrypted with a key that is established with the *AKA* protocol [2]. The aim of this design is to provide some security guarantees: *e.g.* the SMS exchanged between phones should remain confidential from third parties.

Since security protocols are notoriously difficult to design and analyse, formal verification techniques are important. These techniques have become mature and

[⋆] The research leading to these results has received funding from the project ProSecure (ERC grant agreement n° 258865), and the ANR project VIP n° 11 JS02 006 01.

have achieved success. For instance, a flaw has been discovered in the Single-Sign-On protocol used by Google Apps [6], and several verification tools are nowadays available (*e.g.* ProVerif [9], the AVANTSSAR platform [7]). These tools perform well in practice, at least for standard security properties (*e.g.* secrecy, authentication). Regarding privacy properties, the techniques and tools are more recent. Most of the verification techniques are only able to analyse a bounded number of sessions and consider a quite restrictive class of protocols (*e.g.* [18]). A different approach consists in analysing a stronger notion of equivalence, namely *diff-equivalence*. In particular, ProVerif implements a semi-decision procedure for checking diff-equivalence [9].

Security protocols used in practice are more and more complex and it is difficult to analyse them altogether. For example, the UMTS standard [2] specifies tens of sub-protocols running concurrently in 3G phone systems. While one may hope to verify each protocol in isolation, it is however unrealistic to expect that the whole application will be checked relying on a unique automatic tool. Existing tools have their own specificities that prevent them to be used in some cases. Furthermore, most of the techniques do not scale up well on large systems, and sometimes the ultimate solution is to rely on a manual proof. It is therefore important that the protocol under study is as small as possible.

Related work. There are many results studying the composition of security protocols in the symbolic model [15, 13, 12], as well as in the computational model [8, 16] in which the so-called UC (universal composability) framework has been first developed before being adapted in the symbolic setting [10]. Our result belongs to the first approach. Most of the existing composition results are concerned with trace-based security properties, and in most cases only with secrecy (stated as a reachability property), *e.g.* [15, 13, 12, 14]. They are quite restricted in terms of the class of protocols that can be composed, *e.g.* a fixed set of cryptographic primitives and/or no else branch. Lastly, they often only consider parallel composition. Some notable exceptions are the results presented in [17, 14, 12]. This paper is clearly inspired from the approach developed in [12].

Regarding privacy-type properties, very few composition results exist. In a previous work [4], we considered parallel composition only. More precisely, we identified sufficient conditions under which protocols can "safely" be executed in parallel as long as they have been proved secure in isolation. This composition theorem was quite general from the point of view of the cryptographic primitives allowed. We considered arbitrary primitives that can be modelled by a set of equations, and protocols may share some standard primitives provided they are tagged differently. We choose to reuse this quite general setting in this work, but our goal is now to go beyond parallel composition. We want to extend the composition theorem stated in [4] to allow a modular analysis of protocols that use other protocols as sub-programs as it happens in key-exchange protocols.

Our contributions. Our main goal is to analyse privacy-type properties in a modular way. These security properties are usually expressed as equivalences between processes. Roughly, two processes P and Q are equivalent ($P \approx Q$) if, however they behave, the messages observed by the attacker are indistinguishable. Actually, it is well-known that:

$$\text{if } P_1 \approx P_2 \text{ and } Q_1 \approx Q_2 \text{ then } P_1 \mid P_2 \approx Q_1 \mid Q_2.$$

However, this parallel composition result works because the processes that are composed are disjoint (*e.g.* they share no key). In this paper, we want to go beyond parallel composition which was already considered in [4]. In particular, we want to capture the case where a protocol uses a sub-protocol to establish some keys. To achieve this, we propose several theorems that state the conditions that need to be satisfied so that the security of the whole protocol can be derived from the security analysis performed on each sub-protocol in isolation. They are all derived from a generic composition result that allows one to map a trace of the composed protocol into a trace of the disjoint case (protocol where the sub-protocols do not share any data), and conversely. This generic result can be seen as an extension of the result presented in [12] where only a mapping from the shared case to the disjoint case is provided (but not the converse).

We also extend [12] by considering a richer process algebra. In particular, we are able to deal with protocols with else branches and to compose protocols that both rely on asymmetric primitives (*i.e.* asymmetric encryption and signature).

Outline. We present our calculus in Section 2. It can be seen as an extension of the applied pi calculus with an assignment construction. This will allow us to easily express the sharing of some data (*e.g.* session keys) between sub-protocols. In Section 3, we present a first composition result to deal with confidentiality properties. The purpose of this section is to review the difficulties that arise when composing security protocols even in a simple setting. In Section 4, we go beyond parallel composition, and we consider the case of key-exchange protocols. We present in Section 5 some additional difficulties that arise when we want to consider privacy-type properties expressed using trace equivalence. In Section 6, we present our composition results for privacy-type properties. We consider parallel composition as well as the case of key-exchange protocols. In Section 7, we illustrate the usefulness of our composition results on protocols from the 3G phone application, as well as on protocols from the e-passport application. We show how to derive some security guarantees from the analysis performed on each sub-protocol in isolation. The full version of this paper as well as the ProVerif models of our case studies can be found at `http://www.loria.fr/~chevalvi/other/compo/`.

2 Models for security protocols

Our calculus is close to the applied pi calculus [3]. We consider an assignment operation to make explicit the data that are shared among different processes.

2.1 Messages

As usual in this kind of models, messages are modelled using an abstract term algebra. We assume an infinite set of *names* $\mathcal{N}$ of *base type* (used for representing keys, nonces, ...) and a set $\mathcal{C}h$ of names of *channel type*. We also consider a set of *variables* $\mathcal{X}$, and a signature Σ consisting of a finite set of *function symbols*. We rely on a sort system for terms. The details of the sort system are unimportant, as long as the base type differs from the channel type, and we suppose that function symbols only operate on and return terms of base type.

Terms are defined as names, variables, and function symbols applied to other terms. The set of terms built from $\mathsf{N} \subseteq \mathcal{N} \cup \mathcal{C}h$, and $\mathsf{X} \subseteq \mathcal{X}$ by applying function symbols in Σ (respecting sorts and arities) is denoted by $\mathcal{T}(\Sigma, \mathsf{N} \cup \mathsf{X})$. We write $fv(u)$ (resp. $fn(u)$) for the set of variables (resp. names) occurring in a term u. A term u is *ground* if it does not contain any variable, *i.e.* $fv(u) = \emptyset$.

The algebraic properties of cryptographic primitives are specified by the means of an *equational theory* which is defined by a finite set E of equations $u = v$ with $u, v \in \mathcal{T}(\Sigma, \mathcal{X})$, *i.e.* u, v do not contain names. We denote by $=_\mathsf{E}$ the smallest equivalence relation on terms, that contains E and that is closed under application of function symbols and substitutions of terms for variables.

Example 1. Consider the signature $\Sigma_{\mathsf{DH}} = \{\mathsf{aenc}, \mathsf{adec}, \mathsf{pk}, \mathsf{g}, \mathsf{f}, \langle\ \rangle, \mathsf{proj}_1, \mathsf{proj}_2\}$. The function symbols adec, aenc of arity 2 represent asymmetric decryption and encryption. We denote by $\mathsf{pk}(sk)$ the public key associated to the private key sk. The two function symbols f of arity 2, and g of arity 1 are used to model the Diffie-Hellman primitives, whereas the three remaining symbols are used to model pairs. The equational theory E_{DH} is defined by:

$$\mathsf{E}_{\mathsf{DH}} = \left\{ \begin{array}{ll} \mathsf{proj}_1(\langle x, y\rangle) = x & \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), y) = x \\ \mathsf{proj}_2(\langle x, y\rangle) = y & \mathsf{f}(\mathsf{g}(x), y) = \mathsf{f}(\mathsf{g}(y), x) \end{array} \right.$$

Let $u_0 = \mathsf{aenc}(\langle n_A, \mathsf{g}(r_A)\rangle, \mathsf{pk}(sk_B))$. We have that:

$$\mathsf{f}(\mathsf{proj}_2(\mathsf{adec}(u_0, sk_B)), r_B) =_{\mathsf{E}_{\mathsf{DH}}} \mathsf{f}(\mathsf{g}(r_A), r_B) =_{\mathsf{E}_{\mathsf{DH}}} \mathsf{f}(\mathsf{g}(r_B), r_A).$$

2.2 Processes

As in the applied pi calculus, we consider *plain processes* as well as *extended processes* that represent processes having already evolved by *e.g.* disclosing some terms to the environment. *Plain processes* are defined by the following grammar:

$$\begin{array}{llllll} P, Q := & 0 & \text{null} & P \mid Q & \text{parallel} \\ & \mathtt{new}\ n.P & \text{restriction} & !P & \text{replication} \\ & [x := v].P & \text{assignment} & \mathtt{if}\ \varphi\ \mathtt{then}\ P\ \mathtt{else}\ Q & \text{conditional} \\ & \mathtt{in}(c, x).P & \text{input} & \mathtt{out}(c, v).Q & \text{output} \end{array}$$

where c is a name of channel type, φ is a conjunction of tests of the form $u_1 = u_2$ where u_1, u_2 are terms of base type, x is a variable of base type, v is a term of base type, and n is a name of any type. We consider an assignment operation that instantiates x with a term v. Note that we consider private channels but we do not allow channel passing. For the sake of clarity, we often omit the null process, and when there is no "`else`", it means "`else` 0".

