Verifying privacy-type properties in a modular way

Myrto Arapinis *, Vincent Cheval † and Stéphanie Delaune†
*School of Computer Science, University of Birmingham, UK
†LSV, ENS Cachan & CNRS & INRIA Saclay Île-de-France

***Abstract*—Formal methods have proved their usefulness for analysing the security of protocols. In this setting, privacy-type security properties (*e.g.* vote-privacy, anonymity, unlinkability) that play an important role in many modern applications are formalised using a notion of equivalence.**

In this paper, we study the notion of trace equivalence and we show how to establish such an equivalence relation in a modular way. It is well-known that composition works well when the processes do not share secrets. However, there is no result allowing us to compose processes that rely on some shared secrets such as long term keys. We show that composition works even when the processes share secrets provided that they satisfy some reasonable conditions. Our composition result allows us to prove various equivalence-based properties in a modular way, and works in a quite general setting. In particular, we consider arbitrary cryptographic primitives and processes that use non-trivial else branches.

As an example, we consider the ICAO e-passport standard, and we show how the privacy guarantees of the whole application can be derived from the privacy guarantees of its sub-protocols.

***Keywords*-Anonymity and privacy; Formal methods; Security procotols; Trace equivalence; Parallel composition**

I. Introduction

With the emergence of new systems and services like electronic IDs and passports, electronic payment systems and loyalty schemes, electronic tickets like the Navigo pass in Paris or the Oyster card in London, or telecommunication systems like mobile phones, new privacy and security concerns arise. Indeed, governments, financial and transport organisations, or telecommunication companies, all possess and manage important amounts of information concerning all of our everyday activities. As often reported by the media [1]–[3], this exposes us to a number of privacy threats. Security mechanisms should thus secure the offered services, ensuring the confidentiality of the gathered data and enhancing the privacy of users' identity and behaviour.

To this effect, many cryptographic protocols have been designed to prevent third parties from identifying messages as coming from a particular user. For example, mobile phone operators identify mobile phones using temporary identities that are periodically and securely updated to prevent mobile phones from being traceable. The electronic passports also include mechanisms that do not let the passport's chip disclose private information to external users. However, the design of protocols that meet particular security requirements is a notoriously difficult and error prone task. Indeed, numerous deployed protocols have subsequently been found to be flawed. For example, the BAC protocol of electronic passports makes it possible to recognise a previously observed passport, potentially enabling tracking passport holders [4], [5].

In this context, formal methods have proved their usefulness for precisely analysing the security guarantees provided by a protocol. Several techniques have been developed for [6], [7], and successfully applied to the analysis of cryptographic protocols [5], [8]. For example, a flaw has been discovered (see [9]) in the Single-Sign-On protocol used *e.g.* by Google Apps. It has been shown that a malicious application could very easily access any other application (*e.g.* Gmail or Google Calendar) of their users. This flaw has been found when analysing the protocol using formal methods, abstracting messages by a term algebra and using the AVISPA platform [10]. However, existing techniques for analysing protocols with respect to privacy-type properties (*e.g.* [7], [11]), consider protocols to be executed in isolation, *i.e.* without taking into account other protocols which may be running in parallel. But in reality many applications run in parallel and the underlying protocols may interact in unexpected ways if cryptographic material is shared amongst them. This situation can arise if, for example, a user chooses the same password for two different network services, or a server uses the same key for different protocols.

Furthermore, real life protocols are usually complex and composed of several sub-protocols that rely on the same cryptographic material. For example, the UMTS standard [12]–[14] specifies tens of sub-protocols running in parallel in 3G mobile phone systems. And, while one may hope to automatically verify each of these sub-protocols in isolation, it is unrealistic to expect that the whole suite of protocols can be automatically checked. Indeed, due to computational constraints, existing tools and techniques do not scale up well to such large systems, and it is often the case that the sub-components have to be considered and analysed independently.

Unfortunately, security proofs of network services or protocols considered in isolation, do not carry over when they share keys or passwords. Consider for example the two naive protocols:

$$P: \ A \to S : \{A\}^r_{\mathsf{pk}(S)} \qquad Q: \ \begin{array}{l} A \to S : \{N_a\}^r_{\mathsf{pk}(S)} \\ S \to A : N_a \end{array}$$

In protocol P, the agent A simply identifies himself to the server S by sending him his identity encrypted under S's public key (using a probabilistic encryption scheme). In protocol Q, the agent sends some fresh nonce N_a encrypted under S's public key. The server S acknowledges A's message by forwarding A's nonce. While P executed alone guarantees A's anonymity, it is not the case when the protocol Q is run in parallel. Indeed, an adversary may use Q as an oracle to decrypt any message. More realistic examples illustrating interactions between protocols can be found in *e.g.* [15].

In order to enable verification of complex real life systems, composition theorems for modular reasoning about security and privacy are therefore desirable. They may allow one to deduce security guarantees for a complex protocol, from the security guarantees of the individual sub-protocols. The goal of our paper is to study the composition of protocols with respect to privacy-type properties.

Related work: There are a number of papers studying the secure composition of security protocols in the symbolic model (*e.g.* [16], [17]) and in the computational model (*e.g.* [18], [19]). Our result clearly belongs to the first approach.

Actually, many results have been established for trace-based security property, *e.g.* [16], [20], [21]. A result closely related to ours is the one of S. Ciobaca and V. Cortier [17]. Their result holds for any cryptographic primitives that can be modelled using equational theories, and their main result transforms any attack trace of the combined protocol into an attack trace of one of the individual protocols. This allows various ways of combining protocols such as sequentially or in parallel, possibly with inner replications. However, the major difference with our result is that they consider trace-based security properties, and more precisely secrecy (encoded as a reachability property).

Regarding equivalence-based properties, it has been shown that composition works for resistance against guessing attacks in the passive case without any additional hypothesis [22], and in the active case when the protocols are tagged [22], [23]. However, these composition results assume that passwords are the only shared secrets and are not well-suited to analyse privacy-type properties such as anonymity and unlinkability.

Our work is also related to those of Canetti *et al.* who, in the context of computational models, study *universal composability of protocols* [18]. This approach consists of defining for each sub-protocol an *ideal functionality* and then showing that a certain implementation securely emulates the ideal functionality. Since this initial work, the universal composability framework has been improved in several ways, *e.g.* with joint states [24], without pre-established session identifiers [19].

Our contributions: While most existing papers studying compositionality of protocols consider trace-based properties (covering confidentiality and authentication requirements), our work tackles the compositionality problem with respect to privacy-type properties which are usually expressed as equivalences between processes. Roughly, two processes P and Q are equivalent ($P \approx Q$) if no process O can observe any difference between the processes P and Q.

We identify sufficient conditions of *disjointness* under which protocols can "safely" be executed in parallel. In particular, we require protocols run in parallel not to use the same primitives. Our theorems hold for arbitrary primitives that can be modelled by a set of equations, and can thus handle composition of protocols relying on symmetric and asymmetric encryption schemes, hash functions, signatures, zero knowledge proofs, message authentication codes, designated verifier proofs, exclusive or, *etc*.

We first state a composition result that also allows the protocols considered to share the usual cryptographic primitives of symmetric and asymmetric encryption, hashing, and signing, provided that these primitives are tagged and that public and verification keys are not derivable. In this setting, we are able to establish a strong result that basically says that the disjoint scenario is equivalent to the shared one. This allows us to go back to the disjoint case (with no shared keys) for which composition works unsurprisingly well.

Then, we further relax this condition. A second theorem shows that it is possible to compose protocols that share public and verification keys even if those are known by the attacker, provided that they are given to him from the beginning. However, in our setting such a sequence has to be finite, and thus our result can only be applied in presence of a bounded number of public shared keys. This is not a real limitation for the analysis of the e-passport application, but this could lead us to an unrealisitic situation for some other applications.

In both cases, we show that whenever processes P and Q (*resp.* P' and Q') satisfy the corresponding disjointness property, we can derive that P and Q running in parallel under the *composition context* $C[\_]$ are equivalent to P' and Q' running in parallel under the *composition context* $C'[\_]$, *i.e.*

$$C[P \mid Q] \;\approx\; C'[P' \mid Q']$$

from the equivalences $C[P] \approx C'[P']$ and $C[Q] \approx C'[Q']$. The composition context under which two processes are composed contains the shared keys possibly under some replications.

We illustrate the application of our results on a case study. We consider the protocols specified in the e-passport application [13], and show how the privacy guarantees of the whole application can be derived from the privacy guarantees of the individual e-passport protocols. However, due to the limitations of our composition results, our analysis is performed on the tagged version of the protocols. Moreover,

in its current form, our results do not allow us to deal with sequential composition, thus the *BAC* protocol used in the e-passport application for key establishment is modeled in an abstract way. We consider that the keys are "securely" pre-shared.

Due to lack of space, proofs are omitted, but they can be found in [25].

II. Models for security protocols

In this section, we introduce the cryptographic process calculus that we will use for describing protocols. This calculus is close to the applied pi calculus as defined in [26]. However, we use a slightly different syntax and we give a non-compositional semantics that is easier to manipulate than its compositional counterpart as defined in [26].

A. Messages

A protocol consists of some agents communicating on a network. The messages sent by the agents are modelled using an abstract term algebra. For this, we assume an infinite set of *names* $\mathcal{N}$ which is split into the set $\mathcal{B} = \{a, b, k, n, \ldots\}$ of names of *base type* (which are used for representing keys, nonces, ...) and the set $\mathcal{C}h = \{c, c_1, ch, ch_1, \ldots\}$ of names of *channel type* (which are used to name communication channels). We also consider a set of *variables* $\mathcal{X} = \{x, y, \ldots\}$, 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 differ from the channel type. Moreover, we consider in addition the type *seed*. This is a subsort of the base type, and we will assume that this set only contains atomic data, *i.e.* variables and names. As in the applied pi calculus, 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. Let $\mathsf{N} \subseteq \mathcal{N}$ and $\mathsf{X} \subseteq \mathcal{X}$, the set of terms built from N and X by applying function symbols in Σ is denoted by $\mathcal{T}(\Sigma, \mathsf{N} \cup \mathsf{X})$. Of course function symbol application must respect sorts and arities. We write $fv(u)$ (resp. $fn(u)$) for the set of variables (resp. names) occurring in a term u. A term is *ground* if it does not contain any variable.

To model algebraic properties of cryptographic primitives, we define an *equational theory* 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 define $=_{\mathsf{E}}$ to be 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 following signature Σ_0:

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

The function symbols sdec, senc (resp. adec and aenc) of arity 2 represent symmetric (resp. asymmetric) decryption and encryption. Pairing is modelled using a symbol of arity 2, denoted $\langle\,\rangle$, and projection functions denoted proj_1 and proj_2. We consider also signatures and hashes. We denote by $\mathsf{pk}(sk)$ (resp. $\mathsf{vk}(sk)$) the public key (resp. the verification key) associated to the private key sk. Moreover, we consider that the function symbols pk and vk take as argument a term of type *seed*.

