A typing result for trace inclusion (for pair and symmetric encryption only)

Véronique Cortier, Antoine Dallon, Stéphanie Delaune

A typing result for trace inclusion (for pair and symmetric encryption only)

Véronique Cortier Antoine Dallon Stéphanie Delaune

October 12, 2017

Abstract

Privacy-type properties such as vote secrecy, anonymity, or untraceability are typically expressed using the notion of trace equivalence in a process algebra that models security protocols. In this paper, we propose some results to reduce the search space when we are looking for an attack regarding trace equivalence. Our work is strongly inspired from [10], which establishes that, if there is a witness of non trace equivalence, then there is one that is well-typed.

Our main contribution is to establish a similar result for trace inclusion. Our motivation is twofolds: first, this small attack property is needed for proving soundness of the tool SatEquiv [13]. Second, we revisit the proof in order to simplify it. Specifically, we show two results. First, if there is a witness of non-inclusion then there is one that is well-typed. We establish this result by providing a decision procedure for trace inclusion similar to the one proposed in [10] for trace equivalence. We also show that we can reduce the search space when considering the notion of static inclusion. Acutally, if there is a witness of static non-inclusion there is one of a specific shape.

Even if our setting slightly differs from the one considered in [10], our proofs essentially follow the same ideas than the existing proof for trace equivalence. Nevertheless, we hope that this proof will be easier to extend to other primitives such as asymmetric encryption or signatures.

1 Introduction

Privacy properties such as untraceability, vote secrecy, or anonymity are typically expressed as behavioural equivalence (*e.g.* [6, 3]). For example, the anonymity of Bob is typically expressed by the fact that an adversary should not distinguish between the situation where Bob is present and the situation where Alice is present. Formally, the behaviour of a protocol can be modelled through a process algebra such as CSP or the pi calculus, enriched with terms to represent cryptographic messages. Then indistinguishability can be modelled through various behavioural equivalences. We focus here on trace equivalence, denoted $\approx$. Checking for privacy then amounts into checking for trace equivalence between processes, which is of course undecidable in general. Even in the case of a bounded number of sessions, there are few decidability results and the associated decision procedures are complex [5, 16, 8].

Our contributions are two simplification results in the same spirit than the one provided in [10]. We reduce the search space for attacks: if there is an attack, then there exists a well-typed attack. More formally, we show that if there is a witness (*i.e.* a trace) that $P \not\sqsubseteq Q$ then there exists a witness which is well-typed w.r.t. P provided that P is type-compliant and Q is an action-deterministic process. We also show that we can reduce the search space for static inclusion: it is possible to consider only analysis rules. In [10], this result was used to prove that there is a well-typed attack whenever there is an attack. This is no longer the case here. Instead, the two results are now independent and of general interest for subsequent results.

We only consider processes without replication but the result can be easily extended to processes with replications as it was done in [10]. For simplicity, we prove this typing result for the case of symmetric encryption and concatenation but our goal when designing this new proof is to extend it to other standard cryptographic primitives.

As already mentionned, our result is actually a variation of the typing result stated and proved in [10]. We highlight below the main differences between these two results:

1. We consider here a setting that is closer to the one we introduced in [13]. In particular, the properties of our primitives are reflected through a rewriting system in which reductions can only occur if subterms are messages.
2. We establish the typing result for trace inclusion instead of trace equivalence and we slightly modify the algorithm to obtain a complete procedure in that case.
3. We consider processes that are action-deterministic. This is slightly stronger than the notion of determinacy introduced in [10]. We want to emphasize that the notion of determinacy used in [10] is actually too weak to ensure the existence of a unique frame ψ in the algorithm provided in [10]. Such an hypothesis is therefore also needed in [10].

2 Model for security protocols

Security protocols are modelled through a process algebra inspired from [1] that manipulates terms. Actually, we consider here a variant of the calculus provided in [10].

2.1 Messages

We assume an infinite set $\mathcal{N}$ of *names*, which are used to represent keys and nonces, and two infinite disjoint sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. The variables in $\mathcal{W}$ intuitively refer to variables used to store messages learnt by the attacker. We consider the following sets of function symbols:

$$\Sigma_{\mathsf{c}} = \{\mathsf{senc}, \langle\ \rangle\} \qquad \Sigma_{\mathsf{d}} = \{\mathsf{sdec}, \mathsf{proj}_1, \mathsf{proj}_2\} \qquad \Sigma_{\mathsf{std}} = \Sigma_{\mathsf{c}} \cup \Sigma_{\mathsf{d}}$$

The symbols sdec and senc of arity 2 represent symmetric decryption and encryption. Pairing is modelled using a symbol of arity 2, denoted $\langle\ \rangle$, and projection functions are denoted proj_1 and proj_2. The symbols in Σ_{c} are *constructors* whereas those in Σ_{d} are *destructors*. We further assume an infinite sets of *constant symbols* Σ_0 to represent atomic data known to the attacker.

Given a set of A of atoms (*i.e.* names, variables, and constants), and a signature $\mathcal{F}$, we denote by $\mathcal{T}(\mathcal{F}, \mathsf{A})$ the set of terms built from symbols in $\mathcal{F}$, and atoms in A. We denote $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \mathsf{A})$ the set of terms that only contains atoms in key position. More formally, this set is generated by the following grammar:

$$t, t_1, t_2 := \mathsf{senc}(t, a_1) \mid \langle t_1, t_2\rangle \mid a_2 \qquad \text{with } a_1, a_2 \in \mathsf{A}$$

Terms in $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$ are called *messages*. An attacker builds his own messages by applying functions to terms he already knows. Formally, a computation done by the attacker is modelled by a term, called a *recipe*, built on signature Σ_{std} using (public) constants in Σ_0 as well as variables in $\mathcal{W}$, *i.e.* a term $R \in \mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \mathcal{W})$. Note that such a term does not contain any name.

We denote $vars(u)$ the set of variables that occur in u. The application of a substitution σ to a term u is written $u\sigma$, and we denote $dom(\sigma)$ its *domain*. Two terms u_1 and u_2 are *unifiable* when there exists σ such that $u_1\sigma = u_2\sigma$.

The properties of our primitives are reflected through the following rewriting rules where reduction only occurs if the variables are instanciated by messages (and atoms in key position).

$$\mathsf{sdec}(\mathsf{senc}(x, y), y) \ \to\ x \quad \mathsf{proj}_1(\langle x, y\rangle) \ \to\ x \quad \mathsf{proj}_2(\langle x, y\rangle) \ \to\ y$$

More formally, a term u can be rewritten in v if there is a position p in u, and a rewriting rule $\mathsf{g}(t_1, \ldots, t_n) \to t$ such that $u|_p = \mathsf{g}(t_1, \ldots, t_n)\theta$ as well as $t\theta$ are messages. This assumption slightly differs from [10]: whenever an inner decryption/projection fails then the overall evaluation fails. Our rewriting system is convergent, and we denote $u{\downarrow}$ the *normal form* of a given term u.

Example 1. *Let $s, k \in \mathcal{N}$, and $u = \mathsf{senc}(s, k)$. The term $\mathsf{sdec}(u, k)$ models the application of the decryption algorithm on the message u using k. We have that $\mathsf{sdec}(u, k){\downarrow} = s$.*

2.2 Protocols

Our process algebra is inspired from the applied pi calculus [1]. We do not consider else branches. Actually we do not have conditional. Instead, equality tests are performed through pattern-matching. We do not consider replication wither but our reduction results easily extend to processes with replication as explained in [10].

2.2.1 Syntax.

Let $\mathcal{C}h$ be an infinite set of *channels*. We consider processes built using the following grammar:

$$\begin{array}{rcll} P, Q := & 0 & & \text{null process} \\ & \mid & \mathsf{in}(c, u).P & \text{input} \\ & \mid & \mathsf{out}(c, u).P & \text{output} \\ & \mid & (P \mid Q) & \text{parallel} \end{array}$$

where $u \in \mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$, and $c \in \mathcal{C}h$.

The process 0 does nothing. The process "$\mathsf{in}(c, u).P$" expects a message m of the form u on channel c and then behaves like $P\sigma$ where σ is a substitution such that $m = u\sigma$. The process "$\mathsf{out}(c, u).P$" emits u on channel c, and then behaves like P. The variables that occur in u are instantiated when the evaluation takes place. The process $P \mid Q$ runs P and Q in parallel.

For the sake of clarity, we may omit the null process. We also assume that processes are *name and variable distinct*, *i.e.* any name and variable is at most bound once. We write $fv(P)$ for the set of *free variables* that occur in P, *i.e.* the set of variables that are not in the scope of an input.

Definition 1. *A* protocol *P is a process such that P is ground,* i.e. *$fv(P) = \emptyset$; and P is name and variable distinct.*

Example 2. *The Otway-Rees protocol [11] is a key distribution protocol using symmetric encryption and a trusted server. It can be described informally as follows:*

1. $A \to B:\ M, A, B, \{N_a, M, A, B\}_{K_{as}}$
2. $B \to S:\ M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}$
3. $S \to B:\ M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
4. $B \to A:\ M, \{N_a, K_{ab}\}_{K_{as}}$

where $\{m\}_k$ denotes the symmetric encryption of a message m with key k, A and B are agents trying to authenticate each other, S is a trusted server, K_{as} (resp. K_{bs}) is a long term key shared between A and S (resp. B and S), N_a and N_b are nonces generated by A and B, K_{ab} is a session key generated by S, and M is a session identifier.

We propose a modelling of the Otway-Rees protocol in our formalism. Below, k_{as}, k_{bs}, m, n_a, n_b, k_{ab} are names, whereas a and b are constants from Σ_0. We denote by $\langle x_1, \ldots, x_{n-1}, x_n\rangle$ the term $\langle x_1, \langle \ldots \langle x_{n-1}, x_n\rangle\rangle\rangle$.

$$P_{\mathsf{OR}} = P_A \mid\ P_B \mid\ P_S$$

where the processes P_A, P_B, P_S are given below.

$$P_A = \mathsf{out}(c_A, \langle m, \mathsf{a}, \mathsf{b}, \mathsf{senc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as})\rangle).\mathsf{in}(c_A, \langle m, \mathsf{senc}(\langle n_a, x_{ab}\rangle, k_{as})\rangle);$$

$$\begin{array}{l} P_B = \mathsf{in}(c_B, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}\rangle).\mathsf{out}(c_B, \langle y_m, \mathsf{a}, \mathsf{b}, y_{as}, \mathsf{senc}(\langle n_b, y_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle). \\ \qquad \mathsf{in}(c_B, \langle y_m, z_{as}, \mathsf{senc}(\langle n_b, y_{ab}\rangle, k_{bs})\rangle).\mathsf{out}(c_B, \langle y_m, z_{as}\rangle); \\ P_S = \mathsf{in}(c_S, \langle z_m, \mathsf{a}, \mathsf{b}, \mathsf{senc}(\langle y_{na}, y_m, \mathsf{a}, \mathsf{b}\rangle, k_{as}), \mathsf{senc}(\langle y_{nb}, y_m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})). \\ \qquad \mathsf{out}(c_S, \langle y_m, \mathsf{senc}(\langle y_{na}, k_{ab}\rangle, k_{as}), \mathsf{senc}(\langle y_{nb}, k_{ab}\rangle, k_{bs})\rangle); \end{array}$$

$$\textsc{In} \quad (\mathsf{in}(c,u).P \cup \mathcal{P};\ \phi;\ \sigma) \xrightarrow{\mathsf{in}(c,R)} (P \cup \mathcal{P};\ \phi;\ \sigma \uplus \sigma_0)$$ where R is a recipe such that $R\phi\downarrow$ is a message, and $R\phi\downarrow = (u\sigma)\sigma_0$ for σ_0 with $dom(\sigma_0) = vars(u\sigma)$.

$$\textsc{Out} \quad (\mathsf{out}(c,u).P \cup \mathcal{P};\ \phi;\ \sigma) \xrightarrow{\mathsf{out}(c,\mathsf{w})} (P \cup \mathcal{P};\ \phi \cup \{\mathsf{w} \triangleright u\sigma\};\ \sigma)$$ with w a fresh variable from $\mathcal{W}$, and $u\sigma$ is a message.

$$\textsc{Null} \quad (0 \cup \mathcal{P};\ \phi;\ \sigma) \xrightarrow{\tau} (\mathcal{P};\ \phi;\ \sigma)$$

$$\textsc{Par} \quad ((P \mid Q) \cup \mathcal{P};\ \phi;\ \sigma) \xrightarrow{\tau} (P \cup Q \cup \mathcal{P};\ \phi;\ \sigma)$$

Figure 1: Semantics for processes

2.2.2 Semantics.

The operational semantics of a process is defined using a relation over configurations. A *configuration* is a tuple $(\mathcal{P};\ \phi;\ \sigma)$ such that:

- $\mathcal{P}$ is a multiset of ground processes.
- $\phi = \{\mathsf{w}_1 \triangleright m_1, \ldots, \mathsf{w}_n \triangleright m_n\}$ is a *frame*, *i.e.* a substitution where $\mathsf{w}_1, \ldots, \mathsf{w}_n$ are variables in $\mathcal{W}$, and $m_1, \ldots, m_n$ are messages.
- σ is a substitution such that $dom(\sigma) \subseteq \mathcal{X}$, and $img(\sigma) \subseteq \mathcal{T}_0(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \mathcal{N})$.