Names and variables have scopes, which are delimited by restrictions, inputs, and assignment operations. We write $fv(P)$, $bv(P)$, $fn(P)$ and $bn(P)$ for the sets of *free* and *bound variables*, and *free* and *bound names* of a plain process P.

Example 2. Let $P_{\mathsf{DH}} = \mathtt{new}\, sk_A.\mathtt{new}\, sk_B.(P_A \mid P_B)$ a process that models a Diffie-Hellman key exchange protocol:

- $P_A \overset{\mathsf{def}}{=} \mathtt{new}\, r_A.\mathtt{new}\, n_A.\mathtt{out}(c, \mathsf{aenc}(\langle n_A, \mathsf{g}(r_A)\rangle, \mathsf{pk}(sk_B))).\mathtt{in}(c, y_A).$
  $\mathtt{if}\ \mathsf{proj}_1(\mathsf{adec}(y_A, sk_A)) = n_A\ \mathtt{then}\ [x_A := \mathsf{f}(\mathsf{proj}_2(\mathsf{adec}(y_A, sk_A)), r_A)].0$
- $P_B \overset{\mathsf{def}}{=} \mathtt{new}\, r_B.\mathtt{in}(c, y_B).\mathtt{out}(c, \mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(y_B, sk_B)), \mathsf{g}(r_B)\rangle, \mathsf{pk}(sk_A))).$
  $[x_B := \mathsf{f}(\mathsf{proj}_2(\mathsf{adec}(y_B, sk_B)), r_B)].0$

The process P_A generates two fresh random numbers r_A and n_A, sends a message on the channel c, and waits for a message containing the nonce n_A in order to compute his own view of the key that will be stored in x_A. The process P_B proceeds in a similar way and stores the computed value in x_B.

Extended processes add a set of restricted names $\mathcal{E}$ (the names that are *a priori* unknown to the attacker), a sequence of messages Φ (corresponding to the messages that have been sent so far on public channels) and a substitution σ which is used to store the messages that have been received as well as those that have been stored in assignment variables.

Definition 1. *An* extended process *is a tuple* $(\mathcal{E}; \mathcal{P}; \Phi; \sigma)$ *where* $\mathcal{E}$ *is a set of names that represents the names that are restricted in* $\mathcal{P}$, Φ *and* σ*;* $\mathcal{P}$ *is a multiset of* plain processes *where null processes are removed and such that* $fv(\mathcal{P}) \subseteq \mathrm{dom}(\sigma)$*;* $\Phi = \{w_1 \rhd u_1, \ldots, w_n \rhd u_n\}$ *and* $\sigma = \{x_1 \mapsto v_1, \ldots, x_m \mapsto v_m\}$ *are substitutions where* $u_1, \ldots, u_n, v_1, \ldots, v_m$ *are ground terms, and* $w_1, \ldots, w_n$, $x_1, \ldots, x_m$ *are variables.*

For the sake of simplicity, we assume that extended processes are *name and variable distinct*, *i.e.* a name (resp. variable) is either free or bound, and in the latter case, it is at most bound once. We write $(\mathcal{E}; P; \Phi)$ instead of $(\mathcal{E}; P; \Phi; \emptyset)$.

The semantics is given by a set of labelled rules that allows one to reason about processes that interact with their environment (see Figure 1). This defines the relation $\xrightarrow{\ell}$ where ℓ is either an input, an output, or a silent action τ. The relation $\xrightarrow{\mathsf{tr}}$ where tr denotes a sequence of labels is defined in the usual way whereas the relation $\overset{\mathsf{tr}'}{\Longrightarrow}$ on processes is defined by: $A \overset{\mathsf{tr}'}{\Longrightarrow} B$ if, and only if, there exists a sequence tr such that $A \xrightarrow{\mathsf{tr}} B$ and tr' is obtained by erasing all occurrences of the silent action τ in tr.

Example 3. Let $\Phi_{\mathsf{DH}} \overset{\mathsf{def}}{=} \{w_1 \rhd \mathsf{pk}(sk_A), w_2 \rhd \mathsf{pk}(sk_B)\}$. We have that:

$$(\{sk_A, sk_B\}; P_A \mid P_B; \Phi_{\mathsf{DH}})$$
$$\xRightarrow{\nu w_3.\mathtt{out}(c,w_3).\mathtt{in}(c,w_3).\nu w_4.\mathtt{out}(c,w_4).\mathtt{in}(c,w_4)} (\mathcal{E}; \emptyset; \Phi_{\mathsf{DH}} \uplus \Phi; \sigma \cup \sigma')$$

where $\Phi =_{\mathsf{E_{DH}}} \{w_3 \rhd u_0, w_4 \rhd \mathsf{aenc}(\langle n_A, \mathsf{g}(r_B)\rangle, pk_A)\}$, $\mathcal{E} = \{sk_A, sk_B, r_A, r_B, n_A\}$, $\sigma =_{\mathsf{E_{DH}}} \{y_A \mapsto \mathsf{aenc}(\langle n_A, \mathsf{g}(r_B)\rangle, pk_A),\ y_B \mapsto \mathsf{aenc}(\langle n_A, \mathsf{g}(r_A)\rangle, pk_B)\}$, and lastly $\sigma' =_{\mathsf{E_{DH}}} \{x_A \mapsto \mathsf{f}(\mathsf{g}(r_B), r_A),\ x_B \mapsto \mathsf{f}(\mathsf{g}(r_A), r_B)\}$. We used pk_A (resp. pk_B) as a shorthand for $\mathsf{pk}(sk_A)$ (resp. $\mathsf{pk}(sk_B)$).

$$(\mathcal{E}; \{\mathtt{if}\ \varphi\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau} (\mathcal{E}; Q_1 \uplus \mathcal{P}; \Phi; \sigma) \qquad (\text{THEN})$$
$$\text{if } u\sigma =_{\mathsf{E}} v\sigma \text{ for each } u = v \in \varphi$$

$$(\mathcal{E}; \{\mathtt{if}\ \varphi\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau} (\mathcal{E}; Q_2 \uplus \mathcal{P}; \Phi; \sigma) \qquad (\text{ELSE})$$
$$\text{if } u\sigma \neq_{\mathsf{E}} v\sigma \text{ for some } u = v \in \varphi$$

$$(\mathcal{E}; \{\mathtt{out}(c,u).Q_1; \mathtt{in}(c,x).Q_2\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau} (\mathcal{E}; Q_1 \uplus Q_2 \uplus \mathcal{P}; \Phi; \sigma \cup \{x \mapsto u\sigma\}) (\text{COMM})$$

$$(\mathcal{E}; \{[x := v].Q\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau} (\mathcal{E}; Q \uplus \mathcal{P}; \Phi; \sigma \cup \{x \mapsto v\sigma\}) \qquad (\text{ASSGN})$$

$$(\mathcal{E}; \{\mathtt{in}(c,z).Q\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\mathtt{in}(c,M)} (\mathcal{E}; Q \uplus \mathcal{P}; \Phi; \sigma \cup \{z \mapsto u\}) \qquad (\text{IN})$$
$$\text{if } c \notin \mathcal{E},\ M\Phi = u,\ \mathit{fv}(M) \subseteq \mathrm{dom}(\Phi) \text{ and } \mathit{fn}(M) \cap \mathcal{E} = \emptyset$$

$$(\mathcal{E}; \{\mathtt{out}(c,u).Q\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\nu w_i.\mathtt{out}(c,w_i)} (\mathcal{E}; Q \uplus \mathcal{P}; \Phi \cup \{w_i \rhd u\sigma\}; \sigma) \qquad (\text{OUT-T})$$
$$\text{if } c \notin \mathcal{E},\ u \text{ is a term of base type, and } w_i \text{ is a variable such that } i = |\Phi| + 1$$

$$(\mathcal{E}; \{\mathtt{new}\ n.Q\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau} (\mathcal{E} \cup \{n\}; Q \uplus \mathcal{P}; \Phi; \sigma) \qquad (\text{NEW})$$

$$(\mathcal{E}; \{!Q\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau} (\mathcal{E}; \{!Q; Q\rho\} \uplus \mathcal{P}; \Phi; \sigma) \qquad (\text{REPL})$$
$$\rho \text{ is used to rename } \mathit{bv}(Q)/\mathit{bn}(Q) \text{ with fresh variables/names}$$

$$(\mathcal{E}; \{P_1 \mid P_2\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau} (\mathcal{E}; \{P_1, P_2\} \uplus \mathcal{P}; \Phi; \sigma) \qquad (\text{PAR})$$

where n is a name, c is a name of channel type, u, v are terms of base type, and x, z are variables of base type.