Then, we consider the equational theory E_0, defined by the following equations ($i \in \{1, 2\}$):

$$\mathsf{sdec}(\mathsf{senc}(x, y), y) = x \quad \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), y) = x$$
$$\mathsf{proj}_i(\langle x_1, x_2\rangle) = x_i \quad \mathsf{check}(\mathsf{sign}(x, y), \mathsf{vk}(y)) = x$$

To make a message m extractable from a signature $\mathsf{sign}(m, k)$, typically one would just attach the message m to the signature using the pairing function symbol, *i.e.* $\langle m, \mathsf{sign}(m, k)\rangle$. Then, such a signature can be checked using the operator check when the verification key is known.

Let $u_1 = \mathsf{senc}(\mathsf{proj}_2(\langle a, b\rangle), k)$ and $u_2 = \mathsf{senc}(b, k)$. We have that the terms u_1 and u_2 are equal modulo E_0, written $u_1 =_{\mathsf{E}_0} u_2$, while obviously the syntactic equality $u_1 = u_2$ does not hold.

B. Processes

Plain processes are built up in a similar way to plain processes in applied pi calculus. The grammar of the *plain processes* is as follows:

$$\begin{aligned} P, Q := \;& 0 \\ & P \mid Q \\ & \mathtt{new}\; n.P \\ & !P \\ & \mathtt{if}\; u_1 = u_2 \;\mathtt{then}\; P \;\mathtt{else}\; Q \\ & \mathtt{in}(u, x).P \\ & \mathtt{out}(u, v).Q \end{aligned}$$

where u is a term of channel type (*i.e.* a name or a variable), u_1, u_2 are terms having the same type, x is a variable, v is a term, and n is a name. The terms u_1, u_2 and v may contain variables.

As usual, names and variables have scopes, which are delimited by restrictions and by inputs. 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 respectively.

Extended processes add a set of restricted names $\mathcal{E}$, and a sequence of messages Φ.

Definition 1: An *extended process* A is a triple $(\mathcal{E}; \mathcal{P}; \Phi)$:

- $\mathcal{E}$ is a set of names that represents the names that are restricted in $\mathcal{P}$ and Φ;
- $\mathcal{P}$ is a multiset of *plain processes* where null processes are removed and such that $fv(\mathcal{P}) = \emptyset$;
- $\Phi = \{w_1 \rhd u_1, \ldots, w_n \rhd u_n\}$ where $u_1, \ldots, u_n$ are ground terms, and $w_1, \ldots, w_n$ are variables.

We write $\mathrm{dom}(\Phi)$ the domain of Φ, *i.e.* $\mathrm{dom}(\Phi) = \{w_1, \ldots, w_n\}$. We write $fn(A)$ and $bn(A)$ for the sets of free and bound names of an extended process A. Given $A = (\mathcal{E}; \mathcal{P}; \Phi)$, we have that $fn(A) = fn(\mathcal{P}) \setminus \mathcal{E}$, and $bn(A) = bn(\mathcal{P}) \cup \mathcal{E}$.

For the sake of clarity, we often omit brackets and the null process. For instance, we write k_1, $\mathtt{out}(c, u)$ instead of $\{k_1\}$ and $\{\mathtt{out}(c, u).0\}$. When there is no "$\mathtt{else}$", it means "$\mathtt{else}\ 0$"; and we sometimes write

$$\mathtt{if}\ (u_1 = u_2 \wedge u_1' = u_2')\ \mathtt{then}\ P\, \mathtt{else}\ Q$$

instead of nested conditionals. Moreover, we often write P instead of $(\emptyset; P; \emptyset)$.

Example 2: As an illustrative example, consider the process $A_i = \mathtt{new}\ sk_S.(P_i \mid Q)$ that has been informally introduced in Section I. We have that:

- $P_i = \mathtt{new}\ r.\mathtt{out}(c, \mathsf{aenc}(\langle r, id_i\rangle, \mathsf{pk}(sk_S)))$, and
- $Q = \mathtt{in}(c, x).\mathtt{out}(c, \mathsf{proj}_2(\mathsf{adec}(x, sk_S)))$.

The first component generates a fresh random number r, and publishes the message $\mathsf{aenc}(\langle r, id_i\rangle, \mathsf{pk}(sk_S))$ containing its identity id_i by sending it on the public channel c. The second component receives a message on c, uses the private key sk_S to decrypt it, and sends the second part of the resulting plaintext on c.

The semantics is given by a set of labelled rules (see Figure 1) that allows one to reason about processes that interact with their environment. This defines the relation $\xrightarrow{\ell}$ where ℓ is either an input, an output, or a silent action τ. Note that the sent messages of base type are exclusively stored in the frame and not in the labels (the outputs are made by "reference").

Example 3: Let A_i be the extended process defined in Example 2. We have that:

$$\begin{array}{l} A_i \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau} \\ \xrightarrow{\nu w_1.\mathtt{out}(c,w_1)} (\{sk_S, r\}; Q; w_1 \rhd \mathsf{aenc}(\langle r, id_i\rangle, \mathsf{pk}(sk_S))) \\ \xrightarrow{\mathtt{in}(c,w_1)} \\ \qquad (\{sk_S, r\}; \mathtt{out}(c, M_i); w_1 \rhd \mathsf{aenc}(\langle r, id_i\rangle, \mathsf{pk}(sk_S))) \\ \xrightarrow{\nu w_2.\mathtt{out}(c,w_2)} (\{sk_S, r\}; 0; \Phi_i) \stackrel{\mathsf{def}}{=} A_i' \end{array}$$

with $M_i = \mathsf{proj}_2(\mathsf{adec}(\mathsf{aenc}(\langle r, id_i\rangle, \mathsf{pk}(sk_S)), sk_S))$ and $\Phi_i = \{w_1 \rhd \mathsf{aenc}(\langle r, id_i\rangle, \mathsf{pk}(sk_s)),\ w_2 \rhd M_i\}$. Note that $M_i =_{\mathsf{E}_0} id_i$.

The three first steps are performed using the rules NEW and PAR. Then, we used the rules OUT-T and IN. We denote by A_i' the resulting extended process.

Notations: Let $\mathcal{A}$ be the alphabet of actions (in our case this alphabet is infinite and contains the special symbol τ). For every $w \in \mathcal{A}^*$, the relation $\xrightarrow{w}$ on processes is defined in the usual way. For $s \in (\mathcal{A} \setminus \{\tau\})^*$, the relation $\stackrel{s}{\Rightarrow}$ on processes is defined by: $A \stackrel{s}{\Rightarrow} B$ if, and only if there exists $w \in \mathcal{A}^*$ such that $A \xrightarrow{w} B$ and s is obtained by erasing all occurrences of τ.

III. Formalising privacy-type security properties

Many interesting security properties, in particular privacy-type properties such as those studied in [5], [27], [28], are formalised using behavioural equivalence. We will review some of them in Section III-B using the notion of trace equivalence.

A. Trace equivalence

Before defining trace equivalence, we introduce the notion of *static equivalence* that compares sequences of messages, a notion of intruder's knowledge that has been extensively studied (*e.g.* [29]).

To represent the knowledge of an attacker (who may have observed a sequence of messages $u_1, \ldots, u_n$), we use the concept of frame. A frame $\phi = \mathtt{new}\ \mathcal{E}.\Phi$ consists of a finite set $\mathcal{E}$ of restricted names (those initially unknown to the attacker), and a substitution Φ of the form:

$$\{w_1 \rhd u_1, \ldots, w_n \rhd u_n\} \text{ with } \mathrm{dom}(\Phi) = \{w_1, \ldots, w_n\}$$

The variables enable us to refer to each u_i and we always assume that the terms u_i are ground. The names $\mathcal{E}$ are bound in ϕ and can be renamed. Moreover names that do not appear in Φ can be added or removed from $\mathcal{E}$. In particular, we can always assume that two frames share the same set of restricted names.

Two frames are considered equivalent when the attacker cannot detect the difference between the two situations they represent, that is, his ability to distinguish whether two recipes M and N produce the same term does not depend on the frame.

Definition 2: We say that two frames $\phi_1 = \mathtt{new}\,\mathcal{E}.\Phi_1$ and $\phi_2 = \mathtt{new}\,\mathcal{E}.\Phi_2$ are *statically equivalent*, $\phi_1 \sim \phi_2$, when $\mathrm{dom}(\Phi_1) = \mathrm{dom}(\Phi_2)$, and for all terms M, N such that $fn(M, N) \cap \mathcal{E} = \emptyset$, we have that $M\Phi_1 =_{\mathsf{E}} N\Phi_1$, if and only if, $M\Phi_2 =_{\mathsf{E}} N\Phi_2$.

Example 4: Let A_1' (resp. A_2') be the extended process described in Example 3 and ϕ_1 (resp. ϕ_2) be its associated frame, *i.e.* $\phi_i = \mathtt{new}\ \{sk_S, r\}.\Phi_i$ with $i \in \{1, 2\}$. We have that $\phi_1 \not\sim \phi_2$. Indeed, the test $w_2 \stackrel{?}{=} id_1$ can be used to distinguish the two frames. The test holds in ϕ_1 since $w_2\Phi_1 = M_1 =_{\mathsf{E}_0} id_1$, whereas it does not hold in ϕ_2 since $w_2\Phi_2 = M_2 =_{\mathsf{E}_0} id_2 \neq_{\mathsf{E}_0} id_1$. However, we have that:

$$\begin{array}{c} \mathtt{new}\ \{sk_S, r\}.\{w_1 \rhd \mathsf{aenc}(\langle r, id_1\rangle, \mathsf{pk}(sk_S))\} \\ \sim \\ \mathtt{new}\ \{sk_S, r\}.\{w_1 \rhd \mathsf{aenc}(\langle r, id_2\rangle, \mathsf{pk}(sk_S))\}. \end{array}$$

$$(\mathcal{E}; \{\texttt{if } u = v \texttt{ then } Q_1 \texttt{ else } Q_2\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau} (\mathcal{E}; Q_1 \uplus \mathcal{P}; \Phi) \quad \text{if } u =_{\mathsf{E}} v \qquad \text{(THEN)}$$

$$(\mathcal{E}; \{\texttt{if } u = v \texttt{ then } Q_1 \texttt{ else } Q_2\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau} (\mathcal{E}; Q_2 \uplus \mathcal{P}; \Phi) \quad \text{if } u \neq_{\mathsf{E}} v \qquad \text{(ELSE)}$$

$$(\mathcal{E}; \{\texttt{out}(p,u).Q_1; \texttt{in}(p,x).Q_2\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau} (\mathcal{E}; Q_1 \uplus Q_2\{x \mapsto u\} \uplus \mathcal{P}; \Phi) \qquad \text{(COMM)}$$

$$(\mathcal{E}; \{\texttt{in}(p,x).Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\texttt{in}(p,M)} (\mathcal{E}; Q\{x \mapsto u\} \uplus \mathcal{P}; \Phi) \qquad \text{(IN)}$$
if $p \notin \mathcal{E}$, $M\Phi = u$, $fv(M) \subseteq \mathrm{dom}(\Phi)$ and $fn(M) \cap \mathcal{E} = \emptyset$

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

$$(\mathcal{E}; \{\texttt{out}(p,c).Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\texttt{out}(p,c)} (\mathcal{E}; Q \uplus \mathcal{P}; \Phi) \quad \text{if } p, c \notin \mathcal{E} \qquad \text{(OUT-CH)}$$

$$(\mathcal{E}; \{\texttt{out}(p,c).Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\nu ch.\texttt{out}(p,ch)} (\mathcal{E}; (Q \uplus \mathcal{P})\{c \mapsto ch\}; \Phi) \qquad \text{(OPEN-CH)}$$
if $p \notin \mathcal{E}$, $c \in \mathcal{E}$, ch is a fresh channel name

$$(\mathcal{E}; \{\texttt{new } k.Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau} (\mathcal{E} \cup \{n\}; Q\{k \mapsto n\} \uplus \mathcal{P}; \Phi) \qquad \text{(NEW)}$$
if n is a fresh name with the same type as k

$$(\mathcal{E}; \{!Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau} (\mathcal{E}; \{!Q; Q\} \uplus \mathcal{P}; \Phi) \qquad \text{(REPL)}$$

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

where p, c are channel names, u, v are ground terms, and x is a variable.