Intuitively, $\mathcal{P}$ represents the processes that still remain to be executed; and ϕ represents the sequence of messages that have been learnt so far by the attacker. We often write P instead of $(P;\ \emptyset;\ \emptyset)$, and $P \cup \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$. The operational semantics of a process is induced by the relation $\xrightarrow{\alpha}$ over configurations defined in Figure 1.

The first rule (In) allows the attacker to send to some process a term built from publicly available terms and symbols. The second rule (Out) corresponds to the output of a term by some process: the corresponding term is added to the frame of the current configuration, which means that the attacker can now access the sent term. Note that the term is outputted provided that it is a message. In case the evaluation of the term yields e.g. an encryption with a non atomic key, the evaluation fails and there is no output. The two remaining rules are quite standard and are unobservable (τ action) from the point of view of the attacker.

The relation $\xrightarrow{\alpha_1 \ldots \alpha_n}$ between configurations (where $\alpha_1 \ldots \alpha_n$ is a sequence of actions) is defined as the transitive closure of $\xrightarrow{\alpha}$. Given a sequence of observable actions tr, we write $K \overset{\mathsf{tr}}{\Longrightarrow} K'$ when there exists a sequence $\alpha_1 \ldots \alpha_n$ such that $K \xrightarrow{\alpha_1 \ldots \alpha_n} K'$ and tr is obtained from $\alpha_1 \ldots \alpha_n$ by erasing all occurrences of τ.

Definition 2. *Given a configuration $\mathcal{K} = (\mathcal{P};\ \phi;\ \sigma)$, we denote $\mathsf{trace}(\mathcal{K})$ the set of traces defined as follows:*

$$\mathsf{trace}(\mathcal{K}) = \{(\mathsf{tr}, \phi') \mid \mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P};\ \phi';\ \sigma') \text{ for some configuration } (\mathcal{P};\ \phi';\ \sigma')\}.$$

We may note that, by definition of $\mathsf{trace}(\mathcal{K})$, $\mathsf{tr}\phi\downarrow$ only contains terms from $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$.

Example 3. *Consider the following sequence* tr*:*

$$\mathsf{tr} = \mathsf{out}(c_A, \mathsf{w}_1).\mathsf{in}(c_B, \mathsf{w}_1).\mathsf{out}(c_B, \mathsf{w}_2).\mathsf{in}(c_B, R_0).\mathsf{out}(c_B, \mathsf{w}_3).\mathsf{in}(c_A, \mathsf{w}_3)$$

where $R_0 = \langle \mathsf{proj}_{1/5}(\mathsf{w}_2), \mathsf{proj}_{4/5}(\mathsf{w}_2), \mathsf{proj}_{5/5}(\mathsf{w}_2) \rangle$, and $\mathsf{proj}_{i/5}$ is used as a shortcut to extract the i^{th} component of a 5-uplet. This a sequence of actions yields the frame ϕ defined as follows:

$$\phi = \{\mathsf{w}_1 \triangleright \langle m, \mathsf{a}, \mathsf{b}, t_{\mathsf{senc}} \rangle, \mathsf{w}_2 \triangleright \langle m, \mathsf{a}, \mathsf{b}, t_{\mathsf{senc}}, \mathsf{senc}(\langle n_b, m, \mathsf{a}, \mathsf{b} \rangle, k_{bs}) \rangle, \mathsf{w}_3 \triangleright \langle m, t_{\mathsf{senc}} \rangle\}.$$

where $t_{\mathsf{senc}} = \mathsf{senc}(\langle n_a, m, \mathsf{a}, \mathsf{b} \rangle, k_{as})$.

We have that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P_{\mathsf{OR}})$. *The first three actions actually correspond to the expected execution of the protocol. Then, the agent who plays* P_B *accepts as input the message built using* R_0, i.e.

$$u = \langle m, \mathsf{senc}(\langle n_a, m, \mathsf{a}, \mathsf{b}\rangle, k_{as}), \mathsf{senc}(\langle n_b, m, \mathsf{a}, \mathsf{b}\rangle, k_{bs})\rangle.$$

Indeed, this message has the expected form. At this stage, the agent who plays P_B *is waiting for a message of the form:* $u_0 = \langle m, z_{as}, \mathsf{senc}(\langle n_b, y_{ab}\rangle, k_{bs})\rangle$. *The substitution* $\sigma = \{z_{as} \triangleright t_{\mathsf{senc}},\ y_{ab} \triangleright \langle m, \mathsf{a}, \mathsf{b}\rangle\}$ *is such that* $u = u_0\sigma$. *Once this input has been done, a message is outputted (action* $\mathsf{out}(c_B, \mathsf{w}_3)$*) and given as input to* P_A *(action* $\mathsf{in}(c_A, \mathsf{w}_3)$*).*

Note that, at the end of the execution, A and B share a key but it is not the expected one, i.e. *one freshly generated by the trusted server, but* $\langle m, \mathsf{a}, \mathsf{b}\rangle$. *This execution corresponds to (a variant of) a known attack on the Otway-Rees protocol [11].*

2.3 Action-determinism

As mentioned in introduction, we require processes to be deterministic. We consider a definition similar to the one introduced in [4]. This condition is actually stronger than the one considered in [10]. This is actually mandatory to ensure the uniqueness of the frame once a (symbolic) trace has been fixed. This is therefore a missing assumption of [10].

Definition 3. *A configuration* $\mathcal{K}$ *is* action-deterministic *if whenever* $\mathcal{K} \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; \sigma)$, *and* $\alpha.P$ *and* $\beta.Q$ *are two elements of* $\mathcal{P}$ *with* α, β *instructions of the form* $\mathsf{in}(c, u)$, $\mathsf{out}(c, u)$ *then either the underlying channels c differ or the instructions are not of the same nature (that is,* α, β *are not both an input, nor both an output).*

A protocol P *is action-deterministic if* $\mathcal{K} = (P; \emptyset; \emptyset)$ *is action-deterministic.*

For such protocols, the attacker knowledge is entirely determined by its interaction with the protocol.

Lemma 1. *Let* $\mathcal{K}$ *be an action-deterministic configuration such that* $\mathcal{K} \stackrel{\mathsf{tr}}{\Longrightarrow} \mathcal{K}_1$ *and* $\mathcal{K} \stackrel{\mathsf{tr}}{\Longrightarrow} \mathcal{K}_2$ *for some* tr, $\mathcal{K}_1 = (\mathcal{P}_1; \phi_1; \sigma_1)$, *and* $\mathcal{K}_2 = (\mathcal{P}_2; \phi_2; \sigma_2)$. *We have that* $\phi_1 = \phi_2$.

2.4 Trace equivalence

Intuitively, two protocols are equivalent if they cannot be distinguished by any attacker. Trace equivalence can be used to formalise many interesting security properties, in particular privacy-type properties, such as those studied for instance in [6]. We first introduce a notion of intruder's knowledge well-suited to cryptographic primitives for which the success of decryption is visible.

Definition 4. *Two frames* ϕ_1 *and* ϕ_2 *are in* static inclusion, *written* $\phi_1 \sqsubseteq_s \phi_2$, *when we have that* $dom(\phi_1) = dom(\phi_2)$, *and:*

- *for any recipe* R, *we have that* $R\phi_1{\downarrow} \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$ *implies that* $R\phi_2{\downarrow} \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$*; and*
- *for any recipes* R_1 *and* R_2 *such that* $R_1\phi_1{\downarrow}, R_2\phi_1{\downarrow} \in \mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$, *we have that* $R_1\phi_1{\downarrow} = R_2\phi_1{\downarrow}$ *implies* $R_1\phi_2{\downarrow} = R_2\phi_2{\downarrow}$.

They are in static equivalence, *written* $\phi_1 \sim_s \phi_2$, *if* $\phi_1 \sqsubseteq_s \phi_2$, *and* $\phi_2 \sqsubseteq_s \phi_1$.

Intuitively, two frames are statically equivalent if an attacker cannot see the difference between the two situations they represent. If some computation fails in ϕ_1 for some recipe R, *i.e.* $R\phi_1{\downarrow}$ is not a message, it should fail in ϕ_2 as well. Moreover, ϕ_1 and ϕ_2 should satisfy the same equalities. In other words, the ability of the attacker to distinguish whether a recipe R produces a message, or whether two recipes R_1, R_2 produce the same message should not depend on the frame.

Example 4. *Consider* $\phi_1 = \phi \cup \{\mathsf{w}_4 \triangleright \langle m, \mathsf{a}, \mathsf{b}\rangle\}$*, and* $\phi_2 = \phi \cup \{\mathsf{w}_4 \triangleright n\}$ *where* n *is a name. Let* $R = \mathsf{proj}_1(\mathsf{w}_4)$*. We have that* $R\phi_1\downarrow = m \in \mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N})$*, but* $R\phi_2\downarrow = \mathsf{proj}_1(n) \notin \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N})$*, hence* $\phi_1 \not\sim_s \phi_2$*. This non static equivalence can also be witnessed by considering the recipes* $R_1 = \langle \mathsf{proj}_1(\mathsf{w}_3), \mathsf{a}, \mathsf{b}\rangle$ *and* $R_2 = \mathsf{w}_4$*. We have that* $R_1\phi_1\downarrow, R_2\phi_1\downarrow \in \mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N})$*, and* $R_1\phi_1\downarrow = R_2\phi_1\downarrow$ *whereas* $R_1\phi_2\downarrow \neq R_2\phi_2\downarrow$*.*

We can now define equivalence for processes. Intuitively, two protocols are *trace equivalent* if, however they behave, the resulting sequences of messages observed by the attacker are in static equivalence.

Definition 5. *A protocol* P *is* trace included *in a protocol* Q*, written* $P \sqsubseteq Q$*, if for every* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$*, there exists* $(\mathsf{tr}', \phi') \in \mathsf{trace}(Q)$ *such that* $\mathsf{tr} = \mathsf{tr}'$ *and* $\phi \sqsubseteq_s \phi'$*. The protocols* P *and* Q *are* trace equivalent*, written* $P \approx Q$*, if* $P \sqsubseteq Q$ *and* $Q \sqsubseteq P$*.*

This notion of equivalence does not coincide in general with the usual notion of trace equivalence e.g. in [9]. It is actually coarser since we simply require the resulting frames to be in static inclusion ($\phi \sqsubseteq_s \psi$) instead of static equivalence ($\phi \sim_s \psi$). However, these two notions actually coincide (see [7]) for the class of action-deterministic processes that we consider in this paper.

As illustrated by the following example, restricting messages to only contain atoms in key position also provides the adversary with more comparison power when variables occurred in key position in the protocol.

Example 5. *Let* $n, k \in \mathcal{N}$ *and consider the protocol* $P = \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{senc}(n, k))$ *as well as the protocol* $Q = \mathsf{in}(c, x).\mathsf{out}(c, \mathsf{senc}(\mathsf{senc}(n, x), k))$*. An attacker may distinguish between* P *and* Q *by sending a non atomic data and observing whether the process can emit.* Q *will not be able to emit since its first encryption will fail. This attack would not have been detected if arbitrary terms were allowed in key position.*

Assume given a protocol P and an action-deterministic protocol Q such that $P \not\sqsubseteq Q$. A *witness of non-inclusion* is an execution $P \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; \sigma)$ such that:

- either there does not exist ϕ' such that $(\mathsf{tr}, \phi') \in \mathsf{trace}(Q)$,
- or such a ϕ' exists and $\phi \not\sqsubseteq_s \phi'$.