Fig. 1. Semantics of extended processes

2.3 Process equivalences

We are particularly interested in security properties expressed using a notion of equivalence such as those studied in *e.g.* [5, 11]. For instance, the notion of *strong unlinkability* can be formalized using an equivalence between two situations: one where each user can execute the protocol multiple times, and one where each user can execute the protocol at most once.

We consider here the notion of *trace equivalence*. Intuitively, two protocols P and Q are in trace equivalence, denoted $P \approx Q$, if whatever the messages they received (built upon previously sent messages), the resulting sequences of messages sent on public channels are indistinguishable from the point of view of an outsider. Given an extended process A, we define its set of traces as follows:

$$\mathsf{trace}(A) = \{(\mathsf{tr}, \mathtt{new}\ \mathcal{E}.\Phi) \mid A \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{E}; \mathcal{P}; \Phi; \sigma) \text{ for some process } (\mathcal{E}; \mathcal{P}; \Phi; \sigma)\}.$$

The sequence of messages Φ together with the set of restricted names $\mathcal{E}$ (those unknown to the attacker) is called the *frame*.

Definition 2. *We say that a term u is* deducible *(modulo E) from a frame $\phi = \mathtt{new}\mathcal{E}.\Phi$, denoted $\mathtt{new}\mathcal{E}.\Phi \vdash u$, when there exists a term M (called a* recipe*) such that $\mathit{fn}(M) \cap \mathcal{E} = \emptyset$, $\mathit{fv}(M) \subseteq \mathrm{dom}(\Phi)$, and $M\Phi =_{\mathsf{E}} u$.*

Two frames are indistinguishable when the attacker cannot detect the difference between the two situations they represent.

Definition 3. *Two frames ϕ_1 and ϕ_2 with $\phi_i = \mathtt{new}\mathcal{E}.\Phi_i$ ($i \in \{1,2\}$) are* statically equivalent*, denoted by $\phi_1 \sim \phi_2$, when* $\mathrm{dom}(\Phi_1) = \mathrm{dom}(\Phi_2)$*, and for all terms M, N with $fn(\{M,N\}) \cap \mathcal{E} = \emptyset$ and $fv(\{M,N\}) \subseteq \mathrm{dom}(\Phi_1)$, we have that:*

$$M\Phi_1 =_{\mathsf{E}} N\Phi_1, \text{ if and only if, } M\Phi_2 =_{\mathsf{E}} N\Phi_2.$$

Example 4. Consider $\Phi_1 = \{w_1 \rhd \mathsf{g}(r_A), w_2 \rhd \mathsf{g}(r_B), w_3 \rhd \mathsf{f}(\mathsf{g}(r_A), r_B)\}$, and $\Phi_2 = \{w_1 \rhd \mathsf{g}(r_A), w_2 \rhd \mathsf{g}(r_B), w_3 \rhd k\}$. Let $\mathcal{E} = \{r_A, r_B, k\}$. We have that $\mathtt{new}\,\mathcal{E}.\Phi_1 \sim \mathtt{new}\,\mathcal{E}.\Phi_2$ (considering the equational theory E_{DH}). This equivalence shows that the term $\mathsf{f}(\mathsf{g}(r_A), r_B)$ (the Diffie-Hellman key) is indistinguishable from a random key. This indistinguishability property holds even if the messages $\mathsf{g}(r_A)$ and $\mathsf{g}(r_B)$ have been observed by the attacker.

Two processes are trace equivalent if, whatever the messages they sent and received, their frames are in static equivalence.

Definition 4. *Let A and B be two extended processes, $A \sqsubseteq B$ if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}(A)$, there exists $(\mathsf{tr}', \phi') \in \mathsf{trace}(B)$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\phi \sim \phi'$. We say that A and B are* trace equivalent*, denoted by $A \approx B$, if $A \sqsubseteq B$ and $B \sqsubseteq A$.*

This notion of equivalence allows us to express many interesting privacy-type properties *e.g.* vote-privacy, strong versions of anonymity and/or unlinkability.

3 Composition result: a simple setting

It is well-known that even if two protocols are secure in isolation, it is not possible to compose them in arbitrary ways still preserving their security. This has already been observed for different kinds of compositions (*e.g.* parallel [15], sequential [12]) and when studying standard security properties [13] and even privacy-type properties [4]. In this section, we introduce some well-known hypotheses that are needed to safely compose security protocols.

3.1 Sharing primitives

A protocol can be used as an oracle by another protocol to decrypt a message, and then compromise the security of the whole application. To avoid this kind of interactions, most of the composition results assume that protocols do not share any primitive or allow a list of standard primitives (*e.g.* signature, encryption) to be shared as long as they are tagged in different ways. In this paper, we adopt the latter hypothesis and consider the fixed common signature:

$$\Sigma_0 = \{\mathsf{sdec}, \mathsf{senc}, \mathsf{adec}, \mathsf{aenc}, \mathsf{pk}, \langle\,,\rangle, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{sign}, \mathsf{check}, \mathsf{vk}, \mathsf{h}\}$$

equipped with the equational theory E_0, defined by the following equations:

$$\mathsf{sdec}(\mathsf{senc}(x,y),y) = x \qquad \mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) = x$$
$$\mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) = x \qquad \mathsf{proj}_i(\langle x_1, x_2\rangle) = x_i \text{ with } i \in \{1,2\}$$

This allows us to model symmetric/asymmetric encryption, concatenation, signatures, and hash functions. We consider a type *seed* which is a subsort of the base type that only contains names. We denote by $\mathsf{pk}(sk)$ (resp. $\mathsf{vk}(sk)$) the public key (resp. the verification key) associated to the private key sk which has to be a name of type *seed*. We allow protocols to both rely on Σ_0 provided that each application of aenc, senc, sign, and h is tagged (using disjoint sets of tags for the two protocols), and adequate tests are performed when receiving a message to ensure that the tags are correct. Actually, we consider the same tagging mechanism as the one we have introduced in [4] (see Definitions 4 and 5 in [4]). In particular, we rely on the same notation: we use the two function symbols $\mathsf{tag}/\mathsf{untag}$, and the equation $\mathsf{untag}(\mathsf{tag}(x)) = x$ to model the interactions between them. However, since we would like to be able to iterate our composition results (in order to compose *e.g.* three protocols), we consider a family of such function symbols: $\mathsf{tag}_i/\mathsf{untag}_i$ with $i \in \mathbb{N}$. Moreover, a process may be tagged using a subset of such symbols (and not only one). This gives us enough flexibility to allow different kinds of compositions, and to iterate our composition results.

Example 5. In order to compose the protocol introduced in Example 2 with another one that also relies on the primitive aenc, we may want to consider a tagged version of this protocol. The tagged version (using $\mathsf{tag}_1/\mathsf{untag}_1$) of P_B is given below (with $u = \mathsf{untag}_1(\mathsf{adec}(y_B, sk_B))$):

$$\begin{cases} \mathtt{new}\, r_B.\mathtt{in}(c, y_B). \\ \mathtt{if}\ \mathsf{tag}_1(\mathsf{untag}_1(\mathsf{adec}(y_B, sk_B))) = \mathsf{adec}(y_B, sk_B)\ \mathtt{then} \\ \mathtt{if}\ u = \langle \mathsf{proj}_1(u), \mathsf{proj}_2(u)\rangle\ \mathtt{then} \\ \mathtt{out}(c, \mathsf{aenc}(\mathsf{tag}_1(\langle \mathsf{proj}_1(u), \mathsf{g}(r_B)\rangle), \mathsf{pk}(sk_A))).[x_B := \mathsf{f}(\mathsf{proj}_2(u), r_B)].0 \end{cases}$$

The first test allows one to check that y_B is an encryption tagged with tag_1 and the second one is used to ensure that the content of this encryption is a pair as expected. Then, the process outputs the encrypted message tagged with tag_1.

3.2 Revealing shared keys

Consider two protocols, one whose security relies on the secrecy of a shared key whereas the other protocol reveals it. Such a situation will compromise the security of the whole application. It is therefore important to ensure that shared keys are not revealed. To formalise this hypothesis, and to express the sharing of long-term keys, we introduce the notion of *composition context*. This will help us describe under which long-term keys the composition has to be done.

A *composition context* C is defined by the grammar:

$$C := \_ \mid \mathtt{new}\ n.\ C \mid !C \qquad \text{where } n \text{ is a name of base type.}$$

Definition 5. *Let C be a composition context, A be an extended process of the form $(\mathcal{E}; C[P]; \Phi)$, $key \in \{n, \mathsf{pk}(n), \mathsf{vk}(n) \mid n \text{ occurs in } C\}$, and c, s two fresh names. We say that A* reveals *key when*

$$(\mathcal{E} \cup \{s\}; C[P \mid \mathtt{in}(c, x).\ \mathtt{if}\ x = key\ \mathtt{then}\, \mathtt{out}(c, s)]; \Phi) \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{E}'; \mathcal{P}'; \Phi'; \sigma')$$

for some $\mathcal{E}'$, $\mathcal{P}'$, Φ', and σ' such that $\mathtt{new}\, \mathcal{E}'.\Phi' \vdash s$.