Figure 1. Semantics

For every extended process $A = (\mathcal{E}; \mathcal{P}; \Phi)$, we define its set of traces, each trace consisting in a sequence of actions together with the sequence of sent messages:

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

Two processes are trace equivalent if, whatever the messages they received (built upon previously sent messages), the resulting sequences of messages are in static equivalence.

Definition 3: Let A and B be two extended processes, $A \sqsubseteq B$ if for every $(\mathsf{tr}, \phi) \in \mathsf{trace}(A)$ such that $bn(\mathsf{tr}) \cap (fn(B) \cup bn(B)) = \emptyset$, there exists $(\mathsf{tr}', \phi') \in \mathsf{trace}(B)$ such that $\mathsf{tr} = \mathsf{tr}'$ and $\phi \sim \phi'$. Two closed extended processes A and B are *trace equivalent*, denoted by $A \approx B$, if $A \sqsubseteq B$ and $B \sqsubseteq A$.

Example 5: Consider the following trace:

$$\mathsf{tr} = \nu w_1.\texttt{out}(c, w_1) \cdot \texttt{in}(c, w_1) \cdot \nu w_2.\texttt{out}(c, w_2).$$

We have that $(\mathsf{tr}, \phi_1) \in \mathsf{trace}(A_1)$, and the only trace $(\mathsf{tr}', \phi') \in \mathsf{trace}(A_2)$ that satisfies $\mathsf{tr} = \mathsf{tr}'$ leads to the frame ϕ_2 for which we have seen that $\phi_1 \not\sim \phi_2$ (see Example 4). This allows us to conclude that $A_1 \not\approx A_2$.

We do not consider α-renaming on processes, but renaming of bound names is taken into account in the definition of static equivalence (Definition 2), and also via the condition $bn(\mathsf{tr}) \cap (fn(B) \cup bn(B)) = \emptyset$ in the definition of trace equivalence (Definition 3).

Example 6: Consider the two extended processes:

$$A = (\{a\}; \texttt{out}(c, a).\texttt{in}(a, x); \emptyset)$$
$$B = (\{b\}; \texttt{out}(c, b).\texttt{in}(b, x); \emptyset)$$

where a and b are both channel names. Of course, these two extended processes are trace equivalent since they are actually equal up to α-renaming.

According to our semantics, we have that:

$$(\{a\}; \texttt{out}(c, a).\texttt{in}(a, x); \emptyset) \xrightarrow{\nu b.\texttt{out}(c,b)} (\{a\}; \texttt{in}(b, x); \emptyset).$$

It is not possible to fire the same transition, *i.e.* $\nu b.\texttt{out}(c, b)$, from B since the name b occurs in the process B and the rule OPEN-CH requires us to choose a fresh channel name. Actually, the extended process B does not have to mimic such a trace $\mathsf{tr} = \nu b.\texttt{out}(c, b)$. Indeed, we have that $bn(\mathsf{tr}) = \{b\}$ and $fn(B) \cup bn(B) = \{b, c\}$. Thus, we have that $bn(\mathsf{tr}) \cap (fn(B) \cup bn(B)) \neq \emptyset$.

Note that the extended process A can also performed the following transition:

$$(\{a\}; \texttt{out}(c, a).\texttt{in}(a, x); \emptyset) \xrightarrow{\nu d.\texttt{out}(c,d)} (\{a\}; \texttt{in}(d, x); \emptyset).$$

Since $\{d\} \cap (fn(B) \cup bn(B)) = \emptyset$, the process B has to mimic this trace $\mathsf{tr}' = \nu d.\texttt{out}(c, d)$, and the process B can do it. We have that:

$$(\{b\}; \texttt{out}(c, b).\texttt{in}(b, x); \emptyset) \xrightarrow{\nu d.\texttt{out}(c,d)} (\{b\}; \texttt{in}(d, x); \emptyset).$$

B. Some examples

The definitions we present here are informal ones, and we refer the reader to [5] for detailed formal definitions. In Section VI, we will illustrate these definitions through the e-passport application.

Strong anonymity: Anonymity is informally defined by the ISO/IEC standard 15408 [30] as the property ensuring that *a user may use a service or a resource without disclosing the user's identity*. Formally, strong anonymity has been defined to hold [5] when an outside observer cannot tell the difference between a system in which the user with a publicly known identity id_0 executes the analysed protocol, from the system where id_0 is not present at all.

Following this formal definition of anonymity, the protocol introduced in Section I considered in isolation, *i.e.* $P = \texttt{new}\ r.\texttt{out}(c, \mathsf{aenc}(\langle r, id\rangle, \mathsf{pk}(sk_S)))$, is said to satisfy strong anonymity if the following equivalence holds:

$$\texttt{new}\ sk_S.\ ((!\texttt{new}\ id.\ !P) \mid !P\{id_0/id\}) \approx \texttt{new}\ sk_S.(!\texttt{new}\ id.\ !P)$$

In other words, anonymity is satisfied if an observer cannot tell if the user id_0 (known to the attacker) has been executing the protocol P or not.

Strong unlinkability: Unlinkability is informally defined by the ISO/IEC standard 15408 [30] as the property ensuring that *a user may make multiple uses of a service or a resource without others being able to link these uses together*. Formally, strong unlinkability has been defined to hold [5] when a system in which the analysed protocol can be executed by each user multiple times looks the same to an outside observer that the system in which the analysed protocol can be executed by each user at most once.

Again, we can formalise this property for the protocol P when considered in isolation using an equivalence:

$$\texttt{new}\ sk_S.\ (!\texttt{new}\ id.\ !P) \ \approx\ \texttt{new}\ sk_S.(!\texttt{new}\ id.\ P)$$

In other words, unlinkability is satisfied if an observer cannot tell if the users can execute multiple or at most once the protocol P.

IV. Composition result: a simple setting

Even if a protocol is secure for an unbounded number of sessions, there is no guarantee if the protocol is executed in an environment where other protocols sharing some common keys are executed. The interaction with the other protocols may dramatically damage the security of the former protocol. This is a well-known fact that has been already observed for trace-based security properties *e.g.* [16], [17], and that remains true for privacy-type properties.

An attacker may take advantage of a protocol Q to break anonymity of another protocol P that has been proved secure in isolation. This can happen for instance if the security of P relies on the secrecy of a particular shared key that is revealed by the protocol Q.

A. Sharing primitives

Actually, even if shared keys are not revealed, the interaction of two protocols using common primitives may compromise their security.

Example 7: Consider the processes P_i with $i \in \{1, 2\}$ as defined in Example 2. The equivalence expressing the anonymity of P (for one session) holds. We have that $\texttt{new}\ sk_S.P_1 \approx \texttt{new}\ sk_S.P_2$ whereas the equivalence expressing the anonymity of P in presence of Q does not hold anymore. We have that:

$$\texttt{new}\ sk_S.\big(P_1 \mid Q\big) \not\approx \texttt{new}\ sk_S.\big(P_2 \mid Q\big)$$

Intuitively, the security of P is ensured by the fact that its identity id is encrypted using the public key $\mathsf{pk}(sk_S)$ whose associated private key sk_S is kept secret. However, Q can be used as an oracle to decrypt a ciphertext that comes from the process P, and thus Q can be used to reveal the identity hidden in the ciphertext.

To avoid a ciphertext from a process to be decrypted by another one, we can consider processes that use disjoint primitives. However, this is an unnecessarily restrictive condition. So, we consider protocols that may share some cryptographic primitives provided they are tagged.

Tagging is a syntactic transformation that consists in assigning to each protocol an identifier (*e.g.* the protocol's name) that should appear in any encrypted message. Many relevant equational theories are not so easy to tag (*e.g.* exclusive or). So, we consider the fix common equational theory (Σ_0, E_0) defined in Example 1, and we explain how to transform any process built on a signature Σ (possibly larger that Σ_0) into a well-tagged process. For this, we define $\Sigma_{\mathsf{tag}_c} = \{\mathsf{tag}_c, \mathsf{untag}_c\}$ where tag_c and untag_c are two function symbols of arity 1 that we will use for tagging. The role of the tag_c function is to tag its argument with the tag c. The role of the untag_c function is to remove the tag. To model this interaction between tag_c and untag_c, we consider the equational theory:

$$\mathsf{E}_{\mathsf{tag}_c} = \{\mathsf{untag}_c(\mathsf{tag}_c(x)) = x\}.$$

For our composition results, we will assume that the processes P_A and P_B that we want to compose are built on $(\Sigma_a \cup \Sigma_0, \mathsf{E}_a \cup \mathsf{E}_0)$ and $(\Sigma_b \cup \Sigma_0, \mathsf{E}_b \cup \mathsf{E}_0)$, where (Σ_a, E_a), (Σ_b, E_b) and (Σ_0, E_0) are disjoint signatures that are also disjoint from $(\Sigma_{\mathsf{tag}_a}, \mathsf{E}_{\mathsf{tag}_a})$ and $(\Sigma_{\mathsf{tag}_b}, \mathsf{E}_{\mathsf{tag}_b})$. The signature Σ_0 contains the function symbols that can be used by the two processes and that have to be tagged. We denote by $\Sigma_c^+ = \Sigma_c \cup \Sigma_{\mathsf{tag}_c}$ and $\mathsf{E}_c^+ = \mathsf{E}_c \cup \mathsf{E}_{\mathsf{tag}_c}$ with $c \in \{a, b\}$.