Note that when a protocol P is action-deterministic, once the sequence tr is fixed, all the frames reachable through tr are actually equal. This ensures the unicity of ϕ', if it exists (see Lemma 1). Moreover, when the underlying execution is clear from the context, we sometimes simply say that tr is a witness of non-inclusion.

Example 6. *We wish to check strong secrecy of the exchanged key received by the agent A for the Otway-Rees protocol. A way of doing so is to check that* $P^1_{\mathsf{OR}} \approx P^2_{\mathsf{OR}}$ *where the two protocols are defined as follows:*

- P^1_{OR} *is as* P_{OR} *but we add the instruction* $\mathsf{out}(c_A, x_{ab})$ *at the end of process* P_A*;*
- P^2_{OR} *is as* P_{OR} *but we add the instruction* $\mathsf{out}(c_A, n)$ *at the end of process* P_A*.*

The idea is to check whether an attacker can see the difference between the session key obtained by A and a fresh nonce.

As already suggested by the scenario described in Example 3, the secrecy (and so the strong secrecy) of the key received by A is not preserved. More precisely, consider the sequence $\mathsf{tr}' = \mathsf{tr}.\mathsf{out}(c_A, \mathsf{w}_4)$ *where* tr *is as in Example 3. In particular,* $(\mathsf{tr}', \phi_1) \in \mathsf{trace}(P^1_{\mathsf{OR}})$ *and* $(\mathsf{tr}', \phi_2) \in \mathsf{trace}(P^2_{\mathsf{OR}})$ *with* $\phi_1 = \phi \cup \{\mathsf{w}_4 \triangleright \langle m, \mathsf{a}, \mathsf{b}\rangle\}$ *and* $\phi_2 = \phi \cup \{\mathsf{w}_4 \triangleright n\}$*. As described in Example 4,* $\phi_1 \not\sim_s \phi_2$ *and thus* tr' *is a witness of non-equivalence for* P^1_{OR} *and* P^2_{OR}*. This witness is actually a variant of a known attack on the Otway-Rees protocol [11].*

3 Existence of a well-typed witness for an attack

In this section, we present our main contribution: a simplification result that reduces the search space for attacks. Roughly, when looking for a witness of non-inclusion when checking whether $P \sqsubseteq Q$, we can restrict ourselves to consider well-typed executions, i.e. executions of the form $P \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; \sigma)$ such that σ is well-typed meaning that every variable of its domain has the same type as its image. This results holds for a general class of typing systems and as soon as the protocol P is type-compliant and the protocol Q is action-deterministic. We first explain these hypotheses and then we state our general simplification result (see Theorem 1).

3.1 Type compliance

Our simplification result holds for a general class of typing systems: we simply require that types are preserved by unification and application of substitutions. These operations are indeed routinely used in decision procedures.

Definition 6. *A* typing system *is a pair $(\mathcal{T}, \delta)$ where $\mathcal{T}$ is a set of elements called* types*, and δ is a function mapping terms $t \in \mathcal{T}(\Sigma_c, \Sigma_0 \cup \mathcal{N} \cup \mathcal{X})$ to types τ in $\mathcal{T}$ such that:*

- *if t is a term of type τ and σ is a* well-typed *substitution,* i.e. *every variable of its domain has the same type as its image, then $t\sigma$ is of type τ,*
- *for any terms t and t' with the same type,* i.e. *$\delta(t) = \delta(t')$ and which are unifiable, their most general unifier ($mgu(t, t')$) is well-typed.*

We further assume the existence of an infinite number of constants in Σ_0 (resp. variables in $\mathcal{X}$, names in $\mathcal{N}$) of any type.

A straightforward typing system is when all terms are of a unique type, say Msg. Of course, our typing result would then be useless to reduce the search space for attacks. Which typing system shall be used typically depends on the protocols under study.

Our main assumption on the typing of protocols is that any two unifiable encrypted subterms are of the same type. We write $St(t)$ for the set of *(syntactic) subterms* of a term t, and $ESt(t)$ the set of its *encrypted subterms*. More formally, we define:

$$ESt(t) = \{u \in St(t) \mid u \text{ is of the form } \mathsf{senc}(u_1, u_2)\}.$$

We extend this notion to sets/sequences of terms, and to protocols as expected.

Definition 7. *A protocol P is* type-compliant *w.r.t. a typing system $(\mathcal{T}, \delta)$ if for every $t, t' \in ESt(P)$ we have that:*

$$t \text{ and } t' \text{ unifiable implies that } \delta(t) = \delta(t').$$

Consider a protocol P that is type-compliant w.r.t. a typing system $(\mathcal{T}_P, \delta_P)$, an execution $P \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi'; \sigma')$ is *well-typed* if σ' is a well-typed substitution. When the underlying execution is clear from the context, we sometimes say that a trace $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ is well-typed meaning that its underlying execution $P \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; \sigma)$ is well-typed.

3.2 Main result

Given a protocol P, we denote Σ_P the set of constants from Σ_0 that occur in P. We are now ready to state our main result: if there is an attack, then there is a well-typed attack.

Theorem 1. *Let P be a protocol type-compliant w.r.t. $(\mathcal{T}_P, \delta_P)$ and Q be another protocol that is action-deterministic. We have that $P \not\sqsubseteq Q$ if, and only if, there exists a witness of this non-inclusion that is well-typed w.r.t. $(\mathcal{T}_P, \delta_P)$.*

This theorem is actually a consequence of the algorithm presented in Section 4 that returns a well-typed witness of non-inclusion.

4 A type preserving decision algorithm for trace inclusion

We provide a procedure that decides whether two processes are in trace inclusion and if not, returns a well typed witness of non inclusion. One key ingredient of our procedure (described in Section 4.2) is carefully designed to only allow unification between encrypted subterms. We rely on two main results.

1. We rely on an existing algorithm for reachability that only performs unification between encrypted subterms. Most of the existing algorithms (*e.g.* [15, 12, 17]) were not designed with such a goal in mind. However, in the case of the algorithm given in [12], it has already been shown how it can be turned into one that only considers unification between encrypted subterms [14].

2. We establish that it is actually sufficient to consider unification between encrypted subterms when dealing with static inclusion.

This approach allows us to provide a well-typed witness of non-inclusion when protocol P is type-compliant and protocol Q is action-deterministic.

4.1 Some preliminaries

We introduce a special set Σ_{fresh} of constants that are fresh and that will be used in our algorithm. Each constant is either seen as an atom or not. We assume that Σ_{fresh} contains an arbitrary number of atomic (resp. non atomic) constants of any type.

We sometimes write *symbolic* frame, or *symbolic* trace to emphasize the fact that those traces may use constants from Σ_{fresh}.

Definition 8. *A first-order trace* tr_s *is of the form* $\mathsf{io}_1(c_1, u_1) \dots \mathsf{io}_n(c_n, u_n)$ *where* $\mathsf{io}_i \in \{\mathsf{in}, \mathsf{out}\}$, $c_i \in \mathcal{C}h$, *and* $u_i \in \mathcal{T}_0(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \mathcal{N} \cup \Sigma_{\mathsf{fresh}})$. *Moreover, we assume that any constant from* Σ_{fresh} *occurs first in an input action.*

Such a first-order trace tr_s *is* valid *if for all* $1 \leq i_0 \leq n$, *whenever,* $io_{i_0} = \mathsf{in}(c_{i_0}, u_{i_0})$, *we have that* $R\phi_s{\downarrow} = u_{i_0}$ *for some* $R \in \mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \mathcal{W} \cup \Sigma_{\mathsf{fresh}})$ *where* $\phi_s = \{\mathsf{w}_1 \triangleright u_{i_1}, \dots, \mathsf{w}_\ell \triangleright u_{i_\ell}\}$ *and* $i_1 \dots i_\ell$ *is the increasing sequence of indices that captures all the outputs of terms of the trace* tr_s *up to index* i_0, i.e. *such that* $\{i_1, \dots, i_\ell\} = \{j \mid \mathsf{io}_j = \mathsf{out} \text{ and } j \leq i_0\}$.

Given a protocol P, the set of *symbolic traces* $\mathsf{trace}_s(P)$ of a protocol P is defined as follows:

$$\mathsf{trace}_s(P) = \{\mathsf{tr}_s \mid P \xrightarrow{\mathsf{tr}_s}_s Q \text{ for some } Q \}$$

where $\xrightarrow{\mathsf{tr}_s}_s$ is transitive closure of the relation $\xrightarrow{\alpha_s}_s$ defined below:

$$\begin{array}{lrcl}
(\textsc{In}_s) & \mathsf{in}(c,u).P \cup \mathcal{P} & \xrightarrow{\mathsf{in}(c,u)}_s & P \cup \mathcal{P} \\
(\textsc{Out}_s) & \mathsf{out}(c,u).P \cup \mathcal{P} & \xrightarrow{\mathsf{out}(c,u)}_s & P \cup \mathcal{P} \\
(\textsc{Null}_s) & 0 \cup \mathcal{P} & \xrightarrow{\tau} & \mathcal{P} \\
(\textsc{Par}_s) & P \mid Q & \xrightarrow{\tau} & P \cup Q
\end{array}$$

Definition 9 (first-order substitution associated to θ through ϕ_S)**.** *Let* ϕ_S *be a symbolic frame. Let* θ *be a substitution from constants of* Σ_{fresh} *to recipes without fresh constants such that the variables of* $vars(c\theta)$ *are only those that appear before the first occurence of* c *in* ϕ_S *for each* $c \in \Sigma_{\mathsf{fresh}}$. *Then we define the* first-order substitution λ associated to θ through ϕ_S *as the only substitution such that:*

- $dom(\lambda) = dom(\theta)$
- *for all* $c \in dom(\lambda)$, *we have that* $c\lambda = (c\theta)(\phi_S\lambda){\downarrow}$

Lemma 2. *Let* $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. *There exists* $\mathsf{tr}_0 \in \mathsf{trace}_s(P)$, $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}(P)$, *a substitution* σ_0 *which is the mgu of some pairs of encrypted subterms occurring in* tr_0, *a bijective mapping* ρ *from variables in* $\mathsf{tr}_0\sigma_0$ *to fresh constants, and two substitutions* θ *and* λ *such that:*

1. $(\mathsf{tr}_0\sigma_0)\rho = \mathsf{tr}_S\phi_S{\downarrow}$;
2. $(\mathsf{tr}_0\sigma_0)\rho\lambda = \mathsf{tr}\phi{\downarrow}$;
3. *for any* $x \in vars(\mathsf{tr}_0\sigma_0)$, *we have that* $x\rho$ *is an atomic constant if, and only if* $x\rho\lambda$ *is atomic;*
4. $dom(\theta) = dom(\lambda)$ *and* $c\theta = R_c$ *for some* $R_c \in \mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \{\mathsf{w}_1, \ldots, \mathsf{w}_i\})$ *such that* $R_c\phi{\downarrow} = c\lambda$ *and* i *is the number of outputs that occur in* $(\mathsf{tr}_s\sigma_0)\rho$ *before the first occurrence of an input that contains the fresh constant* c.
5. $(\mathsf{tr}_S\theta)(\phi_S\lambda){\downarrow} = \mathsf{tr}\phi{\downarrow}$;
6. *for any encrypted subterms* t_1, t_2 *occurring in* $\mathsf{tr}_0\sigma_0$, *we have that* $t_1\rho\lambda = t_2\rho\lambda$ *implies that* $t_1 = t_2$.

Moreover, λ *is the first-order substitution associated to* θ *through* ϕ_S.

A weaker version of this result (items $1-5$) can be deduced from the decision procedure provided in [14] for a slightly different setting. In particular, in [14], they consider non-atomic key. We explain in Appendix A how to adapt this result in presence of atomic keys, and we show how to obtain item 6.

4.2 Our algorithm for trace inclusion

Our algorithm $\mathcal{A}$ takes as input two protocols P and Q and returns *yes* when $P \sqsubseteq Q$; and a minimal (in term of number of actions) witness tr of non-inclusion otherwise.

Our algorithm $\mathcal{A}(P, Q)$. It consists of the following steps starting at level 1 until ℓ where ℓ denotes the maximal length (*i.e.* number of actions) of a trace in $\mathsf{trace}_s(P)$. If nothing has been returned yet (*i.e.* when the iteration steps for level ℓ has been done), then it returns *yes*, *i.e.* P is trace-included in Q.