3.3 A first composition result

Before stating our first result regarding parallel composition for confidentiality properties, we gather the required hypotheses in the following definition.

Definition 6. *Let C be a composition context and $\mathcal{E}_0$ be a finite set of names of base type. Let P and Q be two plain processes together with their frames Φ and Ψ. We say that P/Φ and Q/Ψ are* composable *under $\mathcal{E}_0$ and C when $fv(P) = fv(Q) = \emptyset$, $\mathrm{dom}(\Phi) \cap \mathrm{dom}(\Psi) = \emptyset$, and*

1. *P (resp. Q) is built over $\Sigma_\alpha \cup \Sigma_0$ (resp. $\Sigma_\beta \cup \Sigma_0$), whereas Φ (resp. Ψ) is built over $\Sigma_\alpha \cup \{\mathsf{pk}, \mathsf{vk}, \langle\ \rangle\}$ (resp. $\Sigma_\beta \cup \{\mathsf{pk}, \mathsf{vk}, \langle\ \rangle\}$), $\Sigma_\alpha \cap \Sigma_\beta = \emptyset$, and P (resp. Q) is tagged;*
2. *$\mathcal{E}_0 \cap (fn(C[P]) \cup fn(\Phi)) \cap (fn(C[Q]) \cup fn(\Psi)) = \emptyset$; and*
3. *$(\mathcal{E}_0; C[P]; \Phi)$ (resp. $(\mathcal{E}_0; C[Q]; \Psi)$) does not reveal any key in*
$$\{n, \mathsf{pk}(n), \mathsf{vk}(n) \mid n \textit{ occurs in } fn(P) \cap fn(Q) \cap bn(C)\}.$$

Condition 1 is about sharing primitives, whereas Conditions 2 and 3 ensure that keys are shared via the composition context C only (not via $\mathcal{E}_0$), and are not revealed by each protocol individually.

We are now able to state the following theorem which is in the same vein as those obtained previously in *e.g.* [15, 13]. However, the setting we consider here is more general. In particular, we consider arbitrary primitives, processes with else branches, and private channels.

Theorem 1. *Let C be a composition context, $\mathcal{E}_0$ be a finite set of names of base type, and s be a name that occurs in C. Let P and Q be two plain processes together with their frames Φ and Ψ, and assume that P/Φ and Q/Ψ are composable under $\mathcal{E}_0$ and C. If $(\mathcal{E}_0; C[P]; \Phi)$ and $(\mathcal{E}_0; C[Q]; \Psi)$ do not reveal s then $(\mathcal{E}_0; C[P \mid Q]; \Phi \uplus \Psi)$ does not reveal s.*

As most of the proofs of similar composition results, we show this result going back to the *disjoint case*. Indeed, it is well-known that parallel composition works well when protocols do not share any data (the so-called *disjoint case*). We show that all the conditions are satisfied to apply our generic result (presented only in the full version of this paper) that allows one to go back to the disjoint case. Thus, we obtain that the disjoint case $D = (\mathcal{E}_0; C[P] \mid C[Q]; \Phi \uplus \Psi)$ and the shared case $S = (\mathcal{E}_0; C[P \mid Q]; \Phi \uplus \Psi)$ are in trace equivalence, and this allows us to conclude.

4 The case of key-exchange protocols

Our goal is to go beyond parallel composition, and to further consider the particular case of key-exchange protocols. Assume that $P = \mathtt{new}\, \tilde{n}.(P_1 \mid P_2)$ is a protocol that establishes a key between two parties. The goal of P is to establish a shared session key between P_1 and P_2. Assume that P_1 stores the key in the variable x_1, while P_2 stores it in the variable x_2, and then consider a protocol Q that uses the values stored in x_1/x_2 as a fresh key to secure communications.

4.1 What is a good key exchange protocol?

In this setting, sharing between P and Q is achieved through the composition context as well as through assignment variables x_1 and x_2. The idea is to abstract these values with fresh names when we analyse Q in isolation. However, in order to abstract them in the right way, we need to know their values (or at least whether they are equal or not). This is the purpose of the property stated below.

Definition 7. *Let C be a composition context and $\mathcal{E}_0$ be a finite set of names. Let $P_1[\_]$ (resp. $P_2[\_]$) be a plain process with a hole in the scope of an assignment of the form $[x_1 := t_1]$ (resp. $[x_2 := t_2]$), and Φ be a frame.*

We say that $P_1/P_2/\Phi$ is a good *key-exchange protocol under $\mathcal{E}_0$ and C when $(\mathcal{E}_0; P_{\mathsf{good}}; \Phi)$ does not reveal bad where P_{good} is defined as follows:*

$$\begin{aligned}
&P_{\mathsf{good}} = \mathtt{new}\, bad.\mathtt{new} d.\big(C[\mathtt{new} id.(P_1[\mathtt{out}(d,\langle x_1, id\rangle)] \mid P_2[\mathtt{out}(d,\langle x_2, id\rangle)])]\\
&\mid \mathtt{in}(d,x).\mathtt{in}(d,y).\mathtt{if}\ \mathsf{proj}_1(x) = \mathsf{proj}_1(y) \wedge \mathsf{proj}_2(x) \neq \mathsf{proj}_2(y)\ \mathtt{then}\, \mathtt{out}(c, bad)\\
&\mid \mathtt{in}(d,x).\mathtt{in}(d,y).\mathtt{if}\ \mathsf{proj}_1(x) \neq \mathsf{proj}_1(y) \wedge \mathsf{proj}_2(x) = \mathsf{proj}_2(y)\ \mathtt{then}\, \mathtt{out}(c, bad)\\
&\mid \mathtt{in}(d,x).\mathtt{in}(c,z).\mathtt{if}\, z \in \{\mathsf{proj}_1(x), \mathsf{pk}(\mathsf{proj}_1(x)), \mathsf{vk}(\mathsf{proj}_1(x))\}\ \mathtt{then}\, \mathtt{out}(c, bad)\big)
\end{aligned}$$

where bad is a fresh name of base type, and c, d are fresh names of channel type.

The expressions $u \neq v$ and $u \in \{v_1, \dots, v_n\}$ used above are convenient notations that can be rigorously expressed using nested conditionals. Roughly, the property expresses that x_1 and x_2 are assigned to the same value if, and only if, they are joined together, *i.e.* they share the same *id*. In particular, two instances of the role P_1 (resp. P_2) cannot assign their variable with the same value: a fresh key is established at each session. The property also ensures that the data shared through x_1/x_2 are not revealed.

Example 6. We have that $P_A/P_B/\Phi_{\mathsf{DH}}$ described in Example 2, as well as its tagged version (see Example 5) are *good* key-exchange protocols under $\mathcal{E}_0 = \{sk_A, sk_B\}$ and $C = \_$. This corresponds to a scenario where we consider only a single execution of the protocol (no replication).

Actually, the property mentioned above is quite strong, and never satisfied when the context C under study ends with a replication, *i.e.* when C is of the form $C'[!\_]$. To cope with this situation, we consider another version of this property. When C is of the form $C'[!\_]$, we define P_{good} as follows (where r_1 and r_2 are two additional fresh names of base type):

$$\begin{aligned}
&\mathtt{new}\, bad, d, r_1, r_2.\big(C'[\mathtt{new}\, id.!(P_1[\mathtt{out}(d,\langle x_1, id, r_1\rangle)] \mid P_2[\mathtt{out}(d,\langle x_2, id, r_2\rangle)])]\\
&\mid \mathtt{in}(d,x).\mathtt{in}(d,y).\mathtt{if}\ \mathsf{proj}_1(x) = \mathsf{proj}_1(y) \wedge \mathsf{proj}_2(x) \neq \mathsf{proj}_2(y)\, \mathtt{then}\, \mathtt{out}(c, bad)\\
&\mid \mathtt{in}(d,x).\mathtt{in}(d,y).\mathtt{if}\ \mathsf{proj}_1(x) = \mathsf{proj}_1(y) \wedge \mathsf{proj}_3(x) = \mathsf{proj}_3(y)\, \mathtt{then}\, \mathtt{out}(c, bad)\\
&\mid \mathtt{in}(d,x).\mathtt{in}(c,z).\mathtt{if}\, z \in \{\mathsf{proj}_1(x), \mathsf{pk}(\mathsf{proj}_1(x)), \mathsf{vk}(\mathsf{proj}_1(x))\}\, \mathtt{then}\, \mathtt{out}(c, bad)\big)
\end{aligned}$$

Note that the *id* is now generated before the last replication, and thus is not uniquely associated to an instance of P_1/P_2. Instead several instances of P_1/P_2 may now share the same *id* as soon as they are identical. This gives us more flexibility. The triplet $\langle u_1, u_2, u_3\rangle$ and the operator $\mathsf{proj}_3(u)$ used above are convenient notations that can be expressed using pairs. This new version forces distinct values in the assignment variables for each instance of P_1 (resp. P_2) through the 3rd line. However, we do not fix in advance which particular instance of P_1 and P_2 should be matched, as in the first version.