Definition 4: Let u be a term built on $\Sigma_c \cup \Sigma_0$ ($c \in \{a, b\}$). The c-tagged version of u, denoted $[u]_c$, is defined as follows:

$$
\begin{array}{rcl}
[u]_c & \stackrel{\mathsf{def}}{=} & u \text{ when } u \text{ is a name or a variable} \\
[\mathsf{senc}(u,v)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{senc}(\mathsf{tag}_c([u]_c), [v]_c) \\
[\mathsf{aenc}(u,v)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{aenc}(\mathsf{tag}_c([u]_c), [v]_c) \\
[\mathsf{sign}(u,v)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{sign}(\mathsf{tag}_c([u]_c), [v]_c) \\
[\mathsf{h}(u)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{h}(\mathsf{tag}_c([u]_c)) \\
[\mathsf{sdec}(u,v)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{untag}_c(\mathsf{sdec}([u]_c, [v]_c)) \\
[\mathsf{adec}(u,v)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{untag}_c(\mathsf{adec}([u]_c, [v]_c)) \\
[\mathsf{check}(u,v)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{untag}_c(\mathsf{check}([u]_c, [v]_c)) \\
[\mathsf{f}(u_1, \ldots, u_n)]_c & \stackrel{\mathsf{def}}{=} & \mathsf{f}([u_1]_c, \ldots, [u_n]_c) \text{ otherwise.}
\end{array}
$$

Note that we do not tag the pairing function symbol (this is actually useless), and we do not tag the pk and vk function symbols. Actually, tagging pk and vk would greatly help us to establish our results and would also avoid us to introduce some additional assumptions, but this would lead us to consider an unrealistic modelling for asymmetric keys. Some of the difficulties encountered with asymmetric keys will be discussed in Section V.

Furthermore, note that tagging preserves equality between terms (modulo the equational theory $\mathsf{E}_c^+ \cup \mathsf{E}_0$). Actually, for any terms u, v built on $(\Sigma_c \cup \Sigma_0, \mathsf{E}_c \cup \mathsf{E}_0)$, we have that $u =_{\mathsf{E}_c^+ \cup \mathsf{E}_0} v$ if and only if $[u]_c =_{\mathsf{E}_c^+ \cup \mathsf{E}_0} [v]_c$. In this paper, we state our composition results directly on the tagged version of the protocols, and consequently, we do not need to rely on this property. However, this property points out that tagging the terms occurring in a protocol should not modify its behavior in a fundamental way.

Example 8: Consider $u_i = \mathsf{aenc}(\langle r, id_i \rangle, \mathsf{pk}(sk_S))$ with $i \in \{1, 2\}$ and $v = \mathsf{proj}_2(\mathsf{adec}(x, sk_S))$. We have that $[u_i]_a = \mathsf{aenc}(\mathsf{tag}_a(\langle r, id_i \rangle), \mathsf{pk}(sk_S))$, whereas $[v]_b = \mathsf{proj}_2(\mathsf{untag}_b(\mathsf{adec}(x, sk_S)))$.

Before extending the notion of tagging to processes, we have to express the tests that are performed by an agent when he receives a message that is supposed to be tagged. This is the purpose of $\mathsf{test}_c(u)$ that represents the tests which ensure that every projection and every untagging performed by an agent during the computation of u is successful.

Definition 5: Let u be a term built on $\Sigma_c^+ \cup \Sigma_0$ with $c \in \{a, b\}$. We define $\mathsf{test}_c(u)$ as follows:

$$
\begin{array}{l}
\mathsf{test}_c(u) \stackrel{\mathsf{def}}{=} \mathsf{test}_c(u_1) \wedge \mathsf{test}_c(u_2) \wedge \mathsf{tag}_c(\mathsf{untag}_c(u)) = u \\
\qquad \text{when } u = \mathsf{g}(u_1, u_2) \text{ with } \mathsf{g} \in \{\mathsf{sdec}, \mathsf{adec}, \mathsf{check}\} \\
\mathsf{test}_c(u) \stackrel{\mathsf{def}}{=} \mathsf{test}_c(u_1) \wedge u_1 = \langle \mathsf{proj}_1(u_1), \mathsf{proj}_2(u_1) \rangle \\
\qquad \text{when } u = \mathsf{proj}_i(u_1) \text{ with } i \in \{1, 2\} \\
\mathsf{test}_c(u) \stackrel{\mathsf{def}}{=} \mathsf{true} \qquad \text{when } u \text{ is a name or a variable} \\
\mathsf{test}_c(u) \stackrel{\mathsf{def}}{=} \mathsf{test}_c(u_1) \wedge \ldots \wedge \mathsf{test}_c(u_n) \\
\qquad \text{otherwise, } u = \mathsf{f}(u_1, \ldots, u_n).
\end{array}
$$

Intuitively, the purpose of $\mathsf{test}_c(u)$ is to check whether the term u is properly tagged, *i.e.* whether u is the result of the tagging operation $[\,]_c$. Actually, this test also allows the process to check that the cryptographic primitives sdec, adec and check (the destructor symbols) succeed. This induces a small change between the behavior of a protocol and its tagged version. Indeed, whereas the term $u = \mathsf{sdec}(a, b)$ is a message that a protocol could accept, this term will not satisfy the test $\mathsf{test}_c(u)$ and will be rejected by the tagged version of the protocol. This behavior is very similar to one where the destructors in Σ_0 are modeled using rewrite rules instead of equations. Note that our composition results are stated on the tagged version of the protocols, and thus, in this paper, we do not have to worry so much about this aspect.

Example 9: Again, consider $u_i = \mathsf{aenc}(\langle r, id_i \rangle, \mathsf{pk}(sk_S))$ with $i \in \{1, 2\}$ and $v = \mathsf{proj}_2(\mathsf{adec}(x, sk_S))$. We have that:

$$
\begin{array}{l}
\mathsf{test}_a([u_i]_a) = \mathsf{true} \\
\mathsf{test}_b([v]_b) = \mathsf{tag}_b(\mathsf{untag}_b(\mathsf{adec}(x, sk_S))) = \mathsf{adec}(x, sk_S) \\
\qquad\qquad \wedge\ \langle \mathsf{proj}_1(v'), \mathsf{proj}_2(v') \rangle = v' \\
\qquad\qquad\qquad \text{where } v' = \mathsf{untag}_b(\mathsf{adec}(x, sk_S)).
\end{array}
$$

Let $A = (\mathcal{E}; \mathcal{P}; \Phi)$ be a process built on $\Sigma_c \cup \Sigma_0$ with $c \in \{a, b\}$ such that $\mathcal{P} = \{P_1, \ldots, P_\ell\}$, and $\Phi = \{w_1 \rhd u_1, \ldots, w_n \rhd u_n\}$. The c-tagged version of the process A, denoted $[A]_c$, is the process $(\mathcal{E}; [\mathcal{P}]_c; [\Phi]_c)$ where $[\mathcal{P}]_c = \{[P_1]_c, \ldots, [P_\ell]_c\}$, and

$$[\Phi]_c = \{w_1 \rhd [u_1]_c, \ldots, w_n \rhd [u_n]_c\}.$$

For plain processes, the transformation $[P]_c$ is defined as follows:

$$
\begin{array}{l}
[0]_c \stackrel{\mathsf{def}}{=} 0 \qquad [!P]_c \stackrel{\mathsf{def}}{=} ![P]_c \qquad [\mathtt{new}\ k.P]_c \stackrel{\mathsf{def}}{=} \mathtt{new}\ k.[P]_c \\
[P \mid Q]_c \stackrel{\mathsf{def}}{=} [P]_c \mid [Q]_c \qquad [\mathtt{in}(u,x).P]_c \stackrel{\mathsf{def}}{=} \mathtt{in}(u,x).[P]_c \\
[\mathtt{out}(u,v).Q]_c \stackrel{\mathsf{def}}{=} \mathtt{if}\ \mathsf{test}_c([v]_c)\ \mathtt{then}\ \mathtt{out}(u, [v]_c).[Q]_c \\
[\mathtt{if}\ u_1 = u_2\ \mathtt{then}\ P\ \mathtt{else}\ Q]_c \stackrel{\mathsf{def}}{=} \\
\quad \mathtt{if}\ \varphi\ \mathtt{then}\ (\mathtt{if}\ [u_1]_c = [u_2]_c\ \mathtt{then}\ [P]_c\ \mathtt{else}\ [Q]_c) \\
\qquad \mathtt{else}\ 0 \\
\qquad\qquad \text{where } \varphi = \mathsf{test}_c([u_1]_c) \wedge \mathsf{test}_c([u_2]_c)
\end{array}
$$

Roughly, instead of simply outputting a term v, a process will first perform some tests to check that the term is correctly tagged and it will output its c-tagged version $[v]_c$. For a conditional, the process will first check that the terms u_1 and u_2 are correctly tagged before checking that the test is satisfied.

Example 10: Consider the processes P_i and Q defined in Example 2.

$$
\begin{array}{l}
[P_i]_a = \mathtt{new}\ r.\ \mathtt{out}(c, \mathsf{aenc}(\mathsf{tag}_a(\langle r, id_i \rangle), \mathsf{pk}(sk_S))) \\
[Q]_b = \mathtt{in}(c,x).\mathtt{if}\ \mathsf{test}_b([v]_b)\ \mathtt{then}\ \mathtt{out}(c, [v]_b)
\end{array}
$$

where $[v]_b$ (resp. $\mathsf{test}_b([v]_b)$) have been defined in Example 8 (resp. Example 9).

Note that the tag will prevent the process Q from decrypting the ciphertext that has been output by P_i. Thus, the equivalence expressing the anonymity of $[P_i]_a$ now holds even in the presence of $[Q]_b$. We have that:

$$\mathtt{new}\ sk_S.\big([P_1]_a \mid [Q]_b\big) \approx \mathtt{new}\ sk_S.\big([P_2]_a \mid [Q]_b\big).$$

This is a non-trivial equivalence that can actually be derived from the equivalence $\mathtt{new}\ sk_S.[P_1]_a \approx \mathtt{new}\ sk_S.[P_2]_a$ using our composition result (Corollary 1).

B. Composition context

As already mentioned, we want to establish a composition result between processes that share the signature (Σ_0, E_0) and also share some keys. Thus, we introduce the notion of *composition context* that will help us describe under which keys the composition has to be done. Note that a composition context may contain several holes, parallel operators, and nested replications. This is needed to express privacy-type properties as those described in Section III-B.

Definition 6: A *composition context* C is defined by the following grammar where n is a name of base type.

$$C, C_1, C_2 := \_ \mid \mathtt{new}\ n.\ C \mid !C \mid C_1|C_2$$

We only allow names of base type (typically keys) to be shared between processes through the composition context. In particular, they are not allowed to share a private channel even if each process can use its own private channels to communicate internally. We also suppose w.l.o.g. that names occurring in C are distinct. A composition context may contain several holes. We can index them to avoid confusion. We write $C[P_1, \ldots, P_\ell]$ (or shortly $C[\overline{P}]$) the process obtained by filling the i^{th} hole with the process P_i (or the i^{th} process of the sequence $\overline{P}$). We will also use $\overline{P} \mid \overline{Q}$ to represent the sequence of processes obtained by putting in parallel the processes of the sequences $\overline{P}$ and $\overline{Q}$ componentwise.