Iteration steps for level n:

1. Consider every symbolic trace $\mathsf{tr}_0 \in \mathsf{trace}_s(P)$ of length n.
2. Let σ_0 be the most general unifier of some pairs of encrypted subterms occurring in tr_0.
3. Let ρ be a bijective renaming mapping variables occurring in $\mathsf{tr}_0\sigma_0$ to fresh constants from Σ_{fresh}. The choice of the constant does not matter but we have to consider the case where a variable is mapped to an atomic constant or a non-atomic constant.
4. Check whether $\mathsf{tr}_0\sigma_0\rho$ is a valid first-order trace.
5. If so, let (tr_S, ϕ_S) be a lifting of such a valid first-order trace to a second-order trace.
6. Check whether tr_S passes in Q. If not, return tr_S.
7. Let ψ_S be the frame such that $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}(Q)$.
8. Check whether $\phi_S \sqsubseteq_s \psi_S$. If not, return tr_S.

Note that the underlying execution corresponding to tr_S is not necessarily uniquely defined since P is not necessarily action-deterministic, but we refer here to the one that follows the symbolic execution tr_0.

4.3 Termination, soundness, and completeness

Deducibility and static inclusion are well known to be decidable for standard primitives. These two decidability results can easily be adapted in our setting. It is therefore easy to establish termination.

Proposition 1 (termination)**.** *Let P and Q be two protocols such that Q is action-deterministic. The algorithm $\mathcal{A}$ applied on P and Q terminates.*

A trace returned by our algorithm is indeed a witness of non-inclusion.

Proposition 2 (soundness)**.** *Let P and Q be two protocols such that Q is action-deterministic. If the algorithm $\mathcal{A}$ applied on P and Q returns a witness tr of non-inclusion, then we have that $P \not\sqsubseteq Q$.*

Establishing completeness is more involved and the full proof is provided in Appendix A. The main difficulty is to ensure that unification performed at step 2 of the algorithm is sufficient to produce all possible relevant equalities. In particular, regarding static inclusiopn, we have to ensure that this is sufficient to consider tests R, R' that reduce to some encrypted subterms. The fact that we consider only unification between encrypted subterms is a key element for proving that our algorithm indeed returns a well-typed witness when P is not trace included in Q (cf. Section 4.4).

Proposition 3 (completeness)**.** *Let P and Q be two protocols such that Q is action-deterministic, and $P \not\sqsubseteq Q$. The algorithm $\mathcal{A}_{\mathcal{B}}$ applied on P and Q returns a minimal (in term of number of actions) witness tr of non-inclusion.*

4.4 Type-preservation

As the algorithm we presented only relies on unification between encrypted subterms of protocol P, and as unifiable encrypted subterms of protocol P have the same type by type compliance we get the following result.

Theorem 2. *Let P be a protocol type-compliant w.r.t. $(\mathcal{T}_P, \delta_P)$, and Q be another protocol that is action-deterministic such that $P \not\sqsubseteq Q$. Assume the algorithm $\mathcal{A}$ uses a well-typed renaming ρ at step 3. Then $\mathcal{A}(P, Q)$ returns a witness tr of this non-inclusion such that $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ for some ϕ and (tr, ϕ) is well-typed w.r.t. $(\mathcal{T}_P, \delta_P)$.*

5 An alternative definition of static inclusion

In this section, we propose an alternative definition of static inclusion and we state that this definition coincides with the original one. The proof of this result can be found in Appendix B. Note that, as opposed to [10], Theorem 2 is independent of this result.

We define precompact recipe. They are destructor-only recipes that do not produce a pair as a result.

Definition 10 (precompact)**.** *Given a frame ϕ, a recipe R is said to be* ϕ-precompact *if R is destructor-only, and $R\phi\downarrow$ is a term in $\mathcal{T}_0(\Sigma_{\mathsf{c}}, \Sigma_0 \cup \mathcal{N})$ but not a pair.*

It allows us to define our new notion of static inclusion. This notion is useful in practice, as we have to explore less recipes to establish static inclusion between frames.

Definition 11. *Two frames ϕ_1 and ϕ_2 are in* static inclusion *w.r.t. precompact recipes, written $\phi_1 \sqsubseteq'_s \phi_2$, when we have that $dom(\phi_1) = dom(\phi_2)$, and:*

1. *for any ϕ_1-precompact recipe R, we have that $R\phi_1\downarrow$ is an atom implies that $R\phi_2\downarrow$ is an atom; and*

2. *for any ϕ_1-precompact recipe R, we have that $R\phi_2\downarrow \in \mathcal{T}_0(\Sigma_c, \Sigma_0 \cup \mathcal{N})$; and*

3. *for any ϕ_1-precompact recipes R_1 and R_2, we have that $R_1\phi_1\downarrow = R_2\phi_1\downarrow$ implies $R_1\phi_2\downarrow = R_2\phi_2\downarrow$.*

We can show that the two notions coincide.

Proposition 4. *Let ϕ_1 and ϕ_2 be two frames. We have that :*

$$\phi_1 \sqsubseteq_s \phi_2 \text{ if, and only if, } \phi_1 \sqsubseteq'_s \phi_2.$$

6 Conclusion

Our typing result for trace inclusion relies on the design of a new procedure for trace inclusion, that preserves typing. Actually this procedure is similar to the one provided in [10]. Specifically, we show that it is sufficient to consider only unification between encrypted (sub)terms. As future work, we plan to extend this result to additional primitives (asymmetric encryption, signature, hashes). We hope that the proof we have done here to establish the result for symmetric encryption and concatenation would be easier to extend than the original proof provided in [10].

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th Symposium on Principles of Programming Languages (POPL'01)*. ACM Press, 2001.

[2] M. Arapinis and M. Duflot. Bounding messages for free in security protocols - extension to various security properties. *Inf. Comput.*, 239:182–215, 2014.

[3] M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 195–209. IEEE Computer Society, 2008.

[4] D. Baelde, S. Delaune, and L. Hirschi. Partial order reduction for security protocols. In *Proc. 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *LIPIcs*, pages 497–510. Leibniz-Zentrum für Informatik, 2015.

[5] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *12th ACM Conference on Computer and Communications Security (CCS'05)*. ACM Press, 2005.

[6] M. Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *23rd Computer Security Foundations Symposium (CSF'10)*, 2010.

[7] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 2016. To appear.

[8] V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM.

[9] V. Cheval, V. Cortier, and S. Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, June 2013.

[10] R. Chrétien, V. Cortier, and S. Delaune. Typing messages for free in security protocols: the case of equivalence properties. In P. Baldan and D. Gorla, editors, *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *Lecture Notes in Computer Science*, pages 372–386, Rome, Italy, Sept. 2014. Springer.

[11] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0, 1997.

[12] H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. Application to key cycles. *ACM Transactions on Computational Logic (TOCL)*, 11(4), 2010.

[13] V. Cortier, A. Dallon, and S. Delaune. Sat-equiv: an efficient tool for equivalence properties. In *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, Santa Barbara, CA, USA, Aug. 2017. IEEE Computer Society Press.

[14] V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, Feb. 2009.

[15] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.

[16] A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321, 2010.

[17] A. Tiu, R. Goré, and J. E. Dawson. A proof theoretic analysis of intruder theories. *Logical Methods in Computer Science*, 6(3), 2010.

A Proof of completeness

We define $\Downarrow$ as the normal form associated to the following rewriting rules:

$$\mathsf{proj}_i(\langle x_1, x_2\rangle) \to x_i, \text{ and } \mathsf{sdec}(\mathsf{senc}(x, y), z) \to x.$$

A recipe in normal form w.r.t. $\Downarrow$ is called *without detour*.

Our algorithm. It consists of several steps starting at level 1 until ℓ (where ℓ denotes the maximal length of a trace in $\mathsf{trace}_s(P)$). If nothing is returned when the iteration steps for level ℓ has been done, then it returns *yes*, *i.e.* P is trace included in Q.

Iteration steps for level n:

1. Consider every symbolic trace $\mathsf{tr}_0 \in \mathsf{trace}_s(P)$ of length n.
2. Let σ_0 be the most general unifier of some pairs of encrypted subterms occurring in tr_0.
3. Let ρ be a bijective renaming mapping variables occurring in $\mathsf{tr}_0\sigma_0$ to fresh constants from Σ_{fresh}. The choice of the constant does not matter but we have to consider the case where a variable is mapped to an atomic constant or a non-atomic constant.
4. Check whether $\mathsf{tr}_0\sigma_0\rho$ is a valid first-order trace.
5. If so, let (tr_S, ϕ_S) be a lifting of such a valid first-order trace to a second-order trace.
6. Check whether tr_S passes in Q. If not, return tr_S.
7. Let ψ_S be the frame such that $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}(Q)$.
8. Check whether $\phi_S \sqsubseteq_s \psi_S$. If not, return tr_S.

Lemma 3. *Let ϕ_S and θ be as in Definition 9. Then λ is well-defined, and if $(c\theta)\phi_S\downarrow$ is a message for each $c \in dom(\theta)$ and an atomic message for each atomic c, then $c\lambda$ is a message for each $c \in dom(\lambda)$ and it is an atomic message for each atomic c.*

Proof. We will prove by induction on the length of ϕ_S that:

- λ is well-defined.
- If $(c\theta)\phi_S\downarrow$ is a message for each $c \in dom(\theta)$, and if moreover $(c\theta)\phi_S\downarrow$ is an atom for each atomic $c \in dom(\theta)$, then:
  - $c\lambda$ is a message for each $c \in dom(\lambda)$.
  - $(c\theta)(\phi_S\lambda)\downarrow = (c\theta)\phi_S\downarrow\lambda$.
  - $c\lambda$ is an atom for each atomic c.

Base case. We prove the result for an empty ϕ_S. The only $c \in dom(\theta)$ are those such that $c\theta$ has no variable. As $\phi_S\lambda$ is empty, $c\lambda = (c\theta)(\phi_S\lambda)\downarrow = c\theta\downarrow$ is well-defined.

If $c\theta\downarrow = (c\theta)\phi_S\downarrow$ is a message, then $c\lambda = c\theta\downarrow$ is a message and $(c\theta)(\phi_S\lambda)\downarrow = c\theta\downarrow = (c\theta)\phi_S\downarrow\lambda$.

If moreover $c\theta\downarrow = (c\theta)\phi_S\downarrow$ is an atom when c is atomic, then $c\lambda = c\theta\downarrow$ is an atom.

Inductive case. Assume that we have the result for any symbolic frame of length n, and let ϕ_S be a frame of length $n+1$. We call ϕ_S^n the frame ϕ_S cut at its n^{th} value. Then consider a $c \in dom(\theta)$. If $vars(c\theta) \subseteq dom(\phi_S^n)$, then we get that $c\lambda$ is well-defined by induction hypothesis. Else, $c\theta$ contains the last variable of ϕ_S, so c does not occur in ϕ_S. So $c\lambda = (c\theta)(\phi_S\lambda)\downarrow$ is well-defined as λ only instantiates the fresh constants at previous steps.

Moreover, if $(c\theta)\phi_S\downarrow$ is a message for each $c \in dom(\theta)$, then for c such that $c\theta$ is defined in ϕ_S^n, we have by induction hypothesis that $c\lambda = (c\theta)(\phi_S\lambda)\downarrow = (c\theta)\phi_S\downarrow\lambda$ and that it is a message, and it is atomic if c is atomic. For c such that $c\theta$ was not defined in ϕ_S^n, $c\lambda = (c\theta)(\phi_S\lambda)\downarrow$ and in $\phi_S\lambda$, λ only instantiates the fresh constants that occur in ϕ_S, that is those that were defined in ϕ_S^n. So on those constants c', $c'\lambda$ is a message and it is atomic if c' is atomic. So $(c\theta)(\phi_S\lambda)\downarrow = (c\theta)\phi_S\downarrow\lambda$ and $c\lambda$ is a message if $(c\theta)\phi_S\downarrow$ is a message, it is atomic if c is atomic (as $(c\theta)\phi_S\downarrow$ is atomic and λ replaces atoms by atoms by induction hypothesis).

It concludes the proof. □

Lemma 4. *Let ϕ_S and θ as in Definition 9. Let λ the first-order substitution associated to θ through ϕ_S. Assume that for any $c \in dom(\lambda)$, $c\lambda$ is an atom whenever c is atomic. Then for any recipe R such that $R\phi_S\downarrow$ is a symbolic message, we have that $(R\theta)(\phi_S\lambda)\downarrow = (R\phi_S\downarrow)\lambda$.*

Proof. Recall that $c\lambda = (c\theta)(\phi_S\lambda)\downarrow$ for any $c \in dom(\lambda) = dom(\theta)$.