Example 7. We have that $P_A/P_B/\Phi_{\mathsf{DH}}$ as well as its tagged version are good key-exchange protocols under $\mathcal{E}_0 = \{sk_A, sk_B\}$ and $C =!\,\_$.

4.2 Do we need to tag pairs?

When analysing Q in isolation, the values stored in the assignment variables x_1/x_2 are abstracted by fresh names. Since P and Q share the common signature Σ_0, we need an additional hypothesis to ensure that in any execution, the values assigned to the variables x_1/x_2 are not of the form $\langle u_1, u_2\rangle$, $\mathsf{pk}(u)$, or $\mathsf{vk}(u)$. These symbols are those of the common signature that are not tagged, thus abstracting them by fresh names in Q would not be safe. This has already been highlighted in [12]. They however left as future work the definition of the needed hypothesis and simply assume that each operator of the common signature has to be tagged. Here, we formally express the required hypothesis.

Definition 8. *An extended process A satisfies the* abstractability property *if for any $(\mathcal{E};\mathcal{P};\Phi;\sigma)$ such that $A\overset{\mathsf{tr}}{\Longrightarrow}(\mathcal{E};\mathcal{P};\Phi;\sigma)$, for any $x \in \mathrm{dom}(\sigma)$ which corresponds to an assignment variable, for any u_1, u_2, we have that $x\sigma \neq_{\mathsf{E}} \langle u_1, u_2\rangle$, $x\sigma \neq_{\mathsf{E}} \mathsf{pk}(u_1)$, and $x\sigma \neq_{\mathsf{E}} \mathsf{vk}(u_1)$.*

Note also that, in [12], the common signature is restricted to symmetric encryption and pairing only. They do not consider asymmetric encryption, and signature. Thus, our composition result generalizes theirs considering both a richer common signature, and a lighter tagging scheme (we do not tag pairs).

4.3 Composition result

We retrieve the following result which is actually a generalization of two theorems established in [12] and stated for specific composition contexts.

Theorem 2. *Let C be a composition context, $\mathcal{E}_0$ be a finite set of names of base type, and s be a name that occurs in C. Let $P_1[\_]$ (resp. $P_2[\_]$) be a plain process without replication and with an hole in the scope of an assignment of the form $[x_1 := t_1]$ (resp. $[x_2 := t_2]$). Let Q_1 (resp. Q_2) be a plain process such that $fv(Q_1) \subseteq \{x_1\}$ (resp. $fv(Q_2) \subseteq \{x_2\}$), and Φ and Ψ be two frames. Let $P = P_1[0] \mid P_2[0]$ and $Q = \mathtt{new}\ k.[x_1 := k].[x_2 := k].(Q_1 \mid Q_2)$ for some fresh name k, and assume that:*

1. *P/Φ and Q/Ψ are composable under $\mathcal{E}_0$ and C;*
2. *$(\mathcal{E}_0; C[Q]; \Psi)$ does not reveal k, $\mathsf{pk}(k)$, $\mathsf{vk}(k)$;*
3. *$(\mathcal{E}_0; C[P]; \Phi)$ satisfies the abstractability property; and*
4. *$P_1/P_2/\Phi$ is a good key-exchange protocol under $\mathcal{E}_0$ and C.*

If $(\mathcal{E}_0; C[P]; \Phi)$ *and* $(\mathcal{E}_0; C[Q]; \Psi)$ *do not reveal* s *then* $(\mathcal{E}_0; C[P_1[Q_1]|P_2[Q_2]]; \Phi \uplus \Psi)$ *does not reveal* s.

Basically, we prove this result relying on our generic composition result. In [12], they do not require P to be good but only ask for secrecy of the shared key. In particular they do not express any freshness or agreement property about the established key. Actually, when considering a simple composition context without replication, freshness is trivial (since there is only one session). Moreover, in their setting, agreement is not important since they do not have else branches. The analysis of Q considering that both parties have agreed on the key corresponds to the worst scenario. Note that this is not true anymore in presence of else branches. The following example shows that as soon as else branches are allowed, as it is the case in the present work, agreement becomes important.

Example 8. Consider a simple situation where:

- $P_1[0] = \mathtt{new}\, k_1.[x_1 := k_1].0$ and $P_2[0] = \mathtt{new}\, k_2.[x_2 := k_2].0$;
- $Q_1 = \mathtt{if}\ \ x_1 = x_2\ \ \mathtt{then}\ \ \mathtt{out}(c, \mathsf{ok})\ \ \mathtt{else}\ \ \mathtt{out}(c, s)$ and $Q_2 = 0$.

Let $\mathcal{E}_0 = \emptyset$, and $C = \mathtt{new}\ s.\_$. We consider the processes $P = P_1[0] \mid P_2[0]$, and $Q = \mathtt{new}\, k.[x_1 := k].[x_2 := k].(Q_1 \mid Q_2)$ and we assume that the frames Φ and Ψ are empty. We clearly have that $(\mathcal{E}_0; C[P]; \Phi)$ and $(\mathcal{E}_0; C[Q]; \Psi)$ do not reveal s whereas $(\mathcal{E}_0; C[P_1[Q_1] \mid P_2[Q_2]; \Phi \uplus \Psi)$ does. The only hypothesis of Theorem 2 that is violated is the fact that $P_1/P_2/\Phi$ is not a good key-exchange protocol due to a lack of agreement on the key which is generated (*bad* can be emitted thanks to the 3rd line of the process P_{good} given in Definition 7).

Now, regarding their second theorem corresponding to a context of the form $\mathtt{new}\, s.\,!\_$, as before agreement is not mandatory but freshness of the key established by the protocol P is crucial. As illustrated by the following example, this hypothesis is missing in the theorem stated in [12] (Theorem 3).

Example 9. Consider $A = (\{k_P\}; \mathtt{new}\, s.!([x_1 := k_P].0 \mid [x_2 := k_P].0); \emptyset)$, as well as $B = (\{k_P\}; \mathtt{new}\, s.\,!Q; \emptyset)$ where $Q = \mathtt{new}\, k.[x_1 := k].[x_2 := k].(Q_1 \mid Q_2)$ with

$$Q_1 = \mathtt{out}(c, \mathsf{senc}(\mathsf{senc}(s, k), k));\text{ and } Q_2 = \mathtt{in}(c, x).\mathtt{out}(c, \mathsf{sdec}(x, k)).$$

Note that neither A nor B reveals s. In particular, the process Q_1 emits the secret s encrypted twice with a fresh key k, but Q_2 only allows us to remove one level of encryption with k. Now, if we plug the key-exchange protocol given above with no guarantee of freshness (the same key is established at each session), the resulting process, *i.e.* $(\mathcal{E}_0; C[P_1[Q_1] \mid P_2[Q_2]]; \emptyset)$ does reveal s.

Note that this example is not a counter example of our Theorem 2: $P_1/P_2/\emptyset$ is not a good key-exchange protocol according to our definition.

5 Dealing with equivalence-based properties

Our ultimate goal is to analyse privacy-type properties in a modular way. In [4], we propose several composition results w.r.t. privacy-type properties, but for parallel composition only. Here, we want to go beyond parallel composition, and consider the case of key-exchange protocols.

5.1 A problematic example

Even in a quite simple setting (the shared keys are not revealed, protocols do not share any primitives), such a sequential composition result does not hold. Let $C = \mathtt{new}\ k.!\,\mathtt{new}\ k_1.!\,\mathtt{new}\ k_2.\ \_$ be a composition context, yes/no, ok/ko be public constants, $u = \mathsf{senc}(\langle k_1, k_2\rangle, k)$, and consider the following processes:

$$\begin{array}{rcl} Q(z_1, z_2) & = & \mathtt{out}(c, u).\mathtt{in}(c, x).\mathtt{if}\ x = u\ \mathtt{then}\ 0\ \mathtt{else} \\ & & \mathtt{if}\ \mathsf{proj}_1(\mathsf{sdec}(x, k)) = k_1\ \mathtt{then}\ \mathtt{out}(c, z_1)\ \mathtt{else}\ \mathtt{out}(c, z_2) \\ P[\_] & = & \mathtt{out}(c, u).\big(\_ \mid \mathtt{in}(c, x).\mathtt{if}\ x = u\ \mathtt{then}\ 0\ \mathtt{else} \\ & & \mathtt{if}\ \mathsf{proj}_1(\mathsf{sdec}(x, k)) = k_1\ \mathtt{then}\ \mathtt{out}(c, \mathsf{ok})\ \mathtt{else}\ \mathtt{out}(c, \mathsf{ko})\big) \end{array}$$