Example 11: In Section III-B, we have seen that unlinkability of P can be modelled using the equivalence:

$$\mathtt{new}\ sk_S.\big(!\mathtt{new}\ id.!P\big) \approx \mathtt{new}\ sk_S.\big(!\mathtt{new}\ id.P\big).$$

The composition contexts used to express this property are:

- $C[\_] = \mathtt{new}\ sk_S.\big(!\mathtt{new}\ id.\ !\ \_\big)$, and
- $C'[\_] = \mathtt{new}\ sk_S.\big(!\mathtt{new}\ id.\ \_\big)$.

Since the name id does not occur in the process Q (see Example 2), it is quite easy to see that $C[Q] \approx C'[Q]$. Unlinkability of P in presence of the process Q will be modelled as $C[P \mid Q] \approx C'[P \mid Q]$, *i.e.*:

$$\mathtt{new}\ sk_S.\big(!\mathtt{new}\ id.\ !(P \mid Q)\big) \approx \mathtt{new}\ sk_S.\big(!\mathtt{new}\ id.(P \mid Q)\big)$$

Taking into account the fact that id does not occur in the process Q, the relation above is equivalent to:

$$\mathtt{new}\ sk_S.!\big((\mathtt{new}\ id.!P) \mid Q\big) \approx \mathtt{new}\ sk_S.!\big((\mathtt{new}\ id.P) \mid Q\big)$$

Note that in a composition context a replication may occur in the scope of some restrictions and this is needed to express many interesting privacy-type properties. Considering composition in a simpler setting where only a bounded number of keys $\tilde{k}$ are shared (as done in *e.g.* [31]), would not allow us to establish unlinkability in a modular way, but only some results of the form:

$$\mathtt{new}\ \tilde{k}.P_1 \approx \mathtt{new}\ \tilde{k}.P_2 \ \Rightarrow\ \mathtt{new}\ \tilde{k}.\ (P_1 \mid Q) \approx \mathtt{new}\ \tilde{k}.\ (P_2 \mid Q)$$

assuming that processes P_1, P_2, and Q satisfy some additional conditions.

Now, we have introduced composition under replication, but have to formalise the notion of revealing a shared key. The names that occur in the composition context represent the names that are shared between the two processes that we want to compose. Since those names may occur under a replication, we have to consider renaming and formalise this notion of revealing accordingly. This is the purpose of the second part of Definition 7. Actually, in order to compose protocols in presence of public shared keys, we have to model those keys using the first component of a process (so that their public counterpart may occur in the third component of the process, *i.e.* the frame). For those keys, this notion of revealing can be modeled in a very similar way. This is the purpose of the first part of Definition 7 which will be useful to state our second composition result (see Section V).

Definition 7: Let C be a composition context, A be an extended process of the form $(\mathcal{E}; C[P_1, \ldots, P_\ell]; \Phi)$, and $key \in \{n, \mathsf{pk}(n), \mathsf{vk}(n) \mid n \in \mathcal{E} \text{ or } n \text{ occurs in } C\}$. We say that *the extended process A reveals the shared key key* when:

Either $fn(key) \in \mathcal{E}$, and

- $A \overset{w}{\Rightarrow} (\mathcal{E}'; \mathcal{P}'; \Phi')$ for some $(\mathcal{E}'; \mathcal{P}'; \Phi')$; and
- $M\Phi' =_{\mathsf{E}} key$ for some M such that $fv(M) \subseteq \mathrm{dom}(\Phi')$ and $fn(M) \cap \mathcal{E}' = \emptyset$.

Or, we have that $fn(key)$ occurs in C, the $i_0{}^{\text{th}}$ hole is in the scope of $\mathtt{new}\ fn(key)$, and

- $(\mathcal{E} \cup \{s\}; C[P_1^+, \ldots, P_\ell^+]; \Phi) \overset{w}{\Rightarrow} (\mathcal{E}'; \mathcal{P}'; \Phi')$ with $P_{i_0}^+ \overset{\mathsf{def}}{=} P_{i_0} \mid \mathtt{in}(c, x).\mathtt{if}\ x = key\ \mathtt{then}\ \mathtt{out}(c, s)$ and $P_i^+ \overset{\mathsf{def}}{=} P_i$ if $i \neq i_0$; and
- $M\Phi' =_{\mathsf{E}} s$ for some M such that $fv(M) \subseteq \mathrm{dom}(\Phi')$ and $fn(M) \cap \mathcal{E}' = \emptyset$

where c is a fresh public channel name, and s is a fresh name of base type.

Example 12: Consider the composition context $C[\_] = \mathtt{new}\ sk_S.\ \_$. The extended process $(\emptyset; C[P]; \emptyset)$ with P

as described in Section III-B does not reveal the keys sk_S, $\mathsf{pk}(sk_S)$ and $\mathsf{vk}(sk_S)$. Indeed, let $key \in \{sk_S, \mathsf{pk}(sk_S), \mathsf{vk}(sk_S)\}$, we have that

$$(\{s\}; C[P \mid \mathtt{in}(c,x).\,\mathtt{if}\ \ x = key\ \mathtt{then}\,\mathtt{out}(c,s)]; \emptyset)$$

cannot reach a configuration from which s will be derivable by the attacker.

C. Going back to the disjoint case

It is well-know that parallel composition works when processes do not share any secret, the so-called disjoint case. A first idea to establish a composition result is to see under which conditions we can go back to the disjoint case. In this section, we will see that this is indeed possible provided that processes are tagged and only share some keys that will never be revealed.

Theorem 1: Let C be a composition context, and $\overline{P_A}$ (resp. $\overline{P_B}$) be two sequences of plain processes built on the signature $\Sigma_a \cup \Sigma_0$ (resp $\Sigma_b \cup \Sigma_0$). Assume that $C[[\overline{P_A}]_a]$ and $C[[\overline{P_B}]_b]$ do not reveal any shared key in $\{k, \mathsf{pk}(k), \mathsf{vk}(k) \mid k \text{ occurs in } C\}$. We have that:

$$C[[\overline{P_A}]_a \mid [\overline{P_B}]_b] \approx C[[\overline{P_A}]_a] \mid C[[\overline{P_B}]_b].$$

Proof: (*sketch*) Consider $S = (\emptyset; C[[\overline{P_A}]_a \mid [\overline{P_B}]_b]; \emptyset)$ and $D = (\emptyset; C[[\overline{P_A}]_a] \mid C[[\overline{P_B}]_b]; \emptyset)$. Actually, we can show that any trace $(\mathsf{tr}, \phi_D) \in \mathsf{trace}(D)$ can be mapped to a trace $(\mathsf{tr}, \phi_S) \in \mathsf{trace}(S)$ such that $\phi_D \sim \phi_S$ and conversely. Note that even if the resulting frames ϕ_S and ϕ_D are not syntactically equal, we can show that they are in static equivalence and the computation performed by the attacker in both executions are exactly the same, namely tr.

For this, we consider the transformation δ on terms whose purpose is to replace the occurrences of the shared keys that come from $\overline{P_B}$ by some fresh names in order to ensure disjointness. However, we do not want to replace any occurrence of a shared key. Thus, we have to identify those that come from $\overline{P_B}$.

For instance, assume that the following term $u = \mathsf{senc}(\mathsf{tag}_b(\mathsf{senc}(\mathsf{tag}_a(n_a), k)), k)$ has been output by the process $P_B = \mathtt{in}(c,x).\mathtt{out}(c, \mathsf{senc}(\mathsf{tag}_b(x), k))$. The purpose of δ is to replace the occurrences of the shared k that "comes from P_B" by a fresh key k'. Actually, we have that:

$$\delta(u) = \mathsf{senc}(\mathsf{tag}_b(\mathsf{senc}(\mathsf{tag}_a(n_a), k)), k').$$

Then the proof can go through thanks to some nice properties that are enjoyed by this transformation δ. In particular, we have that:

- this transformation preserves the equality tests performed by each process: "$\delta(u) = \delta(v) \Leftrightarrow u = v$".
- this transformation preserves deducibility in the sense that for any message u that the attacker can obtain from ϕ_S, we can show that its counterpart $\delta(u)$ can be obtained using "$\delta(\phi_S) = \phi_D$" using the same recipe (and conversely).

■

This result as well as the way we proceed to prove it are close to the one proved in [17]. However, we generalise it in several ways. First, we combine the results of [17] so that we are able to deal with disjoint equational theories together with a common equational theory. Moreover, for the common theory, we consider also pairing and asymmetric primitives. Due to the way tagging is performed, the asymmetric primitives add some difficulties. Second, since we want a composition result for trace equivalence, we have to map any trace of D to a trace of S (and conversely), and we also have to ensure that the resulting sequence of messages are in static equivalence. Third, we consider a process algebra that allows us to express disequality tests (*i.e.* non-trivial else branches).

Note that, we have to ensure that shared keys are never revealed. This is needed for symmetric keys, but as mentioned in the hypothesis of the proposition, this is also required for public keys and verification keys. As we will see in Example 15, this hypothesis is necessary for this result to hold, but we will show how to relax it and still get a composition result (see Section V).

D. A first composition result

The result stated in Theorem 1 allows us to go back to the disjoint case for which composition works quite well. Hence, as a corollary, we are now able to state our first composition result.

Corollary 1: Let C and C' be two composition contexts. Let $\overline{P_A}, \overline{P'_A}$ (resp. $\overline{P_B}, \overline{P'_B}$) be two sequences of plain processes built on the signature $\Sigma_a \cup \Sigma_0$ (resp. $\Sigma_b \cup \Sigma_0$). Assume that $C[[\overline{P_A}]_a]$ and $C[[\overline{P_B}]_b]$ (resp. $C'[[\overline{P'_A}]_a]$ and $C'[[\overline{P'_B}]_b]$) do not reveal any shared key in $\{k, \mathsf{pk}(k), \mathsf{vk}(k) \mid k \text{ occurs in } C\}$ (resp. $\{k, \mathsf{pk}(k), \mathsf{vk}(k) \mid k \text{ occurs in } C'\}$). We have that:

$$\frac{\begin{array}{c} C[[\overline{P_A}]_a] \ \approx\ C'[[\overline{P'_A}]_a] \\ C[[\overline{P_B}]_b] \ \approx\ C'[[\overline{P'_B}]_b] \end{array}}{C[[\overline{P_A}]_a \mid [\overline{P_B}]_b] \ \approx\ C'[[\overline{P'_A}]_a \mid [\overline{P'_B}]_b]}$$

Proof: (sketch) This composition result is proved in three main steps.

1) We have that the equivalences $C[[\overline{P_A}]_a] \approx C'[[\overline{P'_A}]_a]$ and $C[[\overline{P_B}]_b] \approx C'[[\overline{P'_B}]_b]$ hold on the signatures $(\Sigma_a^+ \cup \Sigma_0, \mathsf{E}_a^+ \cup \mathsf{E}_0)$ and $(\Sigma_b^+ \cup \Sigma_0, \mathsf{E}_b^+ \cup \mathsf{E}_0)$ respectively. It is relatively easy to show that the same equivalences also hold on the augmented signature $(\Sigma_a^+ \cup \Sigma_b^+ \cup \Sigma_0, \mathsf{E}_a^+ \cup \mathsf{E}_b^+ \cup \mathsf{E}_0)$.