We prove the result by structural induction on R. If R is a $\mathsf{w} \in dom(\phi_S)$, then $\mathsf{w}\theta = \mathsf{w}$ so (as $c\lambda$ is in normal form by the equation we recalled) $(\mathsf{w}\theta)(\phi_S\lambda)\downarrow = \mathsf{w}\phi_S\lambda\downarrow = \mathsf{w}\phi_S\lambda = \mathsf{w}\phi_S\downarrow\lambda$ which is the result.

If R is a constant $c \notin dom(\lambda) = dom(\theta)$, then both sides of the equation equal c.

If R is a constant $c \in dom(\lambda)$ then $c\lambda = (c\theta)(\phi_S\lambda)\downarrow$, so $R\phi_S\downarrow\lambda = (R\theta)(\phi_S\lambda)\downarrow$.

If $R = \mathsf{f}(R_1, R_2)$ with f a constructor symbol, then by induction hypothesis

$$\begin{aligned}(R\theta)(\phi_S\lambda)\downarrow &= \mathsf{f}((R_1\theta)(\phi_S\lambda)\downarrow, (R_2\theta)(\phi_S\lambda)\downarrow)\\ &= \mathsf{f}(R_1\phi_S\downarrow\lambda, R_2\phi_S\downarrow\lambda)\\ &= R\phi_S\downarrow\lambda\end{aligned}$$

If $R = \mathsf{proj}_i(R')$ and $R\phi_S\downarrow$ is a symbolic message (in particular $R'\phi_S\downarrow = \langle u_S^1, u_S^2\rangle$ is a symbolic message) then by induction hypothesis

$$\begin{aligned}(R\theta)(\phi_S\lambda)\downarrow &= \mathsf{proj}_i((R'\theta)(\phi_S\lambda)\downarrow)\downarrow\\ &= \mathsf{proj}_i((R'\phi_S)\downarrow\lambda)\downarrow\\ &= \mathsf{proj}_i(\langle u_S^1, u_S^2\rangle\lambda)\downarrow\\ &= u_s^1\lambda\downarrow = R\phi_S\downarrow\lambda\downarrow = R\phi_S\downarrow\lambda\end{aligned}$$

(the last step beeing valid because λ only replaces atoms by atoms by hypothesis).

If $R = \mathsf{sdec}(R_1, R_2)$ and $R\phi_S\downarrow$ is a symbolic message (in particular $R_1\phi_S\downarrow = \mathsf{senc}(u_S^1, u_S^2)$ and $R_2\phi_S\downarrow = u_S^2$) then

$$\begin{aligned}(R\theta)(\phi_S\lambda)\downarrow &= \mathsf{sdec}((R_1\theta)(\phi_S\lambda)\downarrow, (R_2\theta)(\phi_S\lambda)\downarrow)\downarrow\\ &= \mathsf{sdec}((R_1\phi_S\downarrow)\lambda, (R_2\phi_S)\downarrow\lambda)\downarrow\\ &= \mathsf{sdec}(\mathsf{senc}(u_S^1, u_S^2)\lambda, u_S^2\lambda)\downarrow = u_S^1\lambda\downarrow = R\phi_S\downarrow\lambda\downarrow = R\phi_S\downarrow\lambda\end{aligned}$$

(the last step beeing valid because λ only replaces atoms by atoms by hypothesis and $R\phi_S\downarrow$ is a message).

It concludes the induction on R and proves that $(R\theta)(\phi_S\lambda)\downarrow = (R\phi_S\downarrow)\lambda$. □

Lemma 2. *Let $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. There exists $\mathsf{tr}_0 \in \mathsf{trace}_s(P)$, $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}(P)$, a substitution σ_0 which is the mgu of some pairs of encrypted subterms occurring in tr_0, a bijective mapping ρ from variables in $\mathsf{tr}_0\sigma_0$ to fresh constants, and two substitutions θ and λ such that:*

1. $(\mathsf{tr}_0\sigma_0)\rho = \mathsf{tr}_S\phi_S\downarrow$;
2. $(\mathsf{tr}_0\sigma_0)\rho\lambda = \mathsf{tr}\phi\downarrow$;
3. *for any $x \in vars(\mathsf{tr}_0\sigma_0)$, we have that $x\rho$ is an atomic constant if, and only if $x\rho\lambda$ is atomic;*

4. *$dom(\theta) = dom(\lambda)$ and $c\theta = R_c$ for some $R_c \in \mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \{\mathsf{w}_1, \ldots, \mathsf{w}_i\})$ such that $R_c\phi{\downarrow} = c\lambda$ and i is the number of outputs that occur in $(\mathsf{tr}_s\sigma_0)\rho$ before the first occurrence of an input that contains the fresh constant c.*

5. *$(\mathsf{tr}_S\theta)(\phi_S\lambda){\downarrow} = \mathsf{tr}\phi{\downarrow}$;*

6. *for any encrypted subterms t_1, t_2 occurring in $\mathsf{tr}_0\sigma_0$, we have that $t_1\rho\lambda = t_2\rho\lambda$ implies that $t_1 = t_2$.*

Moreover, λ is the first-order substitution associated to θ through ϕ_S.

Proof. Let $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$. From [14], we have that there exists $\mathsf{tr}_0 \in \mathsf{trace}_s(P)$, $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}(P)$, a substitution σ_0 which is the mgu of some pairs of encrypted subterms occurring in tr_0, a bijective mapping ρ from variables in $\mathsf{tr}_0\sigma_0$ to fresh constants, and two substitutions θ and λ such that:

- $(\mathsf{tr}_0\sigma_0)\rho = \mathsf{tr}_S\phi_S{\downarrow}$;
- $(\mathsf{tr}_0\sigma_0)\rho\lambda = \mathsf{tr}\phi{\downarrow}$;
- for any $x \in vars(\mathsf{tr}_0\sigma_0)$, we have that $x\rho$ is an atomic constant if, and only if $x\rho\lambda$ is atomic;
- $dom(\theta) = dom(\lambda)$ and $c\theta = R_c$ for some $R_c \in \mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \{\mathsf{w}_1, \ldots, \mathsf{w}_i\})$ such that $R_c\phi{\downarrow} = c\lambda$ and i is the number of outputs that occur in $(\mathsf{tr}_s\sigma_0)\rho$ before the first occurrence of an input that contains the fresh constant c.
- $(\mathsf{tr}_S\theta)(\phi_S\lambda){\downarrow} = \mathsf{tr}\phi{\downarrow}$;

Moreover, λ is the first-order substitution associated to θ through ϕ_S. However, this result holds in a slightly different setting. In particular, they consider a more general framework that allows non-atomic terms in key positions. Actually, it is easy to see that, in case $\mathsf{tr}\phi{\downarrow}$ only contains atomic terms in key position, then $\mathsf{tr}_S\phi_S{\downarrow}$ also satisfies this requirement (thanks to items $1-3$). The only difficulty comes from the fact that it could be the case that $(\mathsf{tr}_S\theta)(\phi_S\lambda){\downarrow}$ only contains messages with atomic keys when rewriting is allowed also in presence of terms with non-atomic keys, and that the rewriting steps can not been done anymore when non-atomic keys are forbidden. Actually, by choosing recipes without detour this can be avoided.

Then note that in [14], we could have unified all terms t_1, t_2 in tr_0 whose instantiation is equal in $\mathsf{tr}\phi{\downarrow}$ before doing anything else. Then we get that for any encrypted subterms t_1, t_2 occurring in tr_0, we have that $t_1\sigma_0\rho\lambda = t_2\sigma_0\rho\lambda$ implies that $t_1\sigma_0 = t_2\sigma_0$. But as σ_0 is a mgu of equalities between subterms of tr_0, we have that $ESt(img(\sigma_0)) \subseteq ESt(\mathsf{tr}_0)\sigma_0$ (a proof of this result is available in [2] - Lemma 3). Therefore, we have that

$$\begin{array}{rcl} ESt(\mathsf{tr}_0\sigma_0) & \subseteq & ESt(\mathsf{tr}_0)\sigma_0 \cup ESt(img(\sigma_0)) \\ & \subseteq & ESt(\mathsf{tr}_0)\sigma_0 \end{array}$$

which concludes the proof. □

Lemma 5. *Let $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}(Q)$. Let θ be a mapping such that for any $(c \mapsto R_c) \in \theta$, we have that c is a fresh constant ($c \in \Sigma_{\mathsf{fresh}}$), and R_c is a recipe built from the ouputs which preceded the introduction of c in tr_S. Moreover, we assume that $R_c\psi_S{\downarrow}$ is a message for any $c \in dom(\theta)$ and when c is atomic then $R_c\psi_S{\downarrow}$ is atomic too. We have that $(\mathsf{tr}_S\theta, \psi_S\lambda) \in \mathsf{trace}(Q)$ where λ is the first-order substitution associated to θ through ψ_S.*

Proof. The proof is by induction on the length of the execution. Since $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}(Q)$, we have that:

$$(Q; \emptyset; \emptyset) = (Q_S^0; \psi_S^0, \sigma_S^0) \xrightarrow{\alpha_S^1} \ldots \xrightarrow{\alpha_S^n} (Q_S^n; \psi_S^n; \sigma_S^n)$$

with $\psi_S^n = \psi_S$. We are going to show that

$$(Q;\emptyset;\emptyset) = (Q^0;\psi^0,\sigma^0) \xrightarrow{\alpha_S^1\theta} \dots \xrightarrow{\alpha_S^n\theta} (Q^n;\psi^n;\sigma^n)$$

where $Q^i = Q_S^i$, $\psi^i = \psi_S^i\lambda$, and $\sigma^i = \sigma_S^i\lambda$.

The base case is obvious. Assume we have the result for length n. We have $(Q_S^n;\psi_S^n;\sigma_S^n)$ and $(Q^n;\psi^n;\sigma^n)$ such that $Q^n = Q_S^n$, $\psi^n = \psi_S^n\lambda$, $\sigma^n = \sigma_S^n\lambda$.

Then α_S^{n+1} is either an input or an output. If it is an input, then $\alpha_S^{n+1} = \mathsf{in}(c,R)$ for some symbolic recipe R. So there exists a $\mathsf{in}(c,u).Q_c \in Q_S^n = Q^n$ such that $u\sigma_S^n$ and $R\psi_S^n\!\downarrow$ (ground term) are unifiable with *mgu* τ. We have $\sigma_S^{n+1} = \sigma_S^n \uplus \tau$.

But applying Lemma 4 (possible thanks to Lemma 3), we have $(R\theta)\psi^n\!\downarrow = (R\psi_S^n\!\downarrow)\lambda = u(\sigma_S^n \uplus \tau)\lambda = (u\sigma_S^{n+1}\lambda)(\tau\lambda)$.

So:

$$(Q^n;\psi^n;\sigma^n) \xrightarrow{\mathsf{in}(c,R\theta)} (Q^{n+1};\psi_S^{n+1}\lambda;\sigma_S^{n+1}\lambda)$$

where $Q^n = Q_S^n$, $Q^{n+1} = Q_S^{n+1}$, $\psi^{n+1} = \psi^n$ and $\psi_S^{n+1} = \psi_S^n$.

Now we can assume that α_S^{n+1} is an output. $\alpha_S^n = \mathsf{out}(c,\mathsf{w})$. So there exists a $\mathsf{out}(c,u).Q_c \in Q_S^n = Q^n$ and $\psi_S^{n+1} = \psi_S^n \uplus \{\mathsf{w} \triangleright u\sigma_S^n\}$.

We have that $\alpha_S^{n+1}\theta = \alpha_S^{n+1}$ and $\psi^{n+1} = \psi^n \uplus \{\mathsf{w} \triangleright u\sigma^n\} = \psi_S^{n+1}\lambda$ by induction hypothesis.

So we deduce that $Q^{n+1} = Q_S^{n+1}$, $\sigma^{n+1} = \sigma^n = \sigma_S^n\lambda = \sigma_S^{n+1}\lambda$. We get that $\sigma^{n+1} = \sigma_S^{n+1}\lambda$ and $\psi^{n+1} = \psi_S^{n+1}\lambda$.

So it allows to conclude the induction on the execution of tr_S.

Then, from this result, we easily derive the result stated in the lemma. □

Before to start the proof of our completeness result, we introduce a measure on recipes.