We have that $C[P[0]] \approx C[P[0]]$ and also that $C[Q(\mathsf{yes}, \mathsf{no})] \approx C[Q(\mathsf{no}, \mathsf{yes})]$. This latter equivalence is non-trivial. Intuitively, when $C[Q(\mathsf{yes}, \mathsf{no})]$ unfolds its outermost ! and then performs an output, then $C[Q(\mathsf{no}, \mathsf{yes})]$ has to mimic this step by unfolding its innermost ! and by performing the only available output. This will allow it to react in the same way as $C[Q(\mathsf{yes}, \mathsf{no})]$ in case encrypted messages are used to fill some input actions. Since the two processes $P[0]$ and $Q(\mathsf{yes}, \mathsf{no})$ (resp. $Q(\mathsf{no}, \mathsf{yes})$) are almost "disjoint", we could expect the equivalence $C[P[Q(\mathsf{yes}, \mathsf{no})]] \approx C[P[Q(\mathsf{no}, \mathsf{yes})]]$ to hold. Actually, this equivalence does *not* hold. The presence of the process P gives to the attacker some additional distinguishing power. In particular, through the outputs ok/ko outputted by P, the attacker will learn which ! has been unfolded. This result holds even if we rename function symbols so that protocols P and Q do not share any primitives. The problem is that the two equivalences we want to compose hold for different reasons, *i.e.* by unfolding the replications in a different and incompatible way. Thus, when the composed process $C[P[Q(\mathsf{yes}, \mathsf{no})]]$ reaches a point where $Q(\mathsf{yes}, \mathsf{no})$ can be executed, on the other side, the process $Q(\mathsf{no}, \mathsf{yes})$ is ready to be executed but the instance that is available is not the one that was used when establishing the equivalence $C[Q(\mathsf{yes}, \mathsf{no})] \approx C[Q(\mathsf{no}, \mathsf{yes})]$. Therefore, in order to establish equivalence-based properties in a modular way, we rely on a stronger notion of equivalence, namely *diff-equivalence*, that will ensure that the two "small" equivalences are satisfied in a compatible way.

Note that this problem does not arise when considering reachability properties and/or parallel composition. In particular, we have that:

$$C[P[0] \mid Q(\mathsf{yes}, \mathsf{no})] \approx C[P[0] \mid Q(\mathsf{no}, \mathsf{yes})].$$

5.2 Biprocesses and diff-equivalence

We consider pairs of processes, called *biprocesses*, that have the same structure and differ only in the terms and tests that they contain. Following the approach of [9], we introduce a special symbol diff of arity 2 in our signature. The idea being to use this diff operator to indicate when the terms manipulated by the processes are different. Given a biprocess B, we define two processes $\mathsf{fst}(B)$ and $\mathsf{snd}(B)$ as follows: $\mathsf{fst}(B)$ is obtained by replacing each occurrence of $\mathsf{diff}(M, M')$ (resp. $\mathsf{diff}(\varphi, \varphi')$) with M (resp. φ), and similarly $\mathsf{snd}(B)$ is obtained by replacing each occurrence of $\mathsf{diff}(M, M')$ (resp. $\mathsf{diff}(\varphi, \varphi')$) with M' (resp. φ').

The semantics of biprocesses is defined as expected via a relation that expresses when and how a biprocess may evolve. A biprocess reduces if, and only if, both sides of the biprocess reduce in the same way: a communication succeeds on both sides, a conditional has to be evaluated in the same way in both sides too. For instance, the `then` and `else` rules are as follows:

$$(\mathcal{E}; \{\texttt{if } \mathsf{diff}(\varphi_L, \varphi_R) \texttt{ then } Q_1 \texttt{ else } Q_2\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau}_{\mathsf{bi}} (\mathcal{E}; Q_1 \uplus \mathcal{P}; \Phi; \sigma)$$
$$\text{if } u\sigma =_{\mathsf{E}} v\sigma \text{ for each } u = v \in \varphi_L, \text{ and } u'\sigma =_{\mathsf{E}} v'\sigma \text{ for each } u' = v' \in \varphi_R$$

$$(\mathcal{E}; \{\texttt{if } \mathsf{diff}(\varphi_L, \varphi_R) \texttt{ then } Q_1 \texttt{ else } Q_2\} \uplus \mathcal{P}; \Phi; \sigma) \xrightarrow{\tau}_{\mathsf{bi}} (\mathcal{E}; Q_2 \uplus \mathcal{P}; \Phi; \sigma)$$
$$\text{if } u\sigma \neq_{\mathsf{E}} v\sigma \text{ for some } u = v \in \varphi_L, \text{ and } u'\sigma \neq_{\mathsf{E}} v'\sigma \text{ for some } u' = v' \in \varphi_R$$

When the two sides of the biprocess reduce in different ways, the biprocess blocks. The relation $\overset{\mathsf{tr}}{\Longrightarrow}_{\mathsf{bi}}$ on biprocesses is defined as for processes. This leads us to the following notion of *diff-equivalence.*

Definition 9. *An extended biprocess B_0 satisfies* diff-equivalence *if for every biprocess $B = (\mathcal{E}; \mathcal{P}; \Phi; \sigma)$ such that $B_0 \overset{\mathsf{tr}}{\Longrightarrow}_{\mathsf{bi}} B$ for some trace* tr*, we have that*

1. *new $\mathcal{E}.\mathsf{fst}(\Phi) \sim$ new $\mathcal{E}.\mathsf{snd}(\Phi)$*
2. *if $\mathsf{fst}(B) \xrightarrow{\ell} A_L$ then there exists B' such that $B \xrightarrow{\ell}_{\mathsf{bi}} B'$ and $\mathsf{fst}(B') = A_L$ (and similarly for* snd*).*

The notions introduced so far on processes are extended as expected on biprocesses: the property has to hold on both $\mathsf{fst}(B)$ and $\mathsf{snd}(B)$. Sometimes, we also say that the biprocess B is in trace equivalence instead of writing $\mathsf{fst}(B) \approx \mathsf{snd}(B)$.

As expected, this notion of diff-equivalence is actually stronger than the usual notion of trace equivalence.

Lemma 1. *A biprocess B that satisfies diff-equivalence is in trace equivalence.*

6 Composition results for diff-equivalence

We first consider the case of parallel composition. This result is in the spirit of the one established in [4]. However, we adapt it to diff-equivalence in order to combine it with the composition result we obtained for the the case of key-exchange protocol (see Theorem 4).

Theorem 3. *Let C be a composition context and $\mathcal{E}_0$ be a finite set of names of base type. Let P and Q be two plain biprocesses together with their frames Φ and Ψ, and assume that P/Φ and Q/Ψ are composable under $\mathcal{E}_0$ and C.*

If $(\mathcal{E}_0; C[P]; \Phi)$ and $(\mathcal{E}_0; C[Q]; \Psi)$ satisfy diff-equivalence (resp. trace equivalence) then the biprocess $(\mathcal{E}_0; C[P \mid Q]; \Phi \uplus \Psi)$ satisfies diff-equivalence (resp. trace equivalence).

Proof. (sketch) As for the proof for Theorem 1, parallel composition works well when processes do not share any data. Hence, we easily deduce that $D = (\mathcal{E}_0; C[P] \mid C[Q]; \Phi \uplus \Psi)$ satisfies the diff-equivalence (resp. trace equivalence).

Then, we compare the behaviours of the biprocess D to those of the biprocess $S = (\mathcal{E}_0; C[P \mid Q]; \Phi \uplus \Psi)$. More precisely, this allows us to establish that $\mathsf{fst}(D)$ and $\mathsf{fst}(S)$ are in diff-equivalence (as well as $\mathsf{snd}(D)$ and $\mathsf{snd}(S)$), and then we conclude relying on the transitivity of the equivalence. □

Now, regarding sequential composition and the particular case of key-exchange protocols, we obtain the following composition result.

Theorem 4. *Let C be a composition context and $\mathcal{E}_0$ be a finite set of names of base type. Let $P_1[\_]$ (resp. $P_2[\_]$) be a plain biprocess without replication and with an hole in the scope of an assignment of the form $[x_1 := t_1]$ (resp. $[x_2 := t_2]$). Let Q_1 (resp. Q_2) be a plain biprocess such that $fv(Q_1) \subseteq \{x_1\}$ (resp. $fv(Q_2) \subseteq \{x_2\}$), and Φ and Ψ be two frames. Let $P = P_1[0] \mid P_2[0]$ and $Q = \mathtt{new}\ k.[x_1 := k].[x_2 := k].(Q_1 \mid Q_2)$ for some fresh name k, and assume that:*

1. *P/Φ and Q/Ψ are composable under $\mathcal{E}_0$ and C;*
2. *$(\mathcal{E}_0; C[Q]; \Psi)$ does not reveal k, $\mathsf{pk}(k)$, $\mathsf{vk}(k)$;*
3. *$(\mathcal{E}_0; C[P]; \Phi)$ satisfies the abstractability property; and*
4. *$P_1/P_2/\Phi$ is a good key-exchange protocol under $\mathcal{E}_0$ and C.*

Let $P^+ = P_1[\mathtt{out}(d, x_1)] \mid P_2[\mathtt{out}(d, x_2)] \mid \mathtt{in}(d, x).\mathtt{in}(d, y).\mathtt{if}\ x = y\ \mathtt{then}\ 0\ \mathtt{else}\ 0$. If the biprocesses $(\mathcal{E}_0; \mathtt{new}\ d.C[P^+]; \Phi)$ and $(\mathcal{E}_0; C[Q]; \Psi)$ satisfy diff-equivalence then $(\mathcal{E}_0; C[P_1[Q_1] \mid P_2[Q_2]]; \Phi \uplus \Psi)$ satisfies diff-equivalence.