2) Then, relying on these two equivalences, we can show that:

$$C[[\overline{P_A}]_a] \mid C[[\overline{P_B}]_b] \;\approx\; C'[[\overline{P'_A}]_a] \mid C'[[\overline{P'_B}]_b].$$

This corresponds to composition in the disjoint case (no shared key). This is a well-know fact that actually holds in many cryptographic calculus.

3) Then, we apply Theorem 1 on both sides of the equivalence, and we obtain the expected result:

$$C[[\overline{P_A}]_a \mid [\overline{P_B}]_b] \;\approx\; C'[[\overline{P'_A}]_a \mid [\overline{P'_B}]_b].$$

■

V. Composition in presence of processes that reveal shared keys

In the previous section, we presented a first composition result. However, this result does not hold as soon as some shared keys are revealed: such a key can be a symmetric shared key, the private part of an asymmetric key pair, but also the public part of an asymmetric key pair. In this section, we will see that we can relax this condition by allowing shared keys to be revealed from the beginning.

A. Some additional difficulties

First, as shown by the example below, we do not want public keys to be revealed (for the first time) during the execution of the protocol.

Example 13: We consider a slightly different version of the process P_i introduced in Example 2. Basically, we remove the random r inside the encryption and we consider its well-tagged version. We consider the following processes:

$$[P'_i]_a \stackrel{\mathsf{def}}{=} \mathtt{out}(c, \mathsf{aenc}(\mathsf{tag}_a(id_i), \mathsf{pk}(sk_S))) \quad i \in \{1,2\}$$

Consider the composition context $C[\_] = \mathtt{new}\ sk_S.\ \_$. Note that, the equivalence $C[[P'_1]_a] \approx C[[P'_2]_a]$ still holds in this setting. Assume now that $[P'_i]_a$ is executed in the presence of the well-tagged process $Q^{\mathsf{pk}} = \mathtt{out}(c, \mathsf{pk}(sk_S))$. Clearly, the equivalence expressing the anonymity of $[P'_i]_a$ does not hold anymore. We have that:

$$C[[P'_1]_a \mid Q^{\mathsf{pk}}] \not\approx C[[P'_2]_a \mid Q^{\mathsf{pk}}].$$

Actually, the knowledge of $\mathsf{pk}(sk_S)$ will allow the attacker to distinguish the message emitted by $[P'_1]_a$ from the one emitted by $[P'_2]_a$.

To avoid the problem mentioned above, we will assume that shared keys that are revealed have to be revealed from the very beginning. This hypothesis seems indeed reasonable since the purpose of a public key is in general to be disclosed at the beginning, or eventually never revealed to an outsider.

Note that the previous example is not a counter-example anymore if we analyse the equivalence expressing the anonymity of $[P'_i]_a$ assuming that $\mathsf{pk}(sk_S)$ is known by the attacker from the beginning. The fact that $\mathsf{pk}(sk_S)$ is revealed during the execution of Q^{pk} will not give any additional power to the attacker.

Example 14: We consider again the process P_i as presented in Example 2 with an additional output to reveal the public key $\mathsf{pk}(sk_S)$ at the very beginning. Basically, we consider the well-tagged process $P''_i \stackrel{\mathsf{def}}{=} \mathtt{out}(c, \mathsf{pk}(sk_S)).[P_i]_a$.

We have that $C[P''_1] \approx C[P''_2]$ with $C[\_] = \mathtt{new}\ sk_S.\_$. Now, the presence of Q^{pk} will not prevent this equivalence to hold. Indeed, we have that:

$$C[P''_1 \mid Q^{\mathsf{pk}}] \approx C[P''_2 \mid Q^{\mathsf{pk}}].$$

This hypothesis that states that shared keys are either known from the beginning or never revealed during the execution of the protocol is reasonable, and seems to be sufficient to establish a composition result. However, this complicates a bit the setting. In particular, as illustrated in Example 15, there is no hope to obtain a result as the one stated in Theorem 1. The situation where the processes share some keys is not equivalent in this setting to the situation where the processes do not share any key.

Example 15: Consider the processes P''_i and Q^{pk} used in Example 14. We have seen that composition works under the composition context $C = \mathtt{new}\ sk_S.\_$. However, we have that ($i \in \{1,2\}$):

$$C[P''_i \mid Q^{\mathsf{pk}}] \;\not\approx\; C[P''_i] \mid C[Q^{\mathsf{pk}}].$$

Indeed, on the left-hand side, the same public-key will be output twice whereas the process on the right-hand side will emit two different public keys. The attacker will observe such a difference. The strong result stated in Theorem 1 allowing us to easily make the link between the joint state case and the disjoint case does not hold anymore.

The problems encountered for composing processes that reveal shared keys are due to the fact that we do not want to tag the function symbols pk and vk that are used to model asymmetric keys: such a tagging scheme would lead us to an unrealistic modelling of asymmetric keys.

B. Composition result

We now consider public keys and verifications keys that can be made public from the beginning through an initial frame Φ_0 that will represent the initial knowledge of the attacker. As illustrated in Section V-A, we cannot rely on Theorem 1 anymore to establish our composition result. We will still go back to the disjoint case but we have to explain how a trace corresponding to the situation where processes share some keys is transformed and mapped to a trace that models the disjoint case. We cannot simply consider the identity transformation as it was done to establish the previous result. The sets of traces issued by both situations are not the same anymore.

Theorem 2: Let $\overline{P_A}, \overline{P'_A}$ (resp. $\overline{P_B}, \overline{P'_B}$) be two sequences of plain processes built over $\Sigma_a \cup \Sigma_0$ (resp. $\Sigma_b \cup \Sigma_0$). Let $\mathcal{K}_0$ be a finite set of names of base type, and C and C' be two composition contexts. Let $\Phi_0 = \{w_1 \rhd \mathsf{f}_1(k_1), \ldots, w_n \rhd \mathsf{f}_n(k_n)\}$ with $\mathsf{f}_i \in \{\mathsf{pk}, \mathsf{vk}\}$, and $k_i \in \mathcal{K}_0$ for any $i \in \{1, \ldots, n\}$.

Assume that $(\mathcal{K}_0; C[[\overline{P_A}]_a]; \Phi_0)$ and $(\mathcal{K}_0; C[[\overline{P_B}]_b]; \Phi_0)$ (resp. $(\mathcal{K}_0; C'[[\overline{P'_A}]_a]; \Phi_0)$, and $(\mathcal{K}_0; C'[[\overline{P'_B}]_b]; \Phi_0)$):

- do not reveal any shared key in $\{k, \mathsf{pk}(k), \mathsf{vk}(k) \mid k \in \mathcal{K}_0\}$ unless if the key occurs explicitly in Φ_0; and
- do not reveal any shared key in C (resp. C');

Lastly, we assume that processes $\overline{P_A}, \overline{P'_A}$ and $\overline{P_B}, \overline{P'_B}$ do not use variables of channel type. We have that:

$$(\mathcal{K}_0; C[[\overline{P_A}]_a]; \Phi_0) \approx (\mathcal{K}_0; C'[[\overline{P'_A}]_a]; \Phi_0)$$
$$(\mathcal{K}_0; C[[\overline{P_B}]_b]; \Phi_0) \approx (\mathcal{K}_0; C'[[\overline{P'_B}]_b]; \Phi_0)$$

$$(\mathcal{K}_0; C[[\overline{P_A}]_a \mid [\overline{P_B}]_b]; \Phi_0) \approx (\mathcal{K}_0; C'[[\overline{P'_A}]_a \mid [\overline{P'_B}]_b]; \Phi_0)$$

Proof: (sketch) Actually, the two first steps are quite similar to the two first steps of the proof of Corollary 1, but we renamed the channel names that occur in $\overline{P_A}, \overline{P'_A}$ (resp. $\overline{P_B}, \overline{P'_B}$) before to compose these processes. This, together with our additional hypothesis on the variables of channel type, will allow us to identify easily whether a given action has been performed by $\overline{P_A}$ or $\overline{P_B}$ (resp. $\overline{P'_A}$ or $\overline{P'_B}$).

Then, consider a trace (tr, ϕ_S) issued by

$$S = (\mathcal{K}_0; C[[\overline{P_A}]_a \mid [\overline{P_B}]_b]; \Phi_0).$$

First, we show that a similar trace (tr', ϕ_D) is also issued by $D = (\mathcal{K}_0; C[[\overline{P_A}]_a] \mid C[[\overline{P_B}]_b]; \Phi_0)$ (where channel names have been renamed). Actually, the processes along these two traces will be very similar (up to a transformation similar to the δ transformation used in the proof of Corollary 1 and a renaming on the channel names) but the labels involved in tr' have to be changed. Indeed, as soon as a message u will involve a public key in a "deducible position", the attacker will not be able to produce u and $\delta(u)$ using the same recipe. The way the recipe has to be changed depends in particular on whether the action has been initiated by $\overline{P_A}$ or $\overline{P_B}$.

Second, relying on our hypothesis, we know that there exists (tr', ϕ'_D) issued by

$$D' = (\mathcal{K}_0; C'[[\overline{P'_A}]_a] \mid C'[[\overline{P'_B}]_b]; \Phi_0)$$

where again channel names have been renamed.

However, to conclude, we have to go back to the process $S' = (\mathcal{K}_0; C'[[\overline{P'_A}]_a \mid [\overline{P'_B}]_b]; \Phi_0)$. This can be done by applying the reverse of the transformation δ on each process that occurs in the trace, but again the labels that occur in tr' have to be changed. Moreover, we have to ensure that this change will allow one to retrieve the original sequence tr. For this, we use the fact that the actions of $\overline{P_A}$ (resp. $\overline{P_B}$) are mimicked by $\overline{P'_A}$ (resp. $\overline{P'_B}$) (this is enforced by the way we have renamed channel names). Actually, some complications appear when an internal communication is performed on a public channel (this is indeed allowed by the semantics), but this problem can be solved by replacing such an internal step with two visible actions (an output followed by an input) having a clearly identifiable origin. ■

VI. Application: e-passport

We illustrate the usefulness of our composition results on the e-passport application. An electronic passport (or e-passport) is a paper passport with an RFID chip that stores the critical information printed on the passport. The International Civil Aviation Organisation (ICAO) standard [32] specifies the communication protocols that are used to access these information.

A. Protocols description

The information stored in the chip is organised in data groups (dg_1 to dg_{19}). For example, dg_5 contains a JPEG copy of the displayed picture, and dg_7 contains the displayed signature. 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}. The corresponding signing key sk_P is stored in a tamper resistant memory, and cannot be read or copied. For authentication purposes, a hash of all the dgs together with a signature on this hash value issued by the Document Signer authority are stored in a separate file, the Security Object Document:

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

The ICAO standard specifies several protocols through which these information can be accessed. First, the Basic Access Control (*BAC*) protocol establishes sessions keys $ksenc$ and $ksmac$ to prevent skimming and eavesdropping on the subsequent communication with the e-passport. Once the *BAC* protocol has been successfully executed, the reader gains access to the information stored in the RFID tag through the Passive Authentication and the Active Authentication protocols that can be executed in any order (see Figure 2).