Our measure on recipes. Before defining our measure, we need to define a multiset associated to a symbolic frame ϕ_S and a recipe R. This multiset is defined inductively as follows:

$\mathsf{Multi}_{\phi_S}(\mathsf{w}) = \{\mathsf{w}\}$

$\mathsf{Multi}_{\phi_S}(c) = \{c\}$ for any $c \in \Sigma_0 \uplus \Sigma_{\mathsf{fresh}}$.

$\mathsf{Multi}_{\phi_S}(\mathsf{sdec}(R_1,c)) = \mathsf{Multi}_{\phi_S}(R_1)$ when $R_1\phi_S\!\downarrow = \mathsf{senc}(\_,c)$ for $c \in \Sigma_{\mathsf{fresh}}$.

$\mathsf{Multi}_{\phi_S}(\mathsf{sdec}(R_1,R_2)) = \mathsf{Multi}_{\phi_S}(R_1) \uplus \mathsf{Multi}_{\phi_S}(R_2)$ otherwise.

$\mathsf{Multi}_{\phi_S}(\mathsf{proj}_i(R')) = \mathsf{Multi}_{\phi_S}(R')$

$\mathsf{Multi}_{\phi_S}(\mathsf{f}(R_1,R_2)) = \mathsf{Multi}_{\phi_S}(R_1) \uplus \mathsf{Multi}_{\phi_S}(R_2)$ where f is any constructor.

Given ϕ_S a symbolic frame and ϕ its concretization through θ, i.e. $\phi = \phi_S\lambda_P$), and a test T (i.e. either $R \in \mathcal{T}(\Sigma_{\mathsf{std}}, \mathcal{W} \cup \Sigma_0 \cup \Sigma_{\mathsf{fresh}})$ or $R_1 = R_2$ with $R_1, R_2 \in \mathcal{T}(\Sigma_{\mathsf{std}}, \mathcal{W} \cup \Sigma_0 \cup \Sigma_{\mathsf{fresh}})$) such that $T\theta$ holds in ϕ, we consider the measure $\mu(T)$ defined as follows (lexicographic order):

- the multiset $\mathsf{Multi}_{\phi_S}(R)$ (resp. $\mathsf{Multi}_{\phi_S}(R_1) \uplus \mathsf{Multi}_{\phi_S}(R_2)$) where constants from Σ_0 are minimal and elements from $\mathcal{W} \cup \Sigma_{\mathsf{fresh}}$ are ordered following the order in which they appear in the frame ϕ_S.
- the number of constructor symbols from Σ_{c} occurring in R (resp. R_1 and R_2);
- $|R\phi\!\downarrow|$ (resp. $|R_1\phi\!\downarrow| + |R_2\phi\!\downarrow|$) where $|u|$ is the number of symbols occurring in u.

Proposition 5. *Let P and Q be two protocols such that Q is action-deterministic and $P \not\sqsubseteq Q$. The algorithm applied on P and Q returns a minimal (in term of number of actions) witness* tr *of non-inclusion.*

Proof. Let $(\mathsf{tr}, \phi) \in \mathsf{trace}(P)$ be a minimal (in length) witness of non-inclusion. Let n be the length of this witness. We have that $P \sqsubseteq_{n-1} Q$.

First, we apply Lemma 2. There exists $\mathsf{tr}_0 \in \mathsf{trace}_s(P)$, $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}(P)$, σ_0 which is the mgu of some pairs of encrypted subterms occurring in tr_0, a bijective mapping ρ from variables in $\mathsf{tr}_0\sigma_0$ to fresh constants, and two subsitutions θ and λ_P such that:

- $(\mathsf{tr}_0\sigma_0)\rho = \mathsf{tr}_S\phi_S{\downarrow}$;
- $(\mathsf{tr}_0\sigma_0)\rho\lambda_P = \mathsf{tr}\phi{\downarrow}$;
- for any $x \in vars(\mathsf{tr}_0\sigma_0)$, we have that $x\rho$ is an atomic constant if, and only if $x\rho\lambda_P$ is atomic;
- $dom(\theta) = dom(\lambda_P)$ and $c\theta = R_c$ for some $R_c \in \mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \{\mathsf{w}_1, \ldots, \mathsf{w}_i\})$ such that $R_c\phi{\downarrow} = c\lambda_P$ and i is the number of outputs that occur in $(\mathsf{tr}_s\sigma_0)\rho$ before the first occurrence of an input that contains the fresh constant c.
- $(\mathsf{tr}_S\theta)(\phi_S\lambda_P){\downarrow} = \mathsf{tr}\phi{\downarrow}$.
- for any encrypted subterms t_1, t_2 occurring in tr_0, we have that $(t_1\sigma_0)\rho\lambda_P = (t_2\sigma_0)\rho\lambda_P$ implies that $t_1\sigma_0 = t_2\sigma_0$.

Therefore, following our algorithm, we choose σ_0 at step 2, and ρ at step 3 accordingly. At step 4, we know that $(\mathsf{tr}_0\sigma_0)\rho$ is a valid first-order trace.

At step 5, we choose $(\mathsf{tr}'_S, \phi'_S)$ a lifting of the valid first-order trace $(\mathsf{tr}_0\sigma_0)\rho$. Since this lifting is not unique, we do not necessarily have that $\mathsf{tr}_S = \mathsf{tr}'_S$ but we have that $\phi'_S = \phi_S$, and $(\mathsf{tr}_S = \mathsf{tr}'_S)\phi_S{\downarrow}$.

If tr'_S does not pass in Q, then it is at its last instruction, as $P \sqsubseteq_{n-1} Q$ and tr'_S is a trace of P. But at step 6 of the algorithm, tr'_S is outputed if it does not pass in Q. So we can assume that tr'_S passes in Q. Let ψ'_S be the resulting frame. Thanks to the action-determinism of Q, we have that $\phi_S = \phi'_S \sqsubseteq_s \psi'_S$.

We have that $\mathsf{tr}'_S\phi_S{\downarrow} = \mathsf{tr}_S\phi_S{\downarrow}$. Since $\phi_S \sqsubseteq_s \psi'_S$, we have also that $\mathsf{tr}'_S\psi'_S{\downarrow} = \mathsf{tr}_S\psi'_S{\downarrow}$, and therefore tr_S passes in Q, and the resulting frame ψ_S is equal to ψ'_S.

We apply Lemma 5 on (tr_S, ψ_S) and θ. First, since $R_c\phi_S{\downarrow}$ is a message and $\phi_S \sqsubseteq \psi'_S$, we know that $R_c\psi_S{\downarrow}$ is a message too. Second, when $R_c\phi_S{\downarrow}$ is atomic, we know that $R_c\psi_S{\downarrow}$ is atomic too. We obtain that $(\mathsf{tr}_S\theta, \psi_S\lambda_Q) \in \mathsf{trace}(Q)$ where λ_Q is the first-order substitution associated to θ through ψ_S.

We have that $(\mathsf{tr}_S\theta = \mathsf{tr})\phi{\downarrow}$ and $\mathsf{tr}_S\theta$ leads to the frame $\psi_S\lambda_Q$ in Q. Since $P \sqsubseteq_{n-1} Q$, we have that $(\mathsf{tr}_S\theta = \mathsf{tr})(\psi_S\lambda_Q)$, and therefore tr passes in Q and the resulting frame is $\psi = \psi_S\lambda_Q$.

So, we still have that tr is a witness of non-inclusion of length n but by contradiction we assume that our algorithm returns no witness at iteration n, i.e. $\phi_S \sqsubseteq_s \psi_S$. We have shown that tr passes in Q and leads to the frame ψ. So, since tr is a witness, it means that $\phi \not\sqsubseteq_s \psi$.

We have that $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}(P)$, $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}(Q)$, $\phi_S\lambda_P = \phi$, and $\psi_S\lambda_Q = \psi$.

Moreover, for each recipe $c \in dom(\theta) = dom(\lambda_P)$, $c\theta$ is a recipe where only the variables which preceded the introduction of c in ϕ_S occur. $c\lambda_P$ is atomic when c is atomic.

Therefore Lemma 4 applies and we have that for each recipe R such that $R\phi_S{\downarrow}$ is a symbolic message, $(R\theta)\phi{\downarrow} = (R\theta)(\phi_S\lambda_P){\downarrow} = (R\phi_S{\downarrow})\lambda_P$.

By static equivalence at the previous step (that is from $P \sqsubseteq_{n-1} Q$) we get that $c\lambda_Q = (c\theta)\psi{\downarrow}$ is an atom whenever $c\lambda_P = (c\theta)\phi{\downarrow}$ is an atom, and in particular when c is atomic.

So Lemma 4 applies and we have that for each recipe R such that $R\psi_S{\downarrow}$ is a symbolic message, $(R\theta)\psi{\downarrow} = (R\theta)(\psi_S\lambda_Q){\downarrow} = (R\psi_S{\downarrow})\lambda_Q$.

We have that $\phi_S \sqsubseteq_s \psi_S$. We have to show that $\phi \sqsubseteq_s \psi$. To establish this result, we consider a test T (built on $\mathcal{T}(\Sigma_{\mathsf{std}}, \Sigma_0 \cup \mathcal{W} \cup \Sigma_{\mathsf{fresh}})$) such that $T\theta$ holds in ϕ. Moreover, we assume that for all T' such that $\mu(T') < \mu(T)$, we have that:

$$T'\theta \text{ holds in } \phi \text{ implies that } T'\theta \text{ holds in } \psi.$$

Now, we prove that $T\theta$ holds in ψ.

A test that holds in ϕ can have the following form:

1. either a recipe R such that $R\phi\downarrow$ is a message (resp. atomic message), and we have to establish that $R\psi\downarrow$ is a message (resp. atomic message).

2. either two recipes R_1 and R_2 such that $R_1\phi\downarrow$ and $R_2\phi\downarrow$ are both messages and $R_1\phi\downarrow = R_2\phi\downarrow$, and we have to show that $R_1\psi\downarrow$, $R_2\psi\downarrow$ are both messages, and $R_1\psi\downarrow = R_2\psi\downarrow$.

Case T is a recipe R such that $(R\theta)\phi\downarrow$ is a message (resp. atomic message).

First, we assume that R is not in normal form w.r.t. $\Downarrow$. In such a case, we have that $R = R_0[\mathsf{sdec}(\mathsf{senc}(R_1, R_2), R_3)]$ (the case where $R = R_0[\mathsf{proj}_i(\langle R_1, R_2\rangle)]$ can be done in a similar way), and since $(R\theta)\phi\downarrow$ is a message, we know that $(R_2\theta)\phi\downarrow$ and $(R_3\theta)\phi\downarrow$ are both atomic messages such that $(R_2\theta)\phi\downarrow = (R_3\theta)\phi\downarrow$. Note that either $\mu(R_2 = R_3) < \mu(R)$, and thus we have that $(R_2\theta)\psi\downarrow$ and $(R_3\theta)\psi\downarrow$ are both messages, and $(R_2\theta)\psi\downarrow = (R_3\theta)\psi\downarrow$. Otherwise, we have that $R_3 = c$ and $R_2\phi_S\downarrow = c$, and thus $(R_2 = R_3)$ holds in ϕ_S, and therefore $(R_2 = R_3)$ holds in ψ_S thanks to our algorithm. This allows us to conlude that $(R_2\theta = R_3\theta)$ holds in ϕ and also in ψ by Lemma 4.

Let $R' = R_0[R_1]$. We have that $(R\theta)\phi\downarrow = (R'\theta)\phi\downarrow$, and $(R\theta)\psi\downarrow = (R'\theta)\psi\downarrow$. Moreover, we have that $\mu(R') < \mu(R)$. Since $(R\theta)\phi\downarrow$ is a message, then we have that $(R'\theta)\phi\downarrow$ is a message too, and by our induction hypothesis, we know that $(R'\theta)\psi\downarrow$ is a message. This allows us to conclude that $(R\theta)\psi\downarrow$ is a message.

Therefore, we know that R is in normal form w.r.t. $\Downarrow$. Moreover, by hypothesis, we know that $(R\theta)\phi\downarrow$ is a message. By structural induction on R, we establish that R is made of constructors on top of destructors (If there is a destructor on top of a constructor in R, then either the constructor is in key position, or there is a destructor on an incompatible destructor. In both cases, $(R\theta)\phi\downarrow$ cannot be a message.).