We require $(\mathcal{E}_0; \mathtt{new}\ d.C[P^+]; \Phi)$ to be in diff-equivalence (and not simply $(\mathcal{E}_0; C[P]; \Phi)$). This ensures that the same equalities between values of assignment variables hold on both sides of the equivalence. Actually, when the composition context C under study is not of the form $C'[!\_]$, and under the hypothesis that $P_1/P_2/\Phi$ is a good key-exchange protocol under $\mathcal{E}_0$ and C, we have that these two requirements coincide. However, the stronger hypothesis is important to conclude when C is of the form $C'[!\_]$. Indeed, in this case, we do not know in advance what are the instances of P_1 and P_2 that will be "matched". This is not a problem but to conclude about the diff-equivalence of the whole process (*i.e.* $(\mathcal{E}_0; C[P_1[Q_1] \mid P_2[Q_2]]; \Phi \uplus \Psi)$), we need to ensure that such a matching is the same on both sides of the equivalence. Note that to conclude about trace equivalence only, this additional requirement is actually not necessary.

7 Case studies

Many applications rely on several protocols running in composition (parallel, sequential, or nested). In this section, we show that our results can help in the analysis of this sort of complex system. Our main goal is to show that the extra hypotheses needed to analyse an application in a moduar way are reasonnable.

7.1 3G mobile phones

We look at confidentiality and privacy guarantees provided by the *AKA* protocol and the Submit SMS procedure (*sSMS*) when run in composition as specified by the 3GPP consortium in [2].

Protocols description. The *sSMS* protocol allows a mobile station (MS) to send an SMS to another MS through a serving network (SN). The confidentiality of the sent SMS relies on a session key *ck* established through the execution of the *AKA* protocol between the MS and the SN. The *AKA* protocol achieves mutual authentication between a MS and a SN, and allows them to establish a shared session key *ck*. The *AKA* protocol consists in the exchange of two messages: the *authentication request* and the *authentication response.* The *AKA* protocol as deployed in real 3G telecommunication systems presents a linkability attack [5], and thus we consider here its fixed version as described in [5]. At the end of a successful execution of this protocol, both parties should agree on a fresh ciphering key *ck*. This situation can be modelled in our calculus as follows:

$$\texttt{new}\ sk_{SN}.\ !\texttt{new}\ IMSI.\ \texttt{new}\ k_{IMSI}.\ !\texttt{new}\ sqn.\ \texttt{new}\ sms.\\ (AKA^{SN}[sSMS^{SN}] \mid AKA^{MS}[sSMS^{MS}])$$

where sk_{SN} represents the private key of the network; while *IMSI* and k_{IMSI} represent respectively the long-term identity and the symmetric key of the MS. The name *sqn* models the sequence number on which SN and MS are synchronised. The two subprocesses AKA^{MS} and $sSMS^{MS}$ (*resp.* AKA^{SN}, and $sSMS^{SN}$) model one session of the MS's (*resp.* SN's) side of the *AKA*, and *sSMS* protocols respectively. Each MS, identified by its identity *IMSI* and its key k_{IMSI}, can run multiple times the *AKA* protocol followed by the *sSMS* protocol.

Security analysis. We explain how some confidentiality and privacy properties of the *AKA* protocol and the *sSMS* procedure can be derived relying on our composition results. We do not need to tag the protocols under study to perform our analysis since they do not share any primitive but the pairing operator. Note that the *AKA* protocol can *not* be modelled in the calculus given in [12] due to the need of non-trivial else branches. Moreover, to enable the use of ProVerif, we had to abstract some details of the considered protocols that ProVerif cannot handle. In particular, we model timestamps using nonces, we replace the use of the xor operation by symmetric encryption, and we assume that the two parties are "magically" synchronised on their counter value.

Strong unlinkability requires that an observer does not see the difference between the two following scenarios: *(i)* a same mobile phone sends several SMSs; or *(ii)* multiple mobile phones send at most one SMS each. To model this requirement, we consider the composition context[5]:

$$\begin{aligned} C_U[\_] \overset{\mathsf{def}}{=} {} & !\texttt{new}\ IMSI_1.\ \texttt{new}\ k_{IMSI_1}.\ !\texttt{new}\ IMSI_2.\ \texttt{new}\ k_{IMSI_2}.\\ & \texttt{let}\ IMSI = \mathsf{diff}[IMSI_1, IMSI_2]\ \texttt{in}\ \texttt{let}\ k_{IMSI} = \mathsf{diff}[k_{IMSI_1}, k_{IMSI_2}]\ \texttt{in}\\ & \texttt{new}\ sqn.\ \texttt{new}\ sms.\ \_ \end{aligned}$$

To check if the considered 3G protocols satisfy strong unlinkability, one needs to check if the following biprocess satisfies diff-equivalence ($\Phi_0 = \{w_1 \rhd \mathsf{pk}(sk_{SN})\}$):

$$(sk_{SN}; C_U[AKA^{SN}[sSMS^{SN}] \mid AKA^{MS}[sSMS^{MS}]]; \Phi_0)$$

[5] We use $\texttt{let}\ x = M\ \texttt{in}\ P$ to denote the process $P\{M/x\}$.

Hypotheses (1-4) stated in Theorem 4 are satisfied, and thus this equivalence can be derived from the following two "smaller" diff-equivalences:

$$(sk_{SN}; \mathtt{new}\ d.\ C_U[AKA^+]; \Phi_0) \quad \text{and} \quad (sk_{SN}; C'_U[sSMS]; \emptyset)$$

- $sSMS \overset{\mathsf{def}}{=} sSMS^{SN} \mid sSMS^{MS}$,
- $AKA^+ \overset{\mathsf{def}}{=} AKA^{SN}[\mathtt{out}(d, xck_{SN})] \mid AKA^{MS}[\mathtt{out}(d, xck_{MS})] \mid$
  $\mathtt{in}(d,x).\ \mathtt{in}(d,y).\ \mathtt{if}\ x = y\ \mathtt{then}\ 0\ \mathtt{else}\ 0$
- $C'_U[\_] \overset{\mathsf{def}}{=} C_U[\mathtt{new}\ ck.\mathtt{let}\ xck_{SN} = ck\ \mathtt{in}\ \mathtt{let}\ xck_{MS} = ck\ \mathtt{in}\ \_]$.

Weak secrecy requires that the sent/received SMS is not deducible by an outsider, and can be modelled using the context

$$C_{WS}[\_] \overset{\mathsf{def}}{=} !\mathtt{new}\ IMSI.\ \mathtt{new}\ k_{IMSI}.\ !\mathtt{new}\ sqn.\mathtt{new}\ sms.\_.$$

The composition context C_{WS} is the same as $\mathsf{fst}(C_U)$ (up to some renaming), thus Hypotheses (1-4) of Theorem 2 also hold and we derive the weak secrecy property by simply analysing this property on AKA and $sSMS$ in isolation.

Strong secrecy means that an outsider should not be able to distinguish the situation where sms_1 is sent (resp. received), from the situation where sms_2 is sent (resp. received), although he might know the content of sms_1 and sms_2. This can be modelled using the following composition context:

$$C_{SS}[\_] \overset{\mathsf{def}}{=} !\mathtt{new}\ IMSI.\ \mathtt{new}\ k_{IMSI}.\ !\mathtt{new}\ sqn.\ \mathtt{let}\ sms = \mathsf{diff}[sms_1, sms_2]\ \mathtt{in}\ \_$$

where sms_1 and sms_2 are two free names known to the attacker. Again, our Theorem 4 allows us to reason about this property in a modular way.

Under the abstractions briefly explained above, all the hypotheses have been checked using ProVerif. Actually, it happens that ProVerif is also able to conclude on the orignal protocol (the one without decomposition) for the three security properties mentioned above. Note that a less abstract model of the same protocol (*e.g.* the one with the xor operator) would have required us to rely on a manual proof. In such a situation, our composition result allows us to reduce a big equivalence that existing tools cannot handle, to a much smaller one which is a more manageable work in case the proof has to be done manually.

7.2 E-passport application

We look at privacy guarantees provided by three protocols of the e-passport application when run in composition as specified in [1].