The Passive Authentication (PA) protocol is an authentication mechanism that proves that the content of the RFID chip is authentic. Through *PA* the reader retrieves the information stored in the dgs and the sod. It then verifies that the hash value stored in the sod corresponds to the one signed by the Document Signer authority. It further checks that this hash value is consistent with the received dgs.

The Active Authentication (AA) protocol is an authentication mechanism that prevents cloning of the passport chip. It relies on the fact that the secret key sk_P of the passport cannot be read or copied. The reader sends a random challenge to the passport, that has to return a signature on

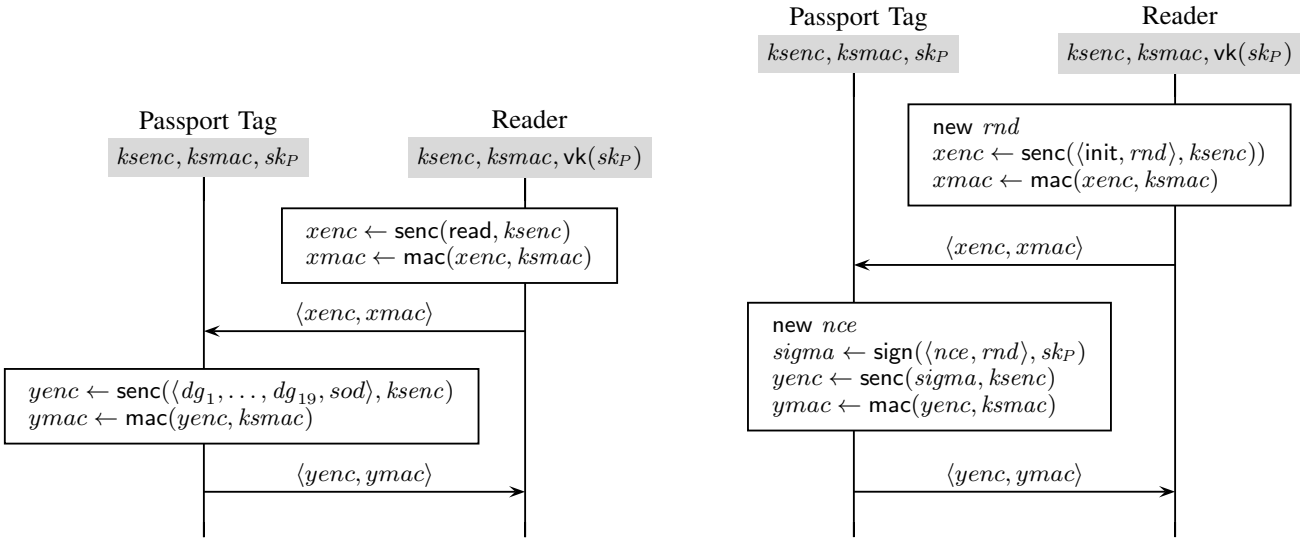

Figure 2. Passive and Active Authentication protocols

this challenge using its private signature key sk_P. The reader can then verify using the verification key $\mathsf{vk}(sk_P)$ that the signature was built using the expected passport key.

B. Privacy analysis

Both protocols *PA* and *AA* rely on symmetric encryption, message authentication codes, signatures and the verification key generation function, to meet their security requirements. Note that $\mathsf{mac}(m, k)$ can be modelled in our setting using the hash function symbol, *i.e.* $\mathsf{mac}(m, k) \stackrel{\mathsf{def}}{=} \mathsf{h}(\langle m, k \rangle)$. Moreover, the only publicly known verification key is $\mathsf{vk}(sk_{DS})$. Thus, we can use our composition results, and in particular Theorem 2, to reason in a modular way about the privacy guarantees provided by the tagged version of the e-passport application.

According to the ICAO standard, once the keys $ksenc$ and $ksmac$ have been established (using the *BAC* protocol), the reader can decide to execute *PA* and/or *AA* in any order. Formally, this corresponds to the parallel composition of *PA* and *AA*. We consider here that the keys $ksenc$ and $ksmac$ are "securely" pre-shared. We consider an arbitrary number of passports, each running an arbitrary number of times the *PA* and the *AA* protocols. This situation can be modelled in our calculus as follows:

$$\begin{aligned} P \stackrel{\mathsf{def}}{=} \; & \mathtt{new}\ sk_{DS}. \\ & !\mathtt{new}\ sk_P.\ \mathtt{new}\ id.\ \mathtt{new}\ sig.\ \mathtt{new}\ pic.\ \ldots \\ & !\mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ (\mathit{PA} \mid \mathit{AA}) \end{aligned}$$

where id, sig, pic, ... represent the name, the signature, the displayed picture, *etc* of the e-passport owner, *i.e.* the data stored in the dgs (1-14) and (16-19). The subprocesses *PA* and *AA* model one session of the *PA* and *AA* protocol respectively. The name sk_{DS} models the signing key of the Document Signing authority used in all passports. Each passport (identified by its signing key sk_P, the owner's name, picture, signature, ...) can run multiple times and in any order the *PA* and *AA* protocols, but with different secret session keys $ksenc$ and $ksmac$, that should be established through execution of the *BAC* protocol (but that we've abstracted from).

1) Strong anonymity: To express strong anonymity as formally defined in [5] and briefly discussed at Section III-B, we will need to consider a victim's e-passport, whose name id_0, signature sig_0, picture pic_0, *etc.* are known to the attacker. The victim's e-passport follows like any other e-passport the *PA* and *AA* protocols which can be respectively modelled by the following processes:

$$\begin{aligned} PA_0 &\stackrel{\mathsf{def}}{=} PA\{id_0/id, sig_0/sig, pic_0/pic, \ldots\} \\ AA_0 &\stackrel{\mathsf{def}}{=} AA\{id_0/id, sig_0/sig, pic_0/pic, \ldots\} \end{aligned}$$

To formally express strong anonymity, we will consider the following situation:

$$\begin{aligned} C[\__1, \__2] \stackrel{\mathsf{def}}{=} \; & !\ \mathtt{new}\ sk_P.\ \mathtt{new}\ id.\ \mathtt{new}\ sig.\ \mathtt{new}\ pic.\ \ldots \\ & \quad !\ \mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ \__1 \\ & \mid \mathtt{new}\ sk_P.\ !\mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ \__2 \end{aligned}$$

where the second hole will be filled with the process modelling the victim's e-passport, while the first hole will be filled with the processes modelling any other e-passport. This system will be compared to the one where the victim's e-passport is not present at all. For this we consider the following situation:

$$C'[\_] \stackrel{\mathsf{def}}{=} !\ \mathtt{new}\ sk_P.\ \mathtt{new}\ id.\ \mathtt{new}\ sig.\ \mathtt{new}\ pic.\ \ldots \\ !\ \mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ \_$$

whose unique hole will be filled with the processes modelling any e-passport but the victim's. In both situations, we will consider that the secret key sk_{DS} is secret whereas its associated verification key $\mathsf{vk}(sk_{DS})$ is publicly known to the attacker from the beginning, *i.e.* $\Phi_0 = \{w_1 \rhd \mathsf{vk}(sk_{DS})\}$.

To check if the tagged version of the e-passport application preserves its users' strong anonymity, one thus needs to check if the following equivalence holds:

$$(sk_{DS}; C[[PA]_a \mid [AA]_b,\ [PA_0]_a \mid [AA_0]_b]; \Phi_0) \\ \approx \\ (sk_{DS}; C'[[PA]_a \mid [AA]_b]; \Phi_0)$$

Now, according to our Theorem 2, instead of checking the above equivalence, one can check *PA*'s and *AA*'s guarantees *w.r.t.* anonymity in isolation. In other words, the above equivalence can be derived from the two following equivalences that are simpler to check:

$$(sk_{DS}; C[[PA]_a, [PA_0]_a]; \Phi_0) \approx (sk_{DS}; C'[[PA]_a]; \Phi_0)$$
$$(sk_{DS}; C[[AA]_b, [AA_0]_b]; \Phi_0) \approx (sk_{DS}; C'[[AA]_b]; \Phi_0)$$

2) Strong unlinkability: To express strong unlinkability as defined in [5] and briefly discussed in Section III-B, we need on one hand to consider a system in which e-passports can execute the *PA* and *AA* protocols multiple times, and on the other hand a system in which e-passports can execute the *PA* and *AA* protocols at most once. For this we consider the two following composition contexts:

$$C[\_] \stackrel{\mathsf{def}}{=} !\mathtt{new}\ sk_P.\ \mathtt{new}\ id.\ \mathtt{new}\ sig.\ \mathtt{new}\ pic.\ \ldots \\ !\mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ \_$$
$$C'[\_] \stackrel{\mathsf{def}}{=} !\mathtt{new}\ sk_P.\ \mathtt{new}\ id.\ \mathtt{new}\ sig.\ \mathtt{new}\ pic.\ \ldots \\ \mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ \_$$

These two composition contexts differ on the replication before the generation of the session keys $ksenc$ and $ksmac$, modelling in the first case an unbounded number of executions of the process that will fill the unique hole, and in the second a unique session of the filling process.

To check if the tagged version of the e-passport application preserves strong unlinkability, one thus needs to check:

$$(sk_{DS}; C[[PA]_A|[AA]_b]; \Phi_0) \approx (sk_{DS}; C'[[PA]_a|[AA]_b]; \Phi_0)$$

We can instead check whether *PA* and *AA* satisfy unlinkability in isolation:

$$(sk_{DS}; C[[PA]_a]; \Phi_0) \approx (sk_{DS}; C'[[PA]_a]; \Phi_0)$$
$$(sk_{DS}; C[[AA]_b]; \Phi_0) \approx (sk_{DS}; C'[[AA]_b]; \Phi_0)$$

Then, using Theorem 2, we derive the required equivalence.

A few words on ProVerif and the analysis of the e-passport application: To the best of our knowledge, the ProVerif tool [7] is the only available tool for automatically analysing equivalences, and thus for establishing privacy-type properties as the ones presented in Section III-B. We tried to use ProVerif to prove that the e-passport application as a whole (both *PA* and *AA* running in parallel) satisfies anonymity, but it failed to terminate. The problem comes from the *PA* protocol. Indeed, while ProVerif fails to prove that *PA* satisfies anonymity and unlinkability (ProVerif does not terminate), it can perfectly prove that *AA* satisfies these two properties.

Without results for modular reasoning we cannot exploit the proof of anonymity and unlinkability of the *AA* protocol provided by ProVerif. This reinforces the need for techniques for modular reasoning.