From now on, we assume that R is made of constructors on top of destructors. In case R is of the form $R = \mathsf{senc}(R_1, R_2)$ (the case where $R = \langle R_1, R_2\rangle$ can be done in a similar way), then we conclude by applying our induction hypothesis on R_1 and R_2. Note that $(R_2\theta)\phi\downarrow$ is atomic, and therefore by IH $(R_2\theta)\psi\downarrow$ is atomic too.

Therefore, we know that R is destructor-only. Assume that $R\phi_S\downarrow$ is not a message. We take the smallest subterm R' of R such that $R'\phi_S\downarrow$ is not a message. Either $R' = \mathsf{proj}_i(R'')$ or $R' = \mathsf{sdec}(R_1, R_2)$ as R is a destructor-only recipe and atomic recipes give messages. Assume $R' = \mathsf{sdec}(R_1, R_2)$ (the case where $R' = \mathsf{proj}_i(R'')$ can be done in a similar way).

- If $R_1\phi_S\downarrow$ is not an encryption, then $R_1\phi_S\downarrow = c$ for some fresh constant c. By our algorithm, we have that $R_1\psi_S\downarrow = c$. Therefore by Lemma 4 we have that $(R_1\theta)\phi\downarrow = (c\theta)\phi$ and $(R_1\theta)\psi\downarrow = (c\theta)\psi\downarrow$. We consider $R_0 = R[R_1 \to c\theta]$. $\mu(R_0) < \mu(R)$ because $R_1\psi_S\downarrow = c$ and c occurs after all variables of $c\theta$ by definition of θ, so there is at least one leaf in R_1 that is bigger than any variable of $c\theta$. $(R_0\theta)\phi\downarrow$ is a message. By IH, $(R_0\theta)\psi\downarrow$ is a message, and since $(R_1\theta = c\theta)\psi\downarrow$, we have that $(R\theta)\psi\downarrow$ is a message too.

- If $R_1\phi_S\downarrow = \mathsf{senc}(u_1, u_2)$ and $R_2\phi_S\downarrow = v$. Note that u_2 and v are atoms, i.e. names, constants from Σ_0, or constants from Σ_{fresh}. Since, we know that the reduction was not possible at the symbolic level, it means that either $v \in \Sigma_{\mathsf{fresh}}$, or $u_2 \in \Sigma_{\mathsf{fresh}}$ (or both), and $u_2 \neq v$.

  In case $v = c \in \Sigma_{\mathsf{fresh}}$, then we consider $R_0 = R[R' \to \mathsf{sdec}(R_1, c\theta)]$, and we have that $\mu(R_0) < \mu(R)$ since $R_2\phi_S\downarrow$ leads to a fresh constant c meaning that either $R_2 = c$ or R_2 contains some w bigger than all the w occurring in $c\theta$. We know that $(R_2 = c)\phi_S$, and $(R_2\theta)\phi\downarrow = (c\theta)\phi\downarrow$ by Lemma 4. Thus, we have that $(R_0\theta)\phi\downarrow$ is a message, and therefore thanks to our IH we have that $(R_0\theta)\psi\downarrow$ is a message. Now, thanks to our algorithm, we have that $(R_2 = c)\psi_S$ and $(R_2\theta)\psi\downarrow = (c\theta)\psi\downarrow$ by Lemma 4. This allows us to conclude that $(R\theta)\psi\downarrow$ is a message.

  Now, we consider the case where $u_2 = c \in \Sigma_{\mathsf{fresh}}$, and we can assume that $v \notin \Sigma_{\mathsf{fresh}}$. Let $R_0 = R[\mathsf{sdec}(R_1, R_2) \to \mathsf{sdec}(R_1, c)]$. We have that $\mu(R_0) < \mu(R)$ since c does not contribute

to the measure as opposed to R_2. We know that $\mathsf{sdec}(R_1, c)\phi_S{\downarrow}$ is a message. Therefore, we have that $\mathsf{sdec}(R_1\theta, c\theta)\phi{\downarrow}$ is a message and we have that $(c\theta = R_2\theta)\phi$. We have that $\mu(c\theta = R_2) < \mu(R)$. Therefore $(c\theta = R_2\theta)\psi$, and this allows us to conclude.

Therefore, we know that $R\phi_S{\downarrow}$ is a message. Thanks to our algorithm, we know that $R\psi_S{\downarrow}$ is a message too, and thus $(R\theta)\psi{\downarrow}$ is a message too, as $(R\theta)\psi{\downarrow} = (R\psi_S{\downarrow})\lambda_Q$, and λ_Q only replace atomic fresh constants by atoms.

Note that, in case $(R\theta)\phi{\downarrow}$ is an atomic message, the proof follows that same lines, and we should be able to conclude that $(R\theta)\psi{\downarrow}$ is atomic too.

In the case where $(R\theta)\phi{\downarrow}$ is an atomic message, $(R\phi_S){\downarrow}$ is an atomic message as it is a message (see case message) and $(R\theta)\phi{\downarrow} = (R\phi_S{\downarrow})\lambda_P$ by Lemma 4. So by our algorithm, $R\psi_S{\downarrow}$ is an atomic message. By Lemma 4, $(R\theta)\psi{\downarrow} = (R\psi_S{\downarrow})\lambda_Q$. If $R\psi_S{\downarrow} \notin \Sigma_{\mathsf{fresh}}$, then $(R\theta)\psi{\downarrow} = R\psi_S{\downarrow}$ is an atomic message.

If $R\psi_S{\downarrow} = c \in \Sigma_{\mathsf{fresh}}$, then c is an atomic fresh constant by our algorithm. We have shown that λ_Q replaces atoms by atoms, so $c\lambda_Q = (c\theta)\psi{\downarrow}$ is atomic. It concludes the case of atomic messages.

Case T is a test $R = R'$ such that $(R\theta)\phi{\downarrow}$, $(R'\theta){\downarrow}$ are both messages, and $(R\theta)\phi{\downarrow} = (R'\theta)\phi{\downarrow}$.

If R or R' is not in normal form, then in the message-test case, we have shown that $(R{\Downarrow}\theta)\phi{\downarrow} = (R\theta)\phi{\downarrow}$ and that $R{\Downarrow}$ is smaller than R so by minimality it is impossible. Now we know that both recipes are constructor on top of destructor. But each head constructor has to be the same both sides, so we can remove it until there is no more constructor one side. Therefore we can assume that say R' is a destructor recipe.

In case $(R'\theta)\phi{\downarrow} = \langle u_1, u_2\rangle$. Let $R'_l = \mathsf{proj}_1(R')$ and $R'_r = \mathsf{proj}_2(R')$. Assuming that R is not destructor-only, we have that $R = \langle R_1, R_2\rangle$ where R_1, R_2 are made of constructors on top of destructors. We have that $R_1 = R'_l$ and $R_2 = R'_r$ are tests smaller than the original one such that $(R_1\theta)\phi{\downarrow} = (R'_l\theta)\phi{\downarrow}$ and $(R_2\theta)\phi{\downarrow} = (R'_r\theta)\phi{\downarrow}$. We can transfer them in ψ. We obtain that $(R_1\theta)\psi{\downarrow} = (R'_l\theta)\psi{\downarrow}$ and $(R_2\theta)\psi{\downarrow} = (R'_r\theta)\psi{\downarrow}$, and we conclude that $(R\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$.

A similar reasoning holds when $(R'\theta)\phi{\downarrow} = \mathsf{senc}(u_1, u_2)$ and assuming that R is not destructor-only. We can replace the constructor senc on top of R by a destructor in the other side of the equality test. The last component of our measure (at least) will decrease.

Therefore, we know that both R and R' are destructor only. Similar to the previous case (case message), we deduce that $R\phi_S{\downarrow}$ and $R'\phi_S{\downarrow}$ are messages.

In case $R\phi_S{\downarrow} = c \in \Sigma_{\mathsf{fresh}}$ then let $R_0 = c\theta$. We have that $(R\theta)\phi{\downarrow} = (c\theta)\phi{\downarrow} = (R_0\theta)\phi{\downarrow}$, and therefore $R_0 = R'$ is a test such that $(R_0\theta)\phi{\downarrow} = (R'\theta)\phi{\downarrow}$ that holds in ϕ and $\mu(R_0 = R') < \mu(R = R')$. Therefore, by IH, we have that $(R_0\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$. Note that $R\psi_S{\downarrow} = c$ (thanks to our algorithm), and therefore $(R\theta)\psi{\downarrow} = (c\theta)\psi{\downarrow}$. This allows us to conclude.

Now, we assume that neither $R\phi_S{\downarrow}$, nor $R'\phi_S{\downarrow}$ leads to a constant from Σ_{fresh}. Therefore either both are constants from Σ_0 or names from $\mathcal{N}$. In such a case, we have that $R\phi_S{\downarrow} = R'\phi_S{\downarrow}$ (else by Lemma 4 we would have $R\phi{\downarrow} \neq R'\phi{\downarrow}$). They lead necessarily to the same constant or name. Therefore, thanks to our algorithm, we know that $R\psi_S{\downarrow} = R'\psi_S{\downarrow}$, and thus $(R\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$ as $(R\theta)\psi{\downarrow} = (R\psi_S{\downarrow})\lambda_Q$ and $(R'\theta)\psi{\downarrow} = (R'\psi_S{\downarrow})\lambda_Q$ by Lemma 4.

Otherwise, $R\phi_S{\downarrow}$ and $R'\phi_S{\downarrow}$ both pairs $\langle u_1, u_2\rangle$ (resp. $\langle u'_1, u'_2\rangle$) or both ciphertexts $\mathsf{senc}(u_1, u_2)$ (resp. $\mathsf{senc}(u'_1, u'_2)$).

In case $R\phi_S{\downarrow} = \langle u_1, u_2\rangle$ and $R'\phi_S{\downarrow} = \langle u'_1, u'_2\rangle$, we consider $R_l = \mathsf{proj}_1(R)$, $R_r = \mathsf{proj}_2(R)$ (similarly we defined R'_l and R'_r). We have that $(R_l\theta)\phi{\downarrow} = (R'_l\theta)\phi{\downarrow}$ (resp. $(R_r\theta)\phi{\downarrow} = (R'_r\theta)\phi{\downarrow}$) Since those tests are smaller than $R = R'$, thanks to our induction hypothesis, we deduce that $(R_l\theta)\psi{\downarrow} = (R'_l\theta)\psi{\downarrow}$ and $(R_r\theta)\psi{\downarrow} = (R'_r\theta)\psi{\downarrow}$. Therefore, we conclude that $(R\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$.

Now, it remains the case where $R\phi_S{\downarrow} = \mathsf{senc}(u_1, u_2)$ and $R'\phi_S{\downarrow} = \mathsf{senc}(u'_1, u'_2)$. Since R and R' are destructor-only, we know that $\mathsf{senc}(u_1, u_2)$ and $\mathsf{senc}(u'_1, u'_2)$ are encrypted subterms from ϕ_S. Since $\mathsf{senc}(u_1, u_2)\lambda_P = \mathsf{senc}(u'_1, u'_2)\lambda_P$ (the equality holds in ϕ), we know that $\mathsf{senc}(u_1, u_2) = \mathsf{senc}(u'_1, u'_2)$ (see sixth item of Lemma 2). Therefore, we have that $R\psi_S{\downarrow} = R'\psi_S{\downarrow}$, and thus $(R\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$. □

B An alternative definition of static inclusion

The purpose of this section is to show that we can consider the alternative Definition 11 of static inclusion instead of the usual one given in Definition 4, as stated in Section 5.

Before proving the main result of this section (Proposition 4), we start by introducing the notion of a recipe without detour, and we prove a property on such recipes that will be useful later on.

Lemma 6. *Let ϕ be a frame, and R be a recipe without detour. If $R\phi\downarrow$ is a message, then R is a* simple recipe*, i.e. a recipe made of constructors on top of destructors.*

Proof. We show this result by structural induction on R.

Base case: $R = \mathsf{w}$ for some $\mathsf{w} \in dom(\phi)$. In such a case, the result trivially holds.

Inductive cases: $R = \mathsf{f}(R_1, R_2)$ with $\mathsf{f} \in \{\mathsf{senc}, \langle\ \rangle\}$. Since $R\phi\downarrow$ is a message, we have that $R_1\phi\downarrow$ and $R_2\phi\downarrow$ are both messages. We apply our induction hypothesis, and we conclude that R_1 (resp. R_2) are made of constructors on top of destructors, and this allows us to conclude that $R = \mathsf{f}(R_1, R_2)$ is also made of constructors on top of destructors.