Protocols description. The information stored in the chip of the passport is organised in data groups (dg_1 to dg_{19}): dg_5 contains a JPEG copy of the displayed picture, dg_7 contains the displayed signature, whereas the verification key $\mathsf{vk}(sk_P)$ of the passport, together with its certificate $\mathsf{sign}(\mathsf{vk}(sk_P), sk_{DS})$ issued by the Document Signer authority are stored in dg_{15}. For authentication purposes, a hash of all the dgs together with a signature on this hash value are stored in a separate file, the Security Object Document:

$$sod \overset{\mathsf{def}}{=} \langle \mathsf{sign}(\mathsf{h}(dg_1, \ldots, dg_{19}), sk_{DS}),\ \mathsf{h}(dg_1, \ldots, dg_{19}) \rangle.$$

The ICAO standard specifies several protocols through which this information can be accessed [1]. First, the *Basic Access Control* (*BAC*) protocol establishes a key seed *kseed* from which a session key *kenc* is derived. The purpose of *kenc* is to prevent skimming and eavesdropping on the subsequent communication with the e-passport. The security of the *BAC* protocol relies on two master keys, *ke* and *km*. Once the *BAC* protocol has been successfully executed, the reader gains access to the information stored in the RFID tag through the *Passive Authentication* (*PA*) and the *Active Authentication* (*AA*) protocols that can be executed in any order. This situation can be modelled in our calculus:

$$P \overset{\mathsf{def}}{=} \mathtt{new}\ sk_{DS}.\ !\mathtt{new}\ ke.\ \mathtt{new}\ km.\ \mathtt{new}\ sk_P.\mathtt{new}\ id.\ \mathtt{new}\ sig.\ \mathtt{new}\ pic.\ \ldots$$
$$!(BAC^R[PA^R \mid AA^R] \mid BAC^P[PA^P \mid AA^P])$$

where id, sig, pic, ... represent the name, the signature, the displayed picture, *etc* of the e-passport's owner, *i.e.* the data stored in the dgs (1-14) and (16-19). The subprocesses BAC^P, PA^P and AA^P (*resp.* BAC^R, PA^R and AA^R) model one session of the passport's (*resp.* reader's) side of the *BAC*, *PA* and *AA* protocols respectively. The name sk_{DS} models the signing key of the Document Signing authority used in all passports. Each passport (identified by its master keys *ke* and *km*, its signing key sk_P, the owner's name, picture, signature, ...) can run multiple times the *BAC* protocol followed by the *PA* and *AA* protocols.

Security analysis. We explain below how *strong anonymity* of these three protocols executed together can be derived from the analysis performed on each protocol in isolation. In [4], as sequential composition could not be handled, the analysis of the e-passports application had to exclude the execution of the *BAC* protocol. Instead, it was assumed that the key *kenc* is "magically" pre-shared between the passport and the reader. Thanks to our Theorem 4, we are now able to complete the analysis of the e-passport application.

To express strong anonymity, we need on the one hand to consider a system in which the particular e-passport with publicly known id_1, sig_1, pic_1, *etc.* is being executed, while on the other hand it is a different e-passport with publicly known id_2, sig_2, pic_2, *etc.* which is being executed. We consider the context:

$$C_A[\_] \overset{\mathsf{def}}{=} !\mathtt{new}\ ke.\ \mathtt{new}\ km.\ \mathtt{new}\ sk_P.\mathtt{let}\ id = \mathsf{diff}[id_1, id_2]\ \mathtt{in}\ \ \ldots !\ \_$$

This composition context differs in the e-passport being executed on the left-hand process and on the right-hand process. In other words, the system satisfies anonymity if an observer cannot distinguish the situation where the e-passport with publicly known id_1, sig_1, pic_1, *etc.* is being executed, from the situation where it is another e-passport which is being executed. To check if the tagged version of the e-passport application (we assume here that *BAC*, *PA*, and *AA* are tagged in different ways) preserves strong anonymity, one thus needs to check if the following biprocess satisfies diff-equivalence (with $\Phi_0 = \{w_1 \rhd \mathsf{vk}(sk_{DS})\}$):

$$(sk_{DS}; C_A[BAC^R[PA^R \mid AA^R] \mid BAC^P[PA^P \mid AA^P]]; \Phi_0)$$

We can instead check whether *BAC*, *PA* and *AA* satisfy anonymity in isolation, *i.e.* if the following three diff-equivalences hold:

$$(sk_{DS}; \mathtt{new}\ d.\ C_A[BAC^{+}]; \emptyset)\ (\alpha) \qquad \begin{array}{c}(sk_{DS}; C'_A[PA^R \mid PA^P]; \Phi_0)\ (\beta)\\ (sk_{DS}; C'_A[AA^R \mid AA^P]; \emptyset)\ (\gamma)\end{array}$$

where

- $BAC^{+} \overset{\mathsf{def}}{=} BAC^R[\mathtt{out}(d, xkenc_R)] \mid BAC^P[\mathtt{out}(d, xkenc_P)]$
  $\mid \mathtt{in}(d,x).\ \mathtt{in}(d,y).\ \mathtt{if}\ x = y\ \mathtt{then}\ 0\ \mathtt{else}\ 0;$
- $C'_A[\_] \overset{\mathsf{def}}{=} C_A[C''_A[\_]]$; and
- $C''_A[\_] \overset{\mathsf{def}}{=} \mathtt{new}\ kenc.\ \mathtt{let}\ xkenc_R = kenc\ \mathtt{in}\ \mathtt{let}\ xkenc_P = kenc\ \mathtt{in}\ \_.$

Then, applying Theorem 3 to (β) and (γ) we derive that the following biprocess satisfies diff-equivalence:

$$(sk_{DS}; C'_A[PA^R \mid AA^R \mid PA^P \mid AA^P]; \Phi_0) \quad (\delta).$$

and applying Theorem 4 to (α) and (δ), we derive the required diff-equivalence:

$$(sk_{DS}; C_A[BAC^R[PA^R \mid AA^R] \mid BAC^P[PA^P \mid AA^P]]; \Phi_0)$$

Note that we can do so because Hypotheses (1-4) stated in Theorem 4 are satisfied, and in particular because $BAC^R/BAC^P/\emptyset$ is a good key-exchange protocol under $\{sk_{DS}\}$ and C_A. Again, all the hypotheses have been checked using ProVerif. Actually, it happens that ProVerif is also able to directly conclude on the whole system.

Unfortunately, our approach does not apply to perform a modular analysis of strong unlinkability. The BAC protocol does not satisfy the diff-equivalence needed to express such a security property, and this hypothesis is mandatory to apply our composition result.

8 Conclusion

We investigate composition results for reachability properties as well as privacy-type properties expressed using a notion of equivalence. Relying on a generic composition result, we derive parallel composition results, and we study the particular case of key-exchange protocols under various composition contexts.

All these results work in a quite general setting, *e.g.* processes may have non trivial else branches, we consider arbitrary primitives expressed using an equational theory, and processes may even share some standard primitives as long as they are tagged in different ways. We illustrate the usefulness of our results through the mobile phone and e-passport applications.

We believe that our generic result could be used to derive further composition results. We may want for instance to relax the notion of being a *good protocol* at the price of studying a less ideal scenario when analysing the protocol Q in isolation. We may also want to consider situations where sub-protocols sharing some data are arbitrarily interleaved. Moreover, even if we consider arbitrary primitives, sub-protocols can only share some standard primitives provided that they are tagged. It would be nice to relax these conditions. This would allow one to compose protocols (and not their tagged versions) or to compose protocols that both rely on primitives for which no tagging scheme actually exists (*e.g.* exclusive-or).

References

1. PKI for machine readable travel documents offering ICC read-only access. Technical report, International Civil Aviation Organization, 2004.
2. 3GPP. Technical specification group services and system aspects; 3G security; security architecture (release 9). Technical report, 3rd Generation Partnership Project, 2010.
3. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, 2001.
4. M. Arapinis, V. Cheval, and S. Delaune. Verifying privacy-type properties in a modular way. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF'12)*, 2012.
5. M. Arapinis, L. I. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *ACM Conference on Computer and Communications Security*, 2012.
6. A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, 2008.
7. A. Armando et al. The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. In *Proc. 18th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2012.
8. B. Barak, R. Canetti, J. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *Proc. 45th Symposium on Foundations of Computer Science (FOCS'04)*, 2004.
9. B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2008.
10. F. Böhl and D. Unruh. Symbolic universal composability. In *Proc. 26th Computer Security Foundations Symposium (CSF'13)*, 2013.
11. M. Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, 2010.
12. Ş. Ciobâcă and V. Cortier. Protocol composition for arbitrary primitives. In *Proc. of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, 2010.
13. V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, Feb. 2009.
14. T. Groß and S. Mödersheim. Vertical protocol composition. In *Proc. 24th Computer Security Foundations Symposium, (CSF'11)*, 2011.
15. J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proc. 13th Computer Security Foundations Workshop (CSFW'00)*, 2000.
16. R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. In *Proc. 18th Conference on Computer and Communications Security (CCS'11)*, 2011.
17. S. Mödersheim and L. Viganò. Secure pseudonymous channels. In *Proc. 14th European Symposium on Research in Computer Security (ESORICS'09)*, 2009.
18. A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, 2010.