To use ProVerif we need to encode the equivalences of interest as biprocesses. We here briefly explain how we did this for analysing the *AA* protocol. As discussed above, *AA* satisfies strong anonymity if the following equivalence holds:

$$(sk_{DS}; C[[AA]_b, [AA_0]_b]; \Phi_0) \approx (sk_{DS}; C'[[AA]_b]; \Phi_0)$$

Actually, this equivalence can be encoded by the following ProVerif biprocess

$$AA_{\text{ANON}} \stackrel{\mathsf{def}}{=} !\ \mathtt{new}\ sk_P.\ \mathtt{new}\ id.\ \mathtt{new}\ sig.\ \mathtt{new}\ pic.\ \ldots \\ \quad !\ \mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ [AA]_b \\ \mid\ \mathtt{new}\ sk_P.\ \mathtt{new}\ id'.\ \mathtt{new}\ sig'.\ \mathtt{new}\ pic'.\ \ldots \\ \quad \mathtt{let}\ id = \mathtt{choice}[id_0, id']\ \mathtt{in} \\ \quad \mathtt{let}\ sig = \mathtt{choice}[sig_0, sig']\ \mathtt{in} \\ \quad \mathtt{let}\ pic = \mathtt{choice}[pic_0, pic']\ \mathtt{in} \\ \quad \ldots \\ \quad !\ \mathtt{new}\ ksenc.\ \mathtt{new}\ ksmac.\ [AA]_b$$

Encoding anonymity in this way, we have the left side of the choice representing the victim e-passport (with publicly known $id_0, sig_0, pic_0, \ldots$), while the right side represents an unknown to the attacker e-passport (with private $id', sig', pic', \ldots$). Hence, we reduce the problem of testing strong unlinkability to the diff-equivalence of a biprocess. ProVerif proves that the strong anonymity property is satisfied by our models of the *AA* protocol.

Similarly, we can encode the equivalence modelling strong unlinkability as a ProVerif biprocess. Indeed, *AA*

satisfies strong unlinkability if the following equivalence holds:

$$(sk_{DS}; C[[AA]_b]; \Phi_0) \approx (sk_{DS}; C'[[AA]_b]; \Phi_0)$$

This equivalence can be encoded by the following ProVerif biprocess

$$\begin{aligned} AA_{\text{UNLINK}} \stackrel{\text{def}}{=}\ & !\texttt{new}\ sk_{P1}.\texttt{new}\ id_1.\texttt{new}\ sig_1.\texttt{new}\ pic_1.\ldots \\ & !\texttt{new}\ sk_{P2}.\texttt{new}\ id_2.\texttt{new}\ sig_2.\texttt{new}\ pic_2.\ldots \\ & \quad \texttt{let}\ id = \texttt{choice}[id_1, id_2]\ \texttt{in} \\ & \quad \texttt{let}\ sig = \texttt{choice}[sig_1, sig_2]\ \texttt{in} \\ & \quad \texttt{let}\ pic = \texttt{choice}[pic_1, pic_2]\ \texttt{in} \\ & \quad \ldots \\ & \quad \texttt{new}\ ksenc.\ \texttt{new}\ ksmac.\ [AA]_b \end{aligned}$$

Encoding unlinkability in this way, we have that the left side of the choice represents a system where an e-passport (identified by $id_1, sig_1, pic_1, \ldots$) may execute the protocol many times, while the right side represents a system where e-passports execute the protocol at most once ($id_2, sig_2, pic_2, \ldots$ are always different and can be used at most once for the execution of the protocol). Hence, we reduce the problem of testing strong unlinkability to the diff-equivalence of a biprocess. ProVerif proves that the strong unlinkability property is satisfied by our models of the *AA* protocol.

VII. Conclusion

In this paper, we investigate composition results for privacy-type properties expressed using trace equivalence. We have shown that secure protocols can be safely composed. We consider arbitrary equational theories and we assume that protocols may share some usual primitives provided they are tagged. Moreover, we have to assume that the shared keys are not revealed.

When shared keys are kept unknown during the whole execution, we transform any trace of the composition of two protocols under shared secrets into a trace on the composition under no shared secrets. This allows us to go back to the disjoint case for which composition works quite well. However, this transformation does not work anymore as soon as a shared key is revealed even if this key is the public part of an asymmetric key pair, and thus cannot be used to decrypt any ciphertext. Nevertheless, we establish a composition result in this setting by assuming that shared keys are either never revealed or known by the attacker from the beginning.

For the sake of simplicity, we only consider composition assuming that the initial knowledge of the attacker contains a bunch of names as well as some public keys and verification keys. We believe that our result can be extended to allow the attacker to have some non atomic messages in his initial provided that they are well-tagged. Our composition result allows one to consider public shared keys by giving them to the attacker initially (using the frame Φ_0). However, in our setting (and in many others) such a sequence has to be finite and thus we are only able to deal with a bounded number of public shared keys. To relax this hypothesis, we probably need to adapt our model.

For our composition result to work, we have to ensure that protocols used disjoint primitives or at least tagged them. However, real-world security protocols, typically do not use tags, at least not explicitly and not necessarily in the particular way stipulated by our composition result. Thus, it would be interesting to relax this condition. We could for instance use the implicit disjointness criterion developed in [19].

We foresee composition results in a more general way. In this paper, protocols are composed in the sense that they can be executed in parallel in the same environment (*i.e.* under the same composition context). We plan to develop composition results for privacy-type properties where protocols can use other protocols as sub-programs. This will allow us to analyse in a modular way key establishment protocols, and in particular to take into account the *BAC* protocol in our modular analysis of the e-passport application.

Acknowledgements. This work has been partially supported by the EPSRC projects *Verifying Interoperability Requirements in Pervasive Systems* (EP/F033540/1) and *Trust Domains* (TS/I002529/1), as well as the project JCJC VIP ANR-11-JS02-006, and the grant DIGITEO API from Région Île-de-France.

References

[1] D. Goodin, "Defects in e-passports allow real-time tracking," the Register, 26th January 2010.

[2] C. Caldwell, "A pass on privacy?" the New York Times, July 17, 2005.

[3] M. Barbaro and T. Z. Jr., "A face is exposed for AOL searcher No. 4417749," the New York Times, August 9, 2006.

[4] T. Chothia and V. Smirnov, "A traceability attack against e-passports," in *Proc. 14th International Conference on Financial Cryptography and Data Security (FC'10)*, ser. LNCS, vol. 6052. Springer, 2010.

[5] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan, "Analysing unlinkability and anonymity using the applied pi calculus," in *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 2010, pp. 107–121.

[6] J. K. Millen and V. Shmatikov, "Constraint solving for bounded-process cryptographic protocol analysis," in *Proc. 8th Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001, pp. 166–175.

[7] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, 2008.

[8] M. Abadi, B. Blanchet, and C. Fournet, "Just fast keying in the pi calculus," in *Proc. 13th European Symposium on Programming Languages and Systems (ESOP'04)*, ser. LNCS, vol. 2986. Springer, 2004.

[9] 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)*. ACM Press, 2008, pp. 1–10.

[10] A. Armando *et al.*, "The AVISPA Tool for the automated validation of internet security protocols and applications," in *Proc. 17th International Conference on Computer Aided Verification, CAV'2005*, ser. LNCS, vol. 3576. Springer, 2005, pp. 281–285.

[11] A. Tiu and J. E. Dawson, "Automating open bisimulation checking for the spi calculus," in *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 2010, pp. 307–321.

[12] 3GPP, "Technical specification group services and system aspects; 3G security; cryptographic algorithm requirements (release 10)," 3rd Generation Partnership Project, Tech. Rep., 2011, 3GPP TS 33.105 V10.0.0.

[13] 3GPP, "Technical specification group services and system aspects; 3G security; security architecture (release 9)," 3rd Generation Partnership Project, Tech. Rep., 2010, 3GPP TS 33.102 V9.3.0.

[14] ——, "Technical specification group core network and terminals; mobile radio interface layer 3 specification; core network protocols; stage 3 (release 9)," 3rd Generation Partnership Project, Tech. Rep., 2010, 3GPP TS 24.008 V9.4.0.

[15] J. Kelsey, B. Schneier, and D. Wagner, "Protocol interactions and the chosen protocol attack." in *Proc. 5th Inter. Workshop on Security Protocols*, ser. LNCS, vol. 1361. Springer, 1997, pp. 91–104.

[16] J. D. Guttman and F. J. Thayer, "Protocol independence through disjoint encryption." in *Proc. 13th Computer Security Foundations Workshop (CSFW'00)*. IEEE Comp. Soc. Press, 2000, pp. 24–34.

[17] Ş. Ciobâcă and V. Cortier, "Protocol composition for arbitrary primitives," in *Proc. of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 2010, pp. 322–336.

[18] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*. Las Vegas (Nevada, USA): IEEE Computer Society Press, 2001, pp. 136–145.

[19] R. Küsters and M. Tuengerthal, "Composition Theorems Without Pre-Established Session Identifiers," in *Proc. 18th Conference on Computer and Communications Security (CCS 2011)*. ACM Press, 2011, pp. 41–50.

[20] S. Andova, C. Cremers, K. G. Steen, S. Mauw, S. M. lsnes, and S. Radomirović, "A framework for compositional verification of security protocols," *Information and Computation*, vol. 206, no. 2-4, pp. 425–459, 2008.

[21] S. Mödersheim and L. Viganò, "Secure pseudonymous channels," in *Proc. 14th European Symposium on Research in Computer Security (ESORICS'09)*, ser. LNCS, vol. 5789. Springer, 2009, pp. 337–354.

[22] S. Delaune, S. Kremer, and M. D. Ryan, "Composition of password-based protocols," in *Proc. 21st IEEE Computer Security Foundations Symposium (CSF'08)*. IEEE Computer Society Press, 2008, pp. 239–251.

[23] C. Chevalier, S. Delaune, and S. Kremer, "Transforming password protocols to compose," in *Proc. 31st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, ser. Leibniz International Proceedings in Informatics. Leibniz-Zentrum für Informatik, 2011, pp. 204–216.

[24] 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)*. IEEE Computer Society Press, 2004, pp. 186–195.

[25] M. Arapinis, V. Cheval, and S. Delaune, "Verifying privacy-type properties in a modular way," Laboratoire Spécification et Vérification, ENS Cachan, France, Research Report LSV-12-03, Feb. 2012. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/rr-lsv-2012-03.pdf

[26] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, 2001, pp. 104–115.

[27] S. Delaune, S. Kremer, and M. D. Ryan, "Verifying privacy-type properties of electronic voting protocols," *Journal of Computer Security*, no. 4, pp. 435–487, Jul. 2008.

[28] M. Bruso, K. Chatzikokolakis, and J. den Hartog, "Formal verification of privacy for RFID systems," in *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*. IEEE Computer Society Press, 2010.

[29] M. Abadi and V. Cortier, "Deciding knowledge in security protocols under equational theories," *Theoretical Computer Science*, vol. 387, no. 1-2, pp. 2–32, 2006.

[30] "ISO 15408-2: Common Criteria for Information Technology Security Evaluation - Part 2: Security functional components," ISO/IEC, Final draft, July 2009.

[31] V. Cortier and S. Delaune, "Safely composing security protocols," *Formal Methods in System Design*, vol. 34, no. 1, pp. 1–36, Feb. 2009.

[32] "PKI for machine readable travel documents offering ICC read-only access," International Civil Aviation Organization, Tech. Rep., 2004.