Now, we consider the case where $R = \mathsf{sdec}(R_1, R_2)$. We have that $R_1\phi\downarrow = \mathsf{senc}(u_1, u_2)$ and $R_2\phi\downarrow = u_2$ for some messages u_1 and u_2. We apply our induction hypothesis on R_1, and R_2. Moreover, since u_2 is an atom (key position), we have that R_2 is destructor-only. Now, to conclude, it remains to show that R_1 is destructor-only. Assume by contradiction that R_1 is not destructor-only, it means that R_1 starts with the symbol senc, and this will contradict the fact that R is without detour. The case where $R = \mathsf{proj}_i(R')$ (with $i \in \{1, 2\}$) can be done in a similar way. This allows us to conclude. □

Proposition 4. *Let ϕ_1 and ϕ_2 be two frames. We have that :*

$$\phi_1 \sqsubseteq_s \phi_2 \text{ if, and only if, } \phi_1 \sqsubseteq'_s \phi_2.$$

Proof. We show the two implications separately.
($\Rightarrow$) The implication $\phi_1 \sqsubseteq_s \phi_2 \Rightarrow \phi_1 \sqsubseteq'_s \phi_2$ is easy. The only non trivial part is to establish that item 1 from Definition 11 holds. Let R be a ϕ_1-precompact recipe such that $R\phi_1\downarrow$ is an atom. Let $R' = \mathsf{senc}(R, R)$. We have that $R'\phi_1\downarrow \in \mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N})$. Therefore, relying on the fact that $\phi_1 \sqsubseteq_s \phi_2$, we deduce that $R'\phi_2\downarrow \in \mathcal{T}_0(\Sigma_\mathsf{c}, \Sigma_0 \cup \mathcal{N})$. This implies that $R\phi_2\downarrow$ is an atom since $R\phi_2\downarrow$ occurs in key position, and this allows us to conclude.

($\Leftarrow$) We now want to establish that $\phi_1 \sqsubseteq'_s \phi_2 \Rightarrow \phi_1 \sqsubseteq_s \phi_2$. We show that all the tests can be transfered from ϕ_1 to ϕ_2. A test is either a recipe R or a pair (R_1, R_2) of two recipes. In order to define our measure, we consider:

1. the number of steps needed to reduce R (resp. both R_1 and R_2) to their normal form using the following rewriting rules, and considering an innermost derivation:
$$\mathsf{sdec}(\mathsf{senc}(x, y), z) \Rightarrow x \quad \mathsf{proj}_1(\langle x, y\rangle) \Rightarrow x \quad \mathsf{proj}_2(\langle x, y\rangle) \Rightarrow y$$
2. the number of constructor symbols involved in R (resp. both R_1 and R_2)
3. the number of pair symbols at top level in $R\phi_1\downarrow$ (resp in $R_1\phi_1\downarrow = R_2\phi_1\downarrow$).

We then consider the lexicographic order.

We now distinguish two cases following the definition of static equivalence.

Case 1: Let R be a recipe such that $R\phi_1\downarrow$ is a message. We have to show that $R\phi_2\downarrow$ is a message.

First, we assume that R can be reduced using one of the rule given above (say $\mathsf{sdec}(\mathsf{senc}(x, y), z) \Rightarrow x$, the other cases being similar), and we consider an innermost redex. Therefore, we have that

$R = C[\mathsf{sdec}(\mathsf{senc}(R_1, R_2), R_3)]$, and we consider $R' = C[R_1]$. We know that $R\phi_1\!\downarrow$ is a message, and thus we have that $R_i\phi_1\!\downarrow$ is a message ($i \in \{1, 2, 3\}$). Moreover, we have that $R_2\phi_1\!\downarrow = R_3\phi_1\!\downarrow$. According to our measure, the test $R_2 = R_3$ is smaller than R, thus we have that $R_2\phi_2\!\downarrow$ and $R_3\phi_2\!\downarrow$ are both messages, and $R_2\phi_2\!\downarrow = R_3\phi_2\!\downarrow$. We have also that $R' = C[R_1]$ is smaller than R. Thus, we have that $R'\phi_2\!\downarrow$ is a message. Therefore, we have that

$$R\phi_2\!\downarrow = C[\mathsf{sdec}(\mathsf{senc}(R_1, R_2), R_3)]\phi_2\!\downarrow = C[R_1]\phi_2\!\downarrow = R'\phi_2\!\downarrow.$$

This allows us to conclude that $R\phi_2\!\downarrow$ is a message.

Now, we assume that R can not be reduced using one of the three rules above. We have that R is a recipe without detour. Since $R\phi_1\!\downarrow$ is a message, thanks to Lemma 6, we know that R is a simple recipe. Let $R = C[R_1, \ldots, R_n]$. In such a case, we have that $R\phi_1\!\downarrow = C[R_1\phi_1\!\downarrow, \ldots, R_n\phi_1\!\downarrow]$, and therefore, we know that $R_i\phi_1\!\downarrow$ ($i \in \{1, \ldots, n\}$) are messages (and actually there are atoms when they occur in key position). Assuming that C is not empty, we have that R_i ($i \in \{1, \ldots, n\}$) are smaller than R, and we deduce that $R_i\phi_2\!\downarrow$ ($i \in \{1, \ldots, n\}$) are messages (and actually atoms when they occur in key position). Therefore, we deduce that $R\phi_2\!\downarrow = C[R_1\phi_2\!\downarrow, \ldots, R_n\phi_2\!\downarrow]$ is a message.

Now, we assume that C is empty. In such a case, we have that R is destructor-only. We know that $R\phi_1\!\downarrow$ is a message. In case this message is not a pair, we therefore have that R is ϕ_1-precompact and, since $\phi_1 \sqsubseteq_s' \phi_2$, we deduce that $R\phi_2\!\downarrow$ is a message. Now, assume that $R\phi_1\!\downarrow = \langle u_1, u_2\rangle$. Let $R_1 = \mathsf{proj}_1(R)$ and $R_2 = \mathsf{proj}_2(R)$. We have that R_1 (resp. R_2) is smaller than R, and $R_i\phi_1\!\downarrow$ ($i \in \{1, 2\}$) is a message. Therefore, we can apply our induction hypothesis and we conclude that $R_i\phi_2\!\downarrow$ ($i \in \{1, 2\}$) is a message. Therefore, we have that $R\phi_2\!\downarrow = \langle R_1\phi_2\!\downarrow, R_2\phi_2\!\downarrow\rangle$ is a message.

Case 2: Let R and R' be two recipes such that $R\phi_1\!\downarrow$ and $R'\phi_1\!\downarrow$ are messages, and $R\phi_1\!\downarrow = R'\phi_1\!\downarrow$.

First, in case R (resp. R') can be reduced using one of the rule given above, we can apply the same reasoning as before to build a recipe $\overline{R}$ (smaller than R) such that $R\phi_2\!\downarrow = \overline{R}\phi_2\!\downarrow$. Since the test $\overline{R} = R'$ is smaller than $R = R'$, we have that $\overline{R}\phi_2\!\downarrow = R'\phi_2\!\downarrow$. This allows us to conclude that $R\phi_2\!\downarrow = R'\phi_2\!\downarrow$.

Now, we assume that R and R' can not be reduced using oneof the three rules above. We have that R and R' are recipes without detour. Since $R\phi_1\!\downarrow$ (resp. $R'\phi_1\!\downarrow$) is a message, thanks to Lemma 6, we know that R (resp. R') is a simple recipe. Let $R = C[R_1, \ldots, R_n]$ and $R' = C'[R'_1, \ldots, R'_{n'}]$. In such a case, we have that $R\phi_1\!\downarrow = C[R_1\phi_1\!\downarrow, \ldots, R_n\phi_1\!\downarrow]$ and $R'\phi_1\!\downarrow = C'[R'_1\phi_1\!\downarrow, R'_{n'}\phi_1\!\downarrow]$. Assuming that C and C' are not empty, we have that $C = \mathsf{f}(C_1, C_2)$ and $C' = \mathsf{f}'(C'_1, C'_2)$, and actually $\mathsf{f} = \mathsf{f}'$. Therefore, we have that

$$C_1[R_1, \ldots, R_n] = C'_1[R'_1, \ldots, R'_{n'}] \text{ (resp. } C_2[R_1, \ldots, R_n] = C'_2[R'_1, \ldots, R'_{n'}])$$

is a test that holds in ϕ_1 (both are smaller that $R = R'$). We deduce that $C_i[R_1\phi_2\!\downarrow, \ldots, R_n\phi_2\!\downarrow] = C'_i[R'_1\phi_2\!\downarrow, \ldots, R'_{n'}\phi_2\!\downarrow]$ with $i \in \{1, 2\}$. This allows us to conclude that $C[R_1\phi_2\!\downarrow, \ldots, R_n\phi_2\!\downarrow] = C'[R'_1\phi_2\!\downarrow, \ldots, R'_{n'}\phi_2\!\downarrow]$, i.e. $R\phi_2\!\downarrow = R'\phi_2\!\downarrow$.

Now, we assume that C is empty. In such a case, we have that R is destructor-only. In case $R\phi_1\!\downarrow = \langle u_1, u_2\rangle$. Let $R_l = \mathsf{proj}_1(R)$ and $R_r = \mathsf{proj}_2(R)$. We have that R_l (resp. R_r) is smaller than R, and $R_l\phi_1\!\downarrow$ and $R_r\phi_1\!\downarrow$ are both messages. Since $R\phi_1\!\downarrow = R'\phi_1\!\downarrow = \langle u_1, u_2\rangle$, we consider two cases:

1. R' is destructor-only (and C' is empty). In such a case, let $R'_l = \mathsf{proj}_1(R')$ and $R'_r = \mathsf{proj}_2(R')$. We have that R'_l (resp. R'_r) is smaller than R', and $R'_l\phi_1\!\downarrow = u_1$ and $R'_r\phi_1\!\downarrow = u_2$ are both messages. Therefore, we have that $R_l = R'_l$ (resp. $R_r = R'_r$) is a test that holds in ϕ_1 and that is smaller than $R = R'$, and this allows us to conclude that $R_l\phi_2\!\downarrow = R'_l\phi_2\!\downarrow$ (resp. $R_r\phi_2\!\downarrow = R'_r\phi_2\!\downarrow$), and thus $R\phi_2\!\downarrow = R'\phi_2\!\downarrow$.

2. $C' = \langle C'_l, C'_r \rangle$. In such a case, we have that $R_l = C'_l[R'_1, \ldots, R'_{n'}]$ (resp. $R_r = C'_r[R'_1, \ldots, R'_{n'}]$) is a test that holds in ϕ_1 and that is smaller than $R = R'$. Therefore, we have that $R_l\phi_2\downarrow = C'_l[R'_1\phi_2\downarrow, \ldots, R'_{n'}\phi_2\downarrow]$ and $R_r\phi_2\downarrow = C'_r[R'_1\phi_2\downarrow, \ldots, R'_{n'}\phi_2\downarrow]$. This allows us to conclude that $\langle R_l, R_r \rangle = \langle C'_l[R'_1, \ldots, R'_{n'}], C'_r[R'_1, \ldots, R'_{n'}] \rangle$ is a test that holds in ϕ_2. Therefore, we have that $R\phi_2\downarrow = R'\phi_2\downarrow$.

Now, it remains to consider the case where $R\phi_1\downarrow$ is not a pair, i.e. $R\phi_1\downarrow$ is either an atom or a ciphertext. In case $R\phi_1\downarrow$ is an atom, then $R'\phi_1\downarrow$ is an atom too, and relying on our induction hypothesis, we have that both R and R' are destructor-only, and we conclude relying on our hypothesis, i.e. $\phi_1 \sqsubseteq'_s \phi_2$. Now, assume that $R\phi_1\downarrow = \mathsf{senc}(u_1, u_2)$. In case both C and C' are empty, then we easily conclude relying on our hypothesis since R and R' are ϕ_1-precompact recipes. Therefore, we assume that $R' = \mathsf{senc}(R'_1, R'_2)$. In such a case, we consider the test $\mathsf{sdec}(R, R'_2) = R'_1$ which is smaller than $R = R'$ and that holds in ϕ_1. Hence, we have that $\mathsf{sdec}(R, R'_2) = R'_1$ holds in ϕ_2, and this allows us to conclude that $R\phi_2\downarrow = \mathsf{senc}(R'_1, R'_2)\phi_2\downarrow = R'\phi_2\downarrow$